SOCS

A COMPUTATIONAL LOGIC MODEL FOR THE DESCRIPTION, ANALYSIS AND VERIFICATION OF GLOBAL AND OPEN SOCIETIES OF HETEROGENEOUS COMPUTEES

IST-2001-32530

Deliverable D8: A computational approach to (societies of) computees

Project number:	IST-2001-32530
Project acronym:	SOCS
Document type:	D (deliverable)
Document distribution:	I (internal to SOCS and PO)
CEC Document number:	IST32530/DIPISA/012/D/I/b1
File name:	2012-b1[D8].pdf
Editor:	P. Mancarella, F. Toni, A. Bracciali
Contributing partners:	All
Contributing workpackages:	WP3
Estimated person months:	53
Date of completion:	29 January 2004
Date of delivery to the EC:	31 January 2004
Number of pages:	229

ABSTRACT
This document presents the computational counterparts for the abstract models for computees and their societies presented in deliverables D4 [63] and D5 [94], respectively, during the first year of the project. The companion deliverable D9 [5] presents a prototype demonstrator obtained by realising the computational models presented here.

Deliverable D8:
A computational approach to (societies of) computees

ICSTM:
Ulle Endriss, Fariba Sadri, Francesca Toni
DIPISA:
Andrea Bracciali, Paolo Mancarella, Giacomo Terreni
UCY:
Neophytos Demetriou, Antonis Kakas, Pavlos Moraitis
CITY:
Kostas Stathis
UNIBO:
Paola Mello, Paolo Torroni
DIFERRARA:
Evelina Lamma, Marco Gavanelli

ABSTRACT
This document presents the computational counterparts for the abstract models for computees and their societies presented in deliverables D4 [63] and D5 [94], respectively, during the first year of the project. The companion deliverable D9 [5] presents a prototype demonstrator obtained by realising the computational models presented here.

Contents

Part I
Introduction and Background

Abstract. This part of the document sets the scene for the document and gives the necessary technical background for the computational models presented in the document. It also summarises the task allocation within the consortium, that has led to the document itself.

1 Introduction

In deliverable D4 [63] we reported a detailed model for logic-based computees (computational entities). These are autonomous entities that are situated in a global computing environment and are equipped with the functionalities necessary for such an environment. They can have goals that they need to achieve, can plan for their goals, have a repertoire of actions, including physical, sensing and communication actions. They pursue their goals while being alert to the environment and adapt their goals and plan to any changes that they perceive. Their behaviour is governed by logic-based cycle theories allowing a highly modular specification of control, adaptable to dynamic environments and allowing heterogeneous operational characteristics for computees with different cycle theories.

In a companion deliverable, D5 [94], we described in detail a model, also logic-based, for societies of computees. The model allows the society to have its own goals and knowledge bases for achieving the goals and for recording the events that happen in the society. In addition, societies have a set of integrity constraints and protocols for governing the behaviour of their member computees. These constraints and protocols place expectations on the computees inhabiting the society. The society provides mechanisms for monitoring whether its members satisfy these expectations or whether they violate them.

The work in the second year of the project concentrated on formulating the computational model of computees and societies (workpackage WP3) and on building a prototype implementation for them (workpackage WP4). This document reports the work of the first such workpackage in the second year. The companion deliverable D9 [5] reports on work on the second such workpackage in the second year of the project.

The document is organised as follows. In the remainder of this Part I we provide brief summaries of the formal models of computees and societies of computees useful for setting the scene for the document. Further details on these models and on extensions and modifications to them during the second year are discussed extensively within Part II (presenting the computational model for the individual computee) and Part III (presenting the computational model for the society). We conclude the introductory part with some background on abductive logic programming (needed both in Part II and in Part III), on reasoning with dynamic priorities in logic programming (needed in Part II), and on constraint logic programming (needed both in Part II and in Part III).

The computational model for the individual computee is presented in Part II. After a recap of the (revised and updated) formal model from D4, the computational model is illustrated top-down: starting from the computational model of the main cycle of the computee, up to the computational models of the single components that are orchestrated by the cycle. The abductive proof procedures introduced, on which the computational model is based, are also presented. Then, soundness properties of the overall computational model are stated, building

on the properties of the lower layers of the model. Part II is concluded by an example, a discussion of related work and final remarks.

The computational model for societies of computees is presented in Part III. After a recap of the (revised and updated) formal model from D5, an interpretation of it in terms of Abductive Logic Programming is given. This leads to the society proof procedure, that is the society computational model. The introduced proof procedure is explained into details and its correctness properties stated. Part III is concluded by a discussion of related work and final remarks.

A summary and evaluation of the work presented in this document appears in Part IV, while the proofs of all (non-trivial) results in the document are sketched in the appendices in Part V.

2 Computee formal model: short recap (and revisions)

Here we briefly summarise the main structure of the computee formal model defined in deliverable D4 [63]. The formal model consists of

- an internal (or mental) state of the computee,
- a set of reasoning capabilities of the computee, supporting planning, temporal reasoning, identification of preconditions of actions, reactivity and goal decision,
- a sensing capability of the computee,
- a set of formal transition rules for the state of the computee defined in terms of the above capabilities,
- a set of selection functions, to provide appropriate inputs to the transitions,
- a set of execution cycles for the combination of the transitions and the selection functions, as provided by the cycle theory of the computee.

The internal (or mental) state of a computee is a triple $\langle KB, Goals, Plan\rangle$, where:

- KB is the knowledge base of the computee, and describes what the computee knows (or believes) of itself and the environment. KB consists of modules supporting different reasoning capabilities:
  - KB_{plan}, for Planning and for the Identification of Preconditions of actions,
  - KB_{TR}, for Temporal Reasoning,
  - KB_{GD}, for Goal Decision,
  - KB_{react}, for Reactivity, and
  - KB_0, for holding the (dynamic) knowledge of the computee about the external world in which it is situated and about past (and expected) states of the environment (including happened communications).

  Syntactically, KB_{plan},KB_{react} and KB_{TR} are abductive logic programs with constraint predicates, sharing a kernel set of definitions and integrity constraints, KB_{GD} is a logic program with priorities over rules, and KB_0 is a set of facts (definite clauses) in logic programming, and it is included in all the other modules.

- *Goals* is the set of properties that the computee wants to achieve, each one is equipped with a temporal constraint defining when it is expected to hold. There are two types of goals, mental goal and sensing goal. Mental goals can be observed to hold (or not to hold) via the Sensing capability or they can be brought about actively by the computee by its planning and its actions. Sensing goals can only be checked in the environment via the Sensing capability and cannot be brought about actively by the computee.

  Goals form two trees, whose roots are represented by $\perp^{nr}$ (for the non-reactive goals) and $\perp^{r}$ (for the reactive goals). All the top-level (non-reactive) goals are either assigned to the computee by its designer at birth, or they are determined by the Goal Decision capability.

  The tree structure is updated by appropriate calls to the Planning and Reactivity capabilities.

- *Plan* is a set of actions scheduled in order to satisfy goals. The actions are partially ordered. Each is equipped with a temporal constraint about when it is expected to be executed. Each action is also equipped with the preconditions for its successful execution, determined by the Identification of Preconditions capability. Actions may be physical actions, communicative actions, or sensing actions. Actions are added to the tree structure for *Goals* by Planning and Reactivity capabilities, which identify their parent in the tree-structure. Actions always occur as leaves in such a tree structure. Similarly to goals in *Goals*, actions in *Plan* may be reactive or non-reactive, depending on their parents (and their ancestors).

The model of the computee is called the KGP model, for short. The reasoning capabilities employed within the KGP model are:

- Planning, which generates partial plans for sets of goals. It provides (temporally constrained) sub-goals and actions designed for achieving the input goals. All partial plans generated are guaranteed to be overall consistent with KB (and the existing *Goals* and *Plan*).

- Reactivity, which reacts to perceived changes in the environment, by replacing reactive goals in *Goals* and actions in *Plan* with (possibly temporally constrained) goals and actions. A non-empty new set of reactive goals and actions will only be generated if overall consistent with the non reactive components of the prior $\langle KB, Goals, Plan\rangle$.

- Goal decision, which continuously revises the top-most level goals of the computee, adapting computee's strategy to changes in its own preferences and in the environment. Differently form Reactivity, it only modifies the top-level (reactive and non-reactive) goals of a computee, does not add actions to *Plan* and does not depend upon the current *Goals* and *Plan* of the computee.

- Identification of Preconditions, which identifies the preconditions for action execution that can be used by a computee to reason about its *Goals* and *Plan*.

- Temporal Reasoning, which reasons about the environment, and makes predictions about properties holding in the environment, based on the partial information a computee may acquire.

In addition to the reasoning capabilities above, the computee is equipped with a Sensing capability which links the computee to its environment.

The state of a computee evolves by applying transition rules, which employ capabilities and a notion of Constraint Satisfaction. The transitions are:

- Goal Introduction, which changes the top-level $Goals$, and uses Goal Decision.
- Plan Introduction, which changes $Goals$ and $Plan$, and uses Planning and Introduction of Preconditions and Constraint Satisfaction.
- Reactivity, which changes $Goals$ and $Plan$, and uses Reactivity and Constraint Satisfaction.
- Sensing Introduction, which changes $Plan$ by introducing new sensing actions for checking the preconditions of actions already in $Plan$, and uses Sensing.
- Passive Observation Introduction, which changes KB_0 of KB by introducing unsolicited information coming from the environment, and uses Sensing.
- Active Observation Introduction, which changes KB_0 of KB, by introducing the outcome of (actively sought) sensing actions, and uses Sensing.
- Action Execution, which executes all types of actions, thus changing KB_0 of KB.
- Goal Revision, which revises $Goals$, and uses Temporal Reasoning and Constraint Satisfaction.
- Plan Revision, which revises $Plan$ and uses Constraint Satisfaction.

The behaviour of a computee is given by the application of transitions in sequences, which change the state of the computee. In the KGP model, these sequences are not determined by fixed cycles of behaviour, as in conventional agent architectures, but rather by reasoning with cycle theories. These are logic programs with priorities over rules, defining preference policies over the order of application of transitions, which may depend on the environment and the internal state of a computee. This provision of a declarative control for computees in the form of cycle theories is a highly novel feature of the model, which could, in principle, be imported into other agent systems. Both the state of computees, their reasoning capabilities and their control (cycle) theories are formulated and realised within computational logic. We envisage that this will ease the task of formulating and verifying properties of computees in later stages of the project (WP5).

The main technical revisions to the computee model presented in the D4 submitted to the Commission [63] will be illustrated when describing the (computational counterparts of the) various components. Here, let us note that these revisions amount to the following:

- Goals in the state are separated into non-reactive, arising from the Goal Introduction (GI) transition, and reactive, arising from the Reactivity (RE) transition. Similarly, actions in the state are separated into non-reactive, if they belong to (partial) plans for non-reactive goals, and reactive, arising from the Reactivity transition or belonging to (partial) plans for reactive goals. This allows us to avoid replication of work between GI and RE and overlapping between KB_{GD} and KB_{react}.

- The capabilities have been rewritten to improve readibility and correct typos. Moreover, Reactivity has been rewritten, so that it does not make use of Temporal Reasoning any longer, in order to avoid some unwanted behaviors with the earlier definition and to ease the implementation in WP4.
- The transitions GI and RE have been rewritten to take into account the changes to the state (two trees), and the changes to the underlying capabilities. Also, GI has been greatly simplified, so that i does not keep plans for goals that are kept. In particular, this had the aim to ease the implementation in WP4 by avoiding the introduction of integrity constraint checking.
- The transitions of Goal revision (GR) and Plan Revision (PR) have been simplified.
- The core selection functions have been simplified.
- The notion of Operational Trace, defining the behaviour of the computee as given by reasoning with its Cycle Theory, has been generalised.

Further details on these revisions will be given later on in the document.

3 Society formal model: short recap (and revisions)

The society model developed in the first year of the project and presented in Deliverable D5 [94] is capable of modelling interactions among computees in an open environment, and also supports goal-directed behaviour of societies of computees.

The model of a society is represented by the following 4-tuple:

$$\langle SOKB, SEKB, ICS, GS \rangle$$

where:

- $SOKB$ is the Social Organisation Knowledge Base,
- $SEKB$ is the Social Environment Knowledge Base,
- ICS is the set of Social Integrity Constraints, and
- GS is the set of Goals of the society.

The Social Environment Knowledge Base ($SEKB$) records occurred events and (positive or negative) expectations about social events. The $SEKB$ is a dynamic knowledge base for representing events perceived by the society, and the expectations on the observable behaviour of computees at society level. In particular, the $SEKB$ dynamically evolves and is composed of:

- Observable and relevant events for the society (happened events: atoms indicated with functor H); we name this set HAP;
- Expectations about the future: events that should (but might not) happen in the future (atoms indicated with functor E, which we called positive expectations), and events that should not (but might indeed) happen in the future (atoms indicated with functor NE, which we called negative expectations); we name this set EXP.

The Social Organisation Knowledge Base ($SOKB$) provides specification for society's goals. In $SOKB$, clauses may contain in their body positive and negative expectations (possibly negated) about the behaviour of computees, and auxiliary literals (positive and negative), while their heads are atoms which possibly correspond to society's goals. Goals (GS) of the society have the same syntax as the body of $SOKB$ clauses.

Finally, Social Integrity Constraints (ICS) are forward rules, of kind $body \Rightarrow head$, which have in their *body* literals, (possibly negated) conditions about happened events and ((possibly negated) positive and negative) expectations, and in their head (disjunctions of) conjunctions of positive and negative (possibly negated) expectations. Furthermore, CLP-like (Constraint Logic Programming) constraints can occur in the body of $SOKB$ clauses, and in the body and head of constraints in ICS. For details about the formal syntax of the society tuple components, and variable quantification in particular, the reader can refer to D5 [94].

The main features achieved by the society model are:

(i) the use of computational logic to model and give semantics to interactions; and

(ii) the use of a uniform formalism (based upon what we called Social Integrity Constraints) for expressing both interaction protocols and social semantics of communication languages.

The society infrastructure is devoted to check compliance of the behaviour of the members within a society, with respect to expectations of the society as prescribed by the social integrity constraints and required to achieve the society's goals. Compliance to the specified protocols of interactions, communication language semantics and required behaviour to achieve certain goals can be checked by a suitable computational logic-based proof procedure, as discussed in part III. We envisage that grounding on computational logic both the society model and its operational counterpart will ease the task of formulating and verifying properties of societies of computees in later stages of the project (WP5).

4 Computational models for computee and society: general approach

In the second year, two approaches appeared possible and were investigated for the single computee:

1. to design a unique, extended proof procedure in charge of supporting all the computational power of a computee, namely its capabilities, transitions and cycle theory,

2. to extend (whenever necessary) and integrate existing proof procedures within a modular framework, afforded by the KGP and the society models, for coordinating the different reasoning processes that a computee may exhibit, and also integrating them within the cycle theory.

After an evaluation of pros and cons of the two approaches, the second one (which indeed is the one more adherent to the SOCS project original proposal) has been chosen, mainly because of the following considerations:

- **Modularity.** A design of the computational model of a computee based on the integration of proof procedures, according to the component based structure of the computee given by its formal KGP model, is expected to better support modularity than the one based on a general single proof procedure. Each integrated proof procedure may be more effectively used and tuned to support the specific reasoning feature for which it has been developed. Moreover, possible future extensions of the whole model, as, for instance, increasing the set of computee capabilities, can be easily supported at the computational model level, by the integration of suitable proof procedures for extensions.

- **Possibility of Reuse.** A number of different proof procedures have been partially developed, supporting the reasoning capabilities of a computee. Formal correctness results for these procedures are available, which could serve as a starting point for correctness of the computational model with respect to the abstract specification given in D4. Several of these computational models are also supported by implementations that are (i) freely available, since developed by academic institutions, often with the contribution of members of the SOCS consortium, (ii) provided with code, which can hence be used as a starting point for the possibly required extensions, and (iii) sometimes, tested and debugged by experimentation. It appears, therefore, convenient and wise, to try to exploit as much as possible this large body of existing work, both within WP3 and WP4. Similarly for the society.

The language of the KGP model for a computee is that of Abductive Logic Programming and Logic Programming with Priorities. The computational model for computees is then based on proof procedures that realise the underlying abductive and preference reasoning of these two frameworks. Based on these we have built the different components (capabilities, transitions, cycle theory etc) of the KGP model.

The language of societies is also that of Abductive Logic Programming. The computational model for societies is based on an abductive proof procedure which extends the same core procedure (IFF [47]) used by the computee. We have considered using a single proof procedure both for the (abductive components of the) computee and for the society. However, the two models (computee and society) differ considerably in the kind of abductive tasks that they encompass (as we will discuss later on), and thus, again for the sake of modularity, we have opted for two different extensions.

5 Task allocation

The allocation of tasks for WP3 during the second year is shown in the table below. Each task concerns investigation of the available candidate proof procedures, proposal of the actual proof procedures to be used and investigation of any modifications and extensions needed to the chosen proof procedures. The allocation of these tasks has been done coherently with the WP4 allocation task plan, shown in the companion deliverable D9, as it is reasonable that the same partners take primary responsibility for both the definition and the implementation of each computational model they are allocated, as summarised in the table below.

	ICSTM	DIPISA	CITY	UCY	UNIBO	DIFERRARA
Planning	X	X				
Identification of Preconditions	X	X				
Reactivity	X	X				
Goal Decision				X		
Temporal Reasoning		X		X		
Transitions	X	X	X			
Selection Functions	X	X				
Cycle Theory	X			X		
Proof procedure for ALP	X	X				
Proof procedure for LLP				X		
Generator of expectations					X	X
Checker of expectations					X	X
Society proof procedure					X	X

Figure 1: Task Allocation for WP3 in the second year

6 Background

6.1 Abductive logic programming and abductive proof procedures

Abduction has been widely recognised as a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge [29, 43, 70, 68, 69, 67]. Incomplete knowledge is handled by labeling some pieces of information as abducibles, i.e., possible hypotheses which can be assumed, provided that they are "consistent" with the given knowledge base. Abductive Logic Programming (ALP) enriches with abduction standard logic programming based upon SLD-resolution. Integrity constraints are often used in ALP to further restrict the set of potential hupotheses.

The computee model that we have presented in deliverable D4 heavily relies upon ALP for many of its components (namely its planning, reactivity and temporal reasoning capabilities). Moroever, the society model we have presented in deliverable D5 also has a natural interpretation in ALP terms, as we will see in part III, section 18. In this section we briefly revise syntactic and semantic ALP notions, overview a number of existing proof procedures for ALP, and describe in some detail the IFF procedure that we extend and modify to provide computational counterparts to the (abductive components of the) computee model and the society model. Some of the background given below is adapted from deliverables D4, D5 and D7.

6.1.1 Abductive logic programming

An *abductive logic program* is a triple $\langle P, A, IC \rangle$ where:

- P is a *normal logic program*, namely a set of rules (clauses) of the form

  $$H \leftarrow L_1, \ldots, L_n$$

  with H atom, $L_1, \ldots, L_n$ (positive or negative) literals, and $n \geq 0$. Each negative literal L_i is of the form *not* B_i, where B_i is an atom. The negation symbol *not* indicates *negation as failure* [26, 89, 7]. All variables in H, L_i are implicitly universally quantified, with scope the entire rule. H is called the *head* and $L_1, \ldots L_n$ is called the *body* or the *conditions* of a rule of the form above. If $H = p(t)$, for some vector of terms t, the rule is said to *define* p.

  A *definite logic program* is a set of *definite clauses*, namely rules without any occurrence of negation as failure,

  A *fact* is a clause of the form above but with $n = 0$.

- A is a set of *abducible predicates*, p, such that p is a predicate in the language of P which does not occur in the head of any clause of P (without loss of generality [69]),

- IC is a set of *integrity constraints*, that is, a set of sentences in the language of P.

In this document, integrity constraints will have the form

$$L_1, \ldots, L_n \Rightarrow false \quad (n > 0)$$

conventionally called *denials*, or [1]

[1] If $n = 0$, then $L_1, \ldots, L_n$ represents the special atom $true$.

$$L_1, \ldots, L_n \Rightarrow H \quad (n \geq 0, H \neq false)$$

or

$$L_1, \ldots, L_n \Rightarrow A_1 \vee \ldots \vee A_m \quad (n \geq 0, m > 1)$$

that we call *implicative integrity constraints* or simply *implications*, where L_i are positive or negative literal, H, A_j are atoms, and $false$ is a special atom in the language of the abductive logic programs. In these integrity constraints, $L_1, \ldots, L_n$ is referred to as the *body* (or *conditions*) and $H, A_1 \vee \ldots \vee A_m$ are referred to as the *head* (or *conclusions*) of the constraint. [2]

All variables in the integrity constraints are implicitly universally quantified from the outside (as for rules), except for variables occurring only in the head which are implicitly existentially quantified with scope the head. In part III, we will define the concept of *social integrity constraints* which are implications with a more liberal quantification on variables.

Abductive answers

Given an abductive logic program $\langle P, A, IC \rangle$ and a formula (*query/observation*) Q, which is a conjunction of literals in the language of the abductive logic program, the purpose of abduction is to find a (possibly minimal) set of atoms Δ in the abducible predicates A which, together with P, "entails" (an appropriate ground instantiation of) Q, with respect to some notion of "entailment" that the language of P is equipped with, and such that the extension of P "satisfies" a given set of (consistent) integrity constraints IC (see [69] for possible notions of integrity constraint "satisfaction"). Here, the notion of "entailment" depends on the semantics associated with the logic program P (there are many different choices for such semantics, as well-documented in the Computational Logic literature). More formally and concretely, given an existentially quantified conjunction of literals (query) Q, a set of abducible atoms Δ, and a variable substitution θ for the variables in Q, the pair (Δ, θ) is an *abductive answer* for Q, with respect to an abductive logic program $\langle P, A, IC \rangle$, iff

1. $P \cup \Delta \models_{LP} Q\theta$, and
2. $P \cup \Delta \models_{LP} IC$,

where $\models_{LP}$ is a chosen semantics for logic programming. In deliverable D4, we have left the choice of $\models_{LP}$ open, and our models of computees and societies could actually work with any choice for $\models_{LP}$. In this document we will choose $\models_{LP}$ to be the (three-valued) *completion semantics* [26, 84]. This choice is dictated by the choice of the IFF abductive proof procedure for ALP, which is at the heart of the computational models for the single computee (and the society) that we will present later on.

Given a logic program P, the *completion of predicate* p in the language of P, for which there are k definitions in P ($k > 0$):
$p(t_1) \leftarrow B_1$
$\ldots$
$p(t_k) \leftarrow B_k$
is the iff-definition

[2] Note that commas in the body of integrity constraints and rules represent conjunction, and, in the sequel, are sometimes written as $\wedge$ instead.

$p(X) \leftrightarrow D_1 \vee \ldots \vee D_k$

where D_i is $X = t_i, B_i$. The variables in each D_i different from X are implicitly existentially quantified within D_i. The variables X are implicitly universally quantified from the outside.

Moreover, the *completion of predicate p* in the language of P for which there are no definitions in P is

$p(X) \leftrightarrow false$

The *completion of a logic program* P, indicated as $Comp(P)$, is the set of all completions of predicates in P. The *selective completion of a logic program* P wrt a set of predicates S, referred to as $Comp_{\overline{S}}(P)$ is the union of the completions of all predicates occurring in P but not occurring in S.

6.1.2 Abductive proof procedures

We have considered a number of existing abductive proof procedures (see also D7), mentioned and briefly described in the following, and focused in particular on the IFF proof procedure [47] (Section 6.1.3), which, among the others, turned out to be most appropriate for our purposes. For a full survey of the alternative proof procedures we considered, see [91].

In [70] Kakas and Mancarella define a proof procedure (here referred to as **KM proof procedure**) for ALP, as an extension of that in [43]. This procedure assumes that the integrity constraints are in the form of *denials*, each with at least one abductible literal in the conditions. The semantics of such integrity constraints within the KM proof procedure requires that at least one of the literals in the integrity constraint does not hold. The procedure starts from a query and a set of initial assumptions Δ_i and results in a set of consistent hypotheses (abduced literals) Δ_o such that $\Delta_o \supseteq \Delta_i$ and Δ_o together with the program P entails the query. The proof procedure uses the notion of *abductive* and *consistency derivations*. Intuitively, an abductive derivation is a standard SLD-derivation suitably extended in order to consider abducibles. As soon as an abducible atom δ is encountered which does not already occur in the current set of hypotheses, it is added to the current set of hypotheses, and it must be proved that any integrity constraint such that δ unifies with an abducible in its is satisfied. For this purpose, a consistency derivation for δ is started. Since the integrity constraints are denials only (i.e., queries), this corresponds to proving that every such query fails to hold. Therefore, δ is removed from all the denials with which it unifies, and we prove that all the resulting queries fail. In this consistency derivations, when an abducible is encountered, an abductive derivation for its complement is started in order to prove the abducible's failure, so that the initial integrity constraint is satisfied.

The original KM proof procedure can only deal integrity constraints in the form of denials. Moreover, it can only deal with abducibles which are ground at selection time, and *flounders* if a selected abducible is not ground. Finally, it treats *constraint predicates*, such as $<, \leq, \neq, \ldots$, as ordinary predicates, thus being unable to use specialised constraint solvers for such predicates. Some extensions of this procedure have been proposed to (partially) cope with these limitations.

The **Active-KM proof procedure** [90] is an extension of the KM-procedure that integrates in the original abductive computational scheme a limited but powerful type of implicative-form integrity constraints. As for the IFF proof procedure (described in the following), it supports forward reasoning via integrity constraints (implications) which fire when their conditions (body) are satisfied. However, differently from IFF, this procedure cannot deal with non-ground

abducibles. An important aspect of the Active-KM is the lower computational complexity than the IFF due to its *depth-first* behaviour (which, on the other hand, requires backtracking and does not allow a complete view of the frontier of the computation).

Abdual [6] is a systems for performing abduction from extended logic programs adopting the well-founded semantics. It handles only ground programs, and relies on tabled evaluation inspired to SLG resolution [25].

Among other abductive proofs existing in literature, we cite here the abductive query evaluation method proposed by Satoh and Iwayama in 1992 [106].

Abductive proof procedures dealing with variables in abducibles and constraint predicates

Some works have proposed abductive proof procedures dealing with non-ground abducibles and constraint predicates, which are relevant for our purposes as we will need to generate hupotheses consisting of possibly non-ground actions whose time is constrained (in the single computee case) and expectations at future times, again constrained (in the society case). In the following, we survey some of them.

ACLP [72] is a recent abductive proof procedure extending the original KM procedure to deal with non-ground abduction and with constraints. ACLP programs can contain constraints on finite domains. ACLP interleaves consistency checking of abducible assumptions and constraint satisfaction.

[32] introduces a proof procedure for normal abductive logic programs by extending SLDNF resolution to the case of abduction. The resulting proof procedure (**SLDNFA**) is correct with respect to the completion semantics. A crucial property of this abductive procedure is the treatment of *non-ground* abductive queries. [32] does not consider general integrity constraints, but only constraints of the kind $a, not\ a \Rightarrow false$. To overcome this limitation, a later work [33] considers the treatment of general integrity constraints but in a quite inefficient way. In practice, all the integrity constraints are checked at the end of the proof for a query, i.e., only when the overall set of abductive hypotheses supporting the query has been computed. More recent work is represented by the **SLDNFA(C)** system [118] which extends SLDNFA with constraint handling.

The **$\mathcal{A}$-system** [76] is a followup of ACLP and SLDNFA(C). $\mathcal{A}$-system differs from the previous two for the explicit treatment of non-determinism that allows the use of heuristic search with different types of heuristics. $\mathcal{A}$-system is able to perform non-ground abduction. But a derivation flounders if universally quantified literals appear in a denial in the derivation. However, according to [117], the system could be extended in order to perform non-ground abduction with universally quantified literals by storing Herbrand equalities as constraints. To date, the $\mathcal{A}$-system lacks a formal semantics and formal correctness results.

Another recent proof-procedure dealing with non-ground abducibles is presented in [47], referred to as **IFF**. This uses backward reasoning with the selective completion of the given logic program (namely its completion, but only with respect to the non-abducible predicates) to compute abductive explanations for given queries. The queries are conjoined with the integrity

constraints at the beginning of the abductive process, and forward reasoning with them is applied. The integrity constraints do not need to be denials, but can be any (closed) implications. The authors describe the IFF-procedure as a sort of "hybrid of the proof procedure of Console et al. [28] and the SLDNFA procedure of Denecker and De Schreye (see [32])" mainly both for its use of the (selective) Clark completion semantics [26] and for not requiring a safe selection rule for abducibles and negation.

This procedure is proposed as the engine underlying the reasoning mechanism of intelligent agents in [80]. An extension of this procedure to deal with constraint predicates is given in [78]. An extension of this procedure to deal with negation as failure in integrity constraints is proposed in [104].

Amongst all the abductive proof procedures mentioned above, we decided to use and extend IFF as the kernel abductive proof procedure for computees and societies, because of its treatment of variables and integrity constraints, as well as for the already existing experimentations of its use for agents [80, 81, 104, 105].

6.1.3 The IFF proof-procedure

IFF is based upon a simple system of *rewrite rules* (or *rewrite steps* or *inference rules*). Its computation starts with an *initial formula* built as the conjunction of the initial query and the integrity constraints IC. The variables in the initial query are referred to as *free*. Then it repeatedly generates (derived) formulas by applying one of the following inference rules:

unfolding, which applies backward reasoning (resolution) with the selective completion with respect to A of the logic program P (indicated as $Comp_{\overline{A}}(P)$);

propagation, which reasons forward with the integrity constraints IC;

splitting, which distributes conjunctions and disjunctions, putting the resulting formula in a (sum-of-products) disjunctive normal form;

case analysis, which is applied to an integrity constraint whose body contains an equality $X = t$, with certain quantification restrictions, and non-deterministically analyses the alternative cases $X = t$ and $X \neq t$, generating a disjunction in the resulting formula;

factoring, which tries to reuse previously made hypotheses;

rewrite rules for equality, which use the inferences in Clark's Equality Theory (CET) [26];

logical simplifications, which simplify formulas through logical equivalences such as $A \wedge false \equiv false$, $[true \Rightarrow A] \equiv A$, etc.

A sequence of rewrite steps can be thought of as building a tree, whose root is the initial formula, and whose frontier (after the application of splitting) is a disjunction of conjunctions. The disjuncts in any frontier are referred to as nodes. In general, each node is a conjunction of atoms and implications. E.g., a frontier can look like:

$$\begin{array}{ll} & (A_1 \wedge A_2 \wedge [B_1 \wedge B_2 \Rightarrow A_3] \wedge [B_3 \wedge B_4 \Rightarrow A_4]) \\ \vee & (A_i \wedge A_j \wedge A_k \wedge [B_y \Rightarrow A_z] \wedge [B_5 \Rightarrow false]) \end{array}$$

In general, each conjunction in a node is one of the following

- an atom
- an implication, with the same syntax as the integrity constraints, but with some of the variables occurring in them possibly free or existentially quantified within the node (existentially quantified variables may be introduced by unfolding with the selective completion).

[47] assume that their abductive logic programs and queries are *allowed*, to simplify the treatment of the quantifiers, avoiding to write them explicitly in derivations.

1. **IFF-Allowed Definitions.** A definition of the form
$$p(X_1, \ldots, X_n) \leftrightarrow D_1 \vee \ldots \vee D_k,$$
in $Comp_{\overline{A}}(P)$ is (IFF) allowed if, for each D_i, every variable distinct from $X_1, \ldots, X_n$ occurring in D_i also occurs in a positive non-equality atom in D_i.
2. **IFF-Allowed Integrity Constraints.** An integrity constraint of the form
$$L_1, \ldots, L_n \Rightarrow A_1 \vee \ldots \vee A_m$$
is (IFF) allowed if every variable occurring in the conclusion $A_1 \vee \ldots \vee A_m$ occurs in the conditions $L_1, \ldots, L_n$.
3. **Queries.** A query of the form
$$L_1 \wedge \ldots \wedge L_n,$$
is (IFF) allowed if every variable in any L_i occurs in a positive, non-equality atom L_j.

Thanks to these allowedness properties, given a frontier, the quantification of the variables in it is implicitly given as follows:

if a variable is in the initial query, then it is *free*;

else if it occurs in an atom, it is existentially quantified with scope the whole frontier;

else (if it occurs only in an implication) it is universally quantified with scope the implication.

In the IFF, negative literals which are conjuncts in nodes in a frontier are always thought of and rewritten as implications (e.g., *not* A is rewritten as $A \Rightarrow false$). Negative literals in the body of an implication are moved to the head (e.g., *not* $A \wedge B \Rightarrow C$ is rewritten as $B \Rightarrow A \vee C$).

A sequence of rewrite steps is referred to as a *derivation* starting from the initial formula (frontier) and terminating with the frontier obtained after having applied the last rewrite step in the sequence. A node in the last frontier in a derivation is referred to as a *leaf node* if no further rewrite step can be applied to it. A leaf node is called *failure leaf node* if it is $false$, *non-failure leaf node* otherwise. A *successful derivation* is a derivation to a frontier with (at least) one non-failure leaf node. A *(finitely) failed derivation* is a derivation to $false$. *Abductive answers* can be extracted from non-failure leaf nodes.

Below, we present the rewrite rules of the IFF in detail, as we will use them and extend/modify them later on when presenting the computational models for (some of the) capabilities in the computee model and the computational counterpart of the society.

While presenting the set of inference rules of the IFF, we assume that they are applied to a frontier F_i in a derivation, starting from the frontier F_0 which is given by the initial formula $Q \wedge IC$. Note that, in F_0, variables in IC are renamed so that they are distinct from those in Q. The application of the inference rules generate a new frontier F_{i+1}.

Unfolding

Given an atom $p(t_1, \ldots, t_k)$ and
$p(X_1, \ldots, X_n) \leftrightarrow D_1 \vee \ldots \vee D_k \ \in \ Comp_{\overline{A}}(P)$
then, given $\Theta = \{X_1/t_1, \ldots, X_n/t_n\}$:

- if the atom is a conjunct in a node in F_i, then F_{i+1} is F_i with the atom replaced by $(D_1 \vee \ldots \vee D_k)\Theta$;
- if the atom is a conjunct in the body of an implication $p(t_1, \ldots, t_n), B \Rightarrow C$[3] which is a conjunct in a node in F_i then F_{i+1} is F_i with the implication replaced by the new implications:

  $[D_1\Theta, B \Rightarrow C] \wedge \ldots \wedge [D_n\Theta, B \Rightarrow C]$.

□

Propagation

Given an atom $p(s_1, \ldots, s_n)$ and an implication $p(t_1, \ldots, t_n), B \rightarrow C$ both conjuncts in the same node N in F_i, then F_{i+1} is F_i with the new implication
$t_1 = s_1, \ldots, t_n = s_n, B \Rightarrow C$
added as a new conjunct of the node N. The propagation rule cannot be applied if the latter implication is already in F_i.

□

Splitting

- Given a formula of the form

  $[C \vee D] \wedge C'$

  in F_i, then F_{i+1} is F_i with the formula replaced by:

  $[C \wedge C'] \vee [D \wedge C']$.

- In addition, given a formula of the form

  $[B \Rightarrow [H \vee R]] \wedge C$

  where H is a single atom which contains no universally quantified variables and R can be empty (equivalent to $false$), then F_{i+1} is F_i with the formula replaced by:

  $[H \wedge C] \vee [[B \Rightarrow R] \wedge C]$.

□

[3]The position of $p(t_1, \ldots, t_n)$ in the body of the implication is not important because any atom in a body of an implication can be selected for unfolding. This is valid also for the other inference rules.

Case analysis

Given a node N in F_i, of the form

$[B, X = t \Rightarrow A] \wedge C$

where X is free or existentially quantified, t does not contain X and t is not an universally quantified variable, F_{i+1} is F_i with the node N replaced by:

$[X = t \wedge [B \Rightarrow A] \wedge C] \vee [X \neq t \wedge C]$.

Case analysis can be applied only to equality atoms. Each variable in t which is universally quantified becomes existentially quantified in the first disjunct (case $X = t$) and remains universally quantified in the second disjunct (case $X \neq t$) [4].

□

Factoring

Given a node N in F_i, of the form

$a(t_1, \ldots, t_n) \wedge a(s_1, \ldots, s_n) \wedge C$

where a is abducible (namely $a \in A$), then F_{i+1} is F_i with the node N replaced by:

$[a(t_1, \ldots, t_n) \wedge a(s_1, \ldots, s_n) \wedge [t_1 = s_1, \ldots, t_n = s_n \Rightarrow false] \wedge C] \vee$
$[a(t_1, \ldots, t_n) \wedge t_1 = s_1 \wedge \ldots \wedge t_n = s_n \wedge C]$.

□

Rewrite rules for equality

This inference rule is a set of equality rules derived from the unification algorithm of Montanari and Martelli [92], and can be applied both to an equality atom which occurs as a conjunct in a node and to an equality atom which occurs in the body of an implication. The rules are the following:

1. $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$ is replaced by $t_1 = s_1 \wedge \ldots \wedge t_n = s_n$.
2. $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_n)$ is replaced by $false$ if f and g are distinct.
3. $t = t$ is replaced by $true$ for any term t.
4. $X = t$ is replaced by $false$ if the term t contains X.
5. $t = X$ is replaced by $X = t$ if X is a variable and t is not.
6. $Y = X$ is replaced by $X = Y$ if X is universally quantified and Y is not.
7. If $X = t$ occurs as a conjunct in a node and t does not contain X then apply the substitution X/t to the entire node, retaining the conjunct $X = t$ intact.
8. If $X = t$ occurs in the body of an implication, t does not contain X and X is universally quantified, then apply the substitution X/t to the entire implication, deleting the conjunct $X = t$.

□

[4] Here and in the remainder of this document the disequality $a \neq b$ will stand for $a = b \Rightarrow false$.

Logical equivalences

This last inference rule is a set of logical rules used to simplify the form of a node. The rules are the following:

1. $C \wedge false \equiv false$
2. $D \vee false \equiv D$
3. $C \wedge true \equiv C$
4. $D \vee true \equiv true$
5. $[true \Rightarrow D] \equiv D$
6. $[false \Rightarrow D] \equiv true$
7. $not\, B \equiv [B \Rightarrow false]$ *(Negative atom elimination)*
8. $[C \wedge not\, B \Rightarrow D] \equiv [C \Rightarrow B \vee D]$ *(Negative body elimination)*

□

Note that a frontier F_i in a derivation $F_0, \ldots, F_i, F_{i+1}, \ldots, F_n, \ldots$ obtained by a step of unfolding, case analysis or factoring, will be in non-disjunctive normal form. In such cases, though, to keep the format of frontiers simple, we assume that splitting is applied immediately to obtain a next frontier F_{i+1} which is in disjunctive normal form.

Also, note that the IFF does not rely on any specific selection strategy to build derivations (e.g. a selection rule can always choose the leftmost node in a frontier giving rise to depth-first search of the tree), but it assumes that such selection strategy is *fair*.

Extracted (abductive) answers by IFF

To extract an abductive answer from a non-failure leaf node N in a formula F_n in a derivation, we build a substitution σ' such that:

- σ' replaces all variables in N which are not universally quantified by ground terms and
- σ' satisfies the equalities and disequalities in N.

Given σ' (there exists at least one substitution σ' because the rewrite rules for equality have been exhaustively applied to N), an *extracted (abductive) answer* to the given query Q is a pair (Δ, σ) where Δ is the set of all abducible atoms that are conjuncts in $N\sigma'$ and σ is the restriction of σ' to the variables occurring in Q.

By construction, it is trivial to see that $Q\sigma' = Q\sigma$ and $IC\sigma' = IC\sigma$. Moreover CET implies all the equalities and disequalities in $N\sigma'$ and all the implications in $N\sigma'$ which are not disequalities are implied by $Comp(\Delta)$ because N is a leaf node and so propagation has been exhaustively applied to it.

Hence, it holds that

1. $Comp(\Delta) \cup CET \models N\sigma'$.

Moreover, it is straightforward that $Comp_{\overline{A}}(P) \cup CET \models F_0 \leftrightarrow F_n$ by construction, and so:

2. $Comp_{\overline{A}}(P) \cup CET \models N \rightarrow [Q \wedge IC]$ and
3. $Comp_{\overline{A}}(P) \cup CET \models N\sigma' \rightarrow [Q \wedge IC]\sigma'$.

By means of 1. and 3. we have that:
$Comp(P \cup \Delta) \cup CET \models N \rightarrow Q\sigma$ and
$Comp(P \cup \Delta) \cup CET \models N\sigma' \rightarrow IC$.

Hence, (Δ, σ) is an abductive answer for Q and is the minimal one in the sense that, by construction, is the smallest set of abducible ground atoms such that $Comp_{\overline{S}}(\Delta) \cup CET \models N\sigma'$, where S is the set of all non-abducible predicates.

The following theorems state more formally soundness and completeness results obtained by the IFF-procedure. Their proofs are sketched in [47] and given completely in [46].

Theorem 6.1 (Soundness). *Let Q be a given query with respect to an abductive logic program $\langle P, A, IC \rangle$.*

1. *Let (Δ, σ) be extracted from a nonfailure node N in a derivation from $Q \wedge IC$. Then (Δ, σ) is an abductive answer to the query Q. Furthermore, if (Δ', σ) is any answer such that $Comp_{\overline{S}}(\Delta') \models N\sigma'$ where σ' is the auxiliary substitution used to construct σ, then Δ is a subset of Δ'.*

2. *If there exists a derivation from $Q \wedge IC$ to $false$ then*
   $Comp_{\overline{A}}(P) \cup CET \cup IC \models Q \leftrightarrow false$.

□

Theorem 6.2 (Completeness). *Let Q be a given query with respect to an abductive logic program $\langle P, A, IC \rangle$. If (Δ, σ) is an abductive answer to Q, then there exists a derivation from $Q \wedge IC$ of a frontier containing a nonfailure leaf node from which an answer (Δ, σ) to Q can be extracted such that Δ is a subset of Δ'.*

□

Theorem 6.1 holds both if $\models$ is logical consequence in three-valued logic and if $\models$ is logical consequence in two-valued logic. Instead, theorem 6.2 holds only if $\models$ is logical consequence in three-valued logic. To have a completeness result also for two-valued logic, theorem 6.2 is weakened in [47].

In the reminder of this document, we will define extensions of the original IFF both for dealing with the abductive components of the single computee model (planning, reactivity, identification of preconditions, temporal reasoning) and for defining the computational counterpart of the society model. In the context of the single computee (part II), the choice of the IFF naturally gives the choice of the completion semantics as the underlying logic programming semantics $\models_{LP}$. This choice will be further discussed later on.

6.2 Logic programming with priorities

A computee is an autonomous entity that operates in an open and dynamic environment where it needs to take decisions related to its different capabilities like problem solving, cooperation, communication, etc, under complex preference policies. As such one of its underlying reasoning processes is that of preference reasoning $\models_{pr}$ in the context of (a specific framework for) Logic Programming with Priorities (LPP), where priorities express preferences. This supports directly the computee's goal decision capability as well as the definition of cycle theories of the computee.

In the KGP model, $\models_{pr}$ is realised in the concrete framework of $LPwNF$ (Logic Programming without Negation as Failure) of logic programs and priorities.

Reasoning with priorities in logic programming has received a lot of attention over the last years partly motivated by problems in multi-agent systems as in our case (another major source of motivation has been legal reasoning). These works include [20, 21, 97, 23, 55, 83, 77]. One of the main differences btween our approach with $LPwNF$ and approaches in the literature is that our framework does not contain negation as failure (NF) in its object level language. This in many cases facilitates a more modular representation since all the non-monotonic reasoning in the formalism is treated in a uniform way via priorities.

6.2.1 The $LPwNF$ framework

In this section we give a brief overview of $LPwNF$ and motivate its computational model that will be presented in section 10.3. This section is adapted from the background of D4.

$LPwNF$ [71, 34, 74] is a logic programming framework for preference reasoning with an argumentation-based semantics. Within this framework we have a powerful form of decision making under a given preference policy. This policy can, as required by the Global Computing setting in which we are working, be sensitive to different information in a dynamic environment allowing the decision making to adapt to the particular circumstances at the time of the decision.

The preference reasoning within $LPwNF$ is based on a model of argumentation where local priority information, given at the level of the rules of a theory (or policy), is lifted to give a global preference over sets of rules that compose arguments and counter arguments for a certain decision. A theory or policy within $LPwNF$ is viewed as a pool of sentences or rules from which we need to select a suitable subset, i.e. an argument, in order to support a conclusion.

In $LPwNF$ knowledge is represented in a classical background logic $(\mathcal{L}, \models_H)$ by means of rules of the form

$$L \leftarrow L_1, \ldots, L_n, \quad (n \geq 0)$$

where $L, L_1, \ldots, L_n$ are positive or negative (classical) literals. A negative literal is a literal of the form $\neg A$, where A is an atom. As usual in logic programming a rule containing variables is a compact representation of all the ground rules obtained from this under the Hebrand universe. Each ground rule has a unique name.

Note that the symbol $\neg$ stands for classical negation, and should not be confused with the symbol *not* used to represent negation as failure above. Indeed, no negation as failure occurs in an $LPwNF$ theory [5]. Note also that, differently from rules in conventional logic programs as given above, here the head L of rules may be negative as in the framework of Extended Logic Programming.

The background derivability $\models_H$ relation of the framework is the monotonic Horn logic given by the single inference rule of modus ponens, treating negative literals as ordinary atoms. In general, we can separate out an *auxiliary part* of a given theory from which the other rules can draw background information in order to satisfy some of its conditions. The reasoning of the auxiliary part of a theory is independent of the main argumentation-based preference reasoning of the framework and hence any appropriate logic can be used.

[5] This is where the framework gets its name: Logic Programming without Negation as Failure but the historical reasons for this name are not important in this document.

Some of the sentences in a $LPwNF$ theory express priorities over the rules of the theory. These have the same form as the rules above except that their head, L, refers to a *higher-priority relation,* h_p, and so such a rule has the general form

$$L = h\_p(rule1, rule2) \leftarrow L_1, \ldots, L_n, \quad (n \geq 0)$$

where $rule1$ and $rule2$ are the names of two rules in the theory.

A rule of this form then means that under the conditions $L_1, \ldots, L_n$, $rule1$ has priority over $rule2$. The role of this priority relation is therefore to encode locally the relative strength of (argument) rules in the theory. The priority relation given by h_p is required to be irreflexive. The rules $rule1$ and $rule2$ can in fact be themselves rules expressing priority between other rules and hence the framework allows higher-order priorities. For simplicity of presentation we will assume that the conditions of any rule in the theory do not refer to the predicate h_p thus avoiding self-reference problems.

The preference reasoning of $LPwNF$ uses the priority relation between rules to find out conclusions that are preferred over their conflicting conclusions. Indeed, an $LPwNF$ theory might give rise, under its background derivability $\models_H$, to *conflicts*, namely between a literal L and its negation $\neg L$. More generally, we can define other forms of conflict within a given theory through an auxiliary predicate *incompatible* and rules of the form

$$incompatible(L_1, L_2) \leftarrow B,$$

stating that literals L_1 and L_2 are conflicting under some (auxiliary) conditions B (see sections 11.5 and 8 for examples of this). Also for any ground atom $h\_p(rule1, rule2)$, its (unique) conflicting literal is defined to be $h\_p(rule2, rule1)$ and vice-versa.

Conflicts together with the priority relation of the theory give rise to a notion of *attack* between sets of sentences in the theory $\mathcal{T}$. Δ attacks Δ', where Δ, Δ' are sets of sentences in the $\mathcal{T}$, means that these two sets have conflicting conclusions (under $\models_H$) and that the rules of Δ that derive this are rendered by Δ to have at least the same priority as the priority that Δ' renders for its own rules that derive the conflicting conclusion. This definition is given as follows (see [74] for more details when h_p is not static).

Let $\mathcal{T}$ be an $LPwNF$ theory and $\Delta, \Delta' \subseteq \mathcal{T}$. Then Δ' *attacks* Δ (or Δ' is a *counter argument* of Δ) iff there exists L, $\Delta_1 \subseteq \Delta'$ and $\Delta_2 \subseteq \Delta$ such that:

(i) $\Delta_1 \vdash_{min} L$ and $\Delta_2 \vdash_{min} \overline{L}$

(ii) $(\exists r' \in \Delta_1, r \in \Delta_2 \text{ s.t. } \Delta_2 \models_H h\_p(r, r')) \Rightarrow (\exists r' \in \Delta_1, r \in \Delta_2 \text{ s.t. } \Delta_1 \models_H h\_p(r', r))$,

where $\overline{L}$ is any literal that conflicts L (e.g. $\overline{L} = \neg L$ or $incompatible(L, \overline{L})$ holds).

Here $\Delta \vdash_{min} L$ means that $\Delta \models_H L$ and that L cannot be derived from any proper subset of Δ. The second condition in this definition states that an argument Δ' for L attacks an argument Δ for the contrary conclusion only if the set of rules that it uses to prove L are at least of the same strength (under the priority relation h_p) as the set of rules in Δ used to prove the contrary. Note that the attacking relation is not symmetric.

Given this notion of attack that lifts the priority relation from individual rules to sets of rules, we define the *admissible* subsets or arguments of a given theory as follows.

Let $\mathcal{T}$ be a theory and $\Delta \subseteq \mathcal{T}$. Then Δ is an *admissible* argument iff Δ is consistent (i.e. conflict free) and for any $\Delta' \subseteq \mathcal{T}$ if Δ' attacks Δ then Δ attacks Δ'.

6.2.2 Preference reasoning in $LPwNF$

Preference reasoning is based on the maximal admissible arguments of a given theory. Usually, two entailment relations are defined, given an $LPwNF$ theory $\mathcal{T}$:

- $\mathcal{T} \models_{pr}^{cred} G$ means that there is at least one maximal admissible subset of $\mathcal{T}$ where G holds;
- $\mathcal{T} \models_{pr}^{skep} G$ means $\mathcal{T} \models_{pr}^{cred} G$ and, for any $\overline{G}$ such that $incompatible(G, \overline{G})$ holds, $\mathcal{T} \not\models_{pr}^{cred} \overline{G}$.

In the sequel, $\models_{pr}$ will indicate $\models_{pr}^{skep}$.

Additional details on $\models_{pr}$ can be found in [71, 34, 74].

As we will see below in section 10.3 it is possible to build a computational model for $LPwNF$ directly from its argumentation based semantics following a standard method for computing argumentation. In [34] a specific proof procedure has been give to compute $LPwNF$ in the special case where the priority relation is static. This interleaves the argumentation reasoning with the process of computing the attacks from the background theory and hence it is difficult to generalize. We will see that in order to develop a more general computational model where the priority relation is dynamic it is easier to abstract the problem and follow a proof theory approach for argumentation as proposed in [75].

6.3 Constraint Logic Programming

Constraint Logic Programming (CLP) [61] extends logic programming with *constraints predicates* which are not processed as ordinary logic programming predicates, defined by rules, but are checked for satisfiability and simplified by means of a built-in, "black-box" constraint solver. These are typically used to constrain, together with unification which is also treated via an equality constraint predicate, the values that variables in the conclusion of a rule can take. For example, constraints can be used to compute the value of time variables, in goals and actions, under a suitable temporal constraint theory (as is the case in our computee model).

The CLP framework is defined over a particular structure $\Re$ consisting of domains $D\Re$ and a set of constraint predicates which includes equality, together with an assignment of relations on $D\Re$ for each such constraint predicate. In CLP, constraints are built as first-order formulae in the usual way from primitive constraints of the form $c(t_1, \ldots, t_n)$ where c is a constraint predicate symbol and $t_1, \ldots, t_n$ are terms constructed over the domain, D($\Re$), of values. Then the rules of a constraint logic program P take the same form as rules in conventional logic programming given by

$$H \leftarrow L_1, \ldots, L_n, C$$

where C is a set of constraints.

A *valuation* θ of a set of variables is a mapping from these variables to the domain D($\Re$) and the natural extension which maps terms to D($\Re$). A valuation θ, on the set of all variables appearing in a set of constraints C, is called an $\Re$-solution of C iff $C\theta$, obtained by applying θ to C, is satisfied i.e. $C\theta$ evaluates to true under the given interpretation of the constraint predicates and terms. The set C is called *($\Re$-)solvable* or *($\Re$-)satisfiable* iff it has at least one $\Re$-solution.

One way to give the meaning of a constraint logic program P is to consider the grounding of the program over its Herbrand base and all possible valuations, over D($\Re$), of its constraint

variables. In each such rule if the ground constraints C are evaluated to true then the rule is kept with the condition of C dropped, otherwise the whole rule is dropped. The meaning of P is then given by the meaning of the resulting logic program.

The frameworks of ALP and LLP described above can be usefully extended with constraints in the same way that CLP extends logic programming.

In ALP together with this extension of constraining the conclusions of the rules we can use constraints to constrain abducible assumptions. In fact the link with constraints allows us to extend an ALP framework to include non-ground abducible hypotheses, an extension that increases significantly the versatility of applying abductive reasoning to various problems and that plays an important role within the model we propose in this document.

One such framework of integration of ALP and CLP is that of $ACLP$ [72, 3] and the $\mathcal{A}$-system [76] that has followed it. In this framework abductive theories are defined as usual but now over the combined language of a given underlying framework of D($\Re$) and a user given language for the problem domain that is represented by the abductive logic program. The essential extension is that now the set of abducible atoms in the predicates of A is extended to consist of the following formulae:

1. $a(d)$, where a is an abducible predicate and d is a vector of constants in D$\Re$ (such abducibles are called ground abducibles)

2. $\exists X(a_1(X), .., a_n(X), C(X))$, where $n \geq 1$, a_i is an abducible predicate $\forall i. 1 \leq i \leq n$ and $C(X)$ is a (possibly empty) set of constraints.

The subset Δ of abducibles of an abductive answer can now include also non-ground abducibles, e.g. $\Delta = \exists X(a(X), C(X))$. The notion of an *abductive answer* is build on the previous definition for (ground) abduction generalising this by considering the different possible groundings of non-ground elements of Δ allowed by the constraints C that they involve. Given an abductive logic program and a query Q then a set Δ of abducibles is an abductive answer for Q iff:

1. there exists a valuation (or grounding) g such that, for each abducible formula ϕ in Δ, the constraints in ϕg are ($\Re-$)satisfied, and

2. for any such grounding g, Δg is a ground abductive explanation of Q.

Hence Δ is an abductive solution for a query Q iff there exists (in $\Re$) at least one consistent grounding of Δ and for any such grounding g, Δg constitutes a ground abductive solution of Q. Note that essentially this grounding g gives also the associated variable substitution θ for the variables in the goals Q, since in CLP the implicit unification of logic programming is replaced by an explicit equality theory that is always part of the constraint theory $\Re$ of the CLP framework.

Operationally, an abductive answer can be computed, in CLP fashion, by interleaving the generation (and consistency checking with respect to the integrity constraints) of abducible hypotheses together with the generation of an associated constraint store C of constraints on these hypotheses and its own check for satisfiability. This constraint store can grow during the computation provided that it remains satisfiable. A "black box" constraint solver, that is transparent to the abductive reasoning itself, is used to decide the satisfiability of this store when necessary and to reduce the constraints accordingly. The satisfiability or not of C in turn affects the computation of abductive hypotheses and their check of satisfying the integrity

constraints. This is the approach that we will take in this document when defining the C-IFF and $\mathcal{S}$CIFF proof procedures.

In this document, when clear from the context, we will sometimes refer to both integrity constraints in abductive logic programming and atoms defined in terms of constraint predicates (also called CLP constraints) simply as constraints.

CLP and the KGP model

In part II, we will denote by $\models^3_{\Re}$ truth (in three-valued logic) together with solvability over valuations in $\Re$. E.g. $Comp(P \cup \Delta) \models^3_{\Re} Q$, for some set of (ground) abducibles Δ, means that there exists a valuation θ of the variables in the query Q, such that $Q\theta$ is solvable in $\Re$ and $Q\theta$ is true in (the three-valued completion of) $P \cup \Delta$. Moreover, for any given set of constraints C, $\models_{\Re} C$ will stand for C is $\Re$-solvable. [6]

Furthermore, we will assume a correct and complete computational counterpart $\vdash_{\Re}$ of $\models_{\Re}$, that we will refer to as *constraint solver*. In particular, we will assume that, for any (set of) temporal constraints TC, defined as in D4,

$\vdash_{\Re} TC$ if and only if there exists a (total) substitution σ such that $\sigma \models_{\Re} TC$.

[6] Namely, $\models_{\Re}$ is $\models$ as we used it in D4.

Part II
Computees

Abstract. This part of the document presents the computational counterpart of the computee formal model. This computational model is defined modularly, reflecting the modularity of the model itself, in terms of the computational models for the capabilities, the transitions, the selection functions and the cycle components of the model. The computational counterparts of the capabilities of planning, identification of preconditions, reactivity and temporal reasoning are defined in terms of a single, novel abductive proof procedure that extends an existing one. The computational counterparts of the goal decision capability and of cycle are defined in terms of a single, novel proof procedure for logic programming with priorities. The computational counterparts for transitions and selection functions are straightforward mappings from the formal definitions, and their correctness is a by-product of the correctness of the capabilities that are invoked by them.
The computational models for the components are presented as top-down, from cycle, to the selection functions and transitions, to the capabilities. The underlying proof procedures are given prior to the capabilities. The correctness results are then given in reverse, building upon the correctness of the underlying proof procedures. The proofs for the (non-trivial) results in this part are given in appendices B and C.

7 KGP model: recap

In this section we give an overview of the state of computees in the KGP model, and operations on it. Other components (capabilities, transitions, selection functions and cycle) are recapped later on in the document, within specific sections.

7.1 Preliminaries

In the KGP model we assume (possibly infinite) vocabularies of *time constants* (e.g., the set of all natural numbers), *time variables* (that we will indicate with $t, t', s, \ldots$), *fluents* (that we will indicate with $l, l', \ldots$), *action operators* (that we will indicate with $a, a', \ldots$), and *names of computees* that we will indicate with c, c', ... Given a fluent l, l and $\neg l$ are referred to as *fluent literals*. We will use $l, l', \ldots$ to denote fluent literals as well. Moreover, given a fluent literal l, by $\overline{l}$ we will denote its complement, namely $\neg f$ if l is f, f if l is $\neg f$.

We assume that the set of fluents is partitioned in two disjoint sets: *mental fluents* and *sensing fluents*. Intuitively, mental fluents represent properties such that the computee itself is able to plan for so that they can be satisfied, but can also be observed. On the other hand, sensing fluents represent properties which are not under the control of the computee and can only be observed by sensing the external environment. For example, $problem\_fixed$ and $get\_resource$ may represent mental fluents, namely the properties that (given) problems be fixed and (given) resources be obtained, whereas $request\_accepted$ and $connection\_on$ may represent sensing fluents, namely the properties that a request for (given) resources is accepted and that some (given) connection is active.

We also assume that the set of action operators is partitioned in three disjoint sets: *sensing action operators*, *physical action operators* and *communication action operators*. In the sequel, we will refer to physical and communication action operators jointly as *non-sensing action operators*. Intuitively, sensing actions represent actions that the computee performs in order to establish whether some fluents hold in the environment. These fluents may be *sensing* fluents, but they can also represent effects of actions that the computee may need to check in the environment. On the other hand, *physical* actions are actions that the computee performs in order to achieve some specific effect, which typically causes some changes in the environment. Finally, *communication* actions are actions which involve communications with other computees. For example, $sense(connection\_on, t)$ is a sensing action, aiming at checking whether or not the sensing fluent *connection_on* holds; $do(clear\_table, t)$ may be a physical action operator, and $tell(c_1, c_2, request(r1), d, t)$ may be a communication action expressing that computee c_1 is requesting from computee c_2 the resource r_1 within a dialogue d, at time t.

Temporal constraints

In the sequel we will use temporal constraints associated with goals and actions of a computee. Temporal constraints are formulae defined by the following syntax:

TC	::=	AtomicTC | TC $\wedge$ TC | TC $\vee$ TC | $\neg$ TC
AtomicTC	::=	Variable Relop Term
Relop	::=	$= \mid \neq \mid < \mid > \mid \leq \mid \geq$
Term	::=	Time_Constant | Time_Variable | Term Op Term
Op	::=	$+ \mid - \mid * \mid \div$

As mentioned above, time constants are simply natural numbers. Time variables are distinguished variables which can be instantiated to time points. Notice that no explicit quantification is introduced in temporal constraints: indeed, in a temporal constraint occurring in goals and actions (see below) all variables are implicitly existentially quantified.
In Section 10.1 we will introduce an extension of an existing abductive proof procedure in order to handle these type of temporal constraints. In that context we will denote the relational operators of the above syntactic category `Relop` by prefixing them with the symbol #. For instance, the equality constraint predicate symbol will be denoted by #=.

Goals

A goal G is a triple of the form $\langle l[t], G', Tc\rangle$ where

- $l[t]$ is the *fluent literal* of the goal possibly referring to the time t; we will refer to $l[t]$ as a *timed fluent literal*;
- G' is the *parent* of G;
- Tc is a (possibly empty) temporal constraint which typically refers to the time t.

Top-level goals are goals which have no parent. We will denote them by triples of the form $G = \langle l[t], \bot, Tc\rangle$, where $\bot$ can be either $\bot^r$ or $\bot^{nr}$. In the first case the goal will be referred to as top-level *non reactive* goal, in the second case it will be referred to as top-level *reactive* goal. Whenever it is not important to distinguish between reactiove and non-reactive goals, we will simply write $\bot$ instead of $\bot^r$ and $\bot^{nr}$.

As an example, we may have a top-level goal G of the form

$$\langle problem\_fixed(p2, t), \bot, 5 \leq t \leq 10 \rangle$$

and a subgoal G' of G of the form

$$\langle get\_resource(r1, t'), G, 5 \leq t' \leq t \rangle$$

meaning that to fix problem $p2$ within a certain time interval, the computee needs to have (or acquire) a resource $r1$.

In the sequel *mental goals* are goals whose fluent is mental, and *sensing goals* are goals whose fluent is sensing.

Actions

An action A is a 4-tuple of the form $\langle a[t], G, C, Tc \rangle$ where

- $a[t]$ is the *operator* of the action, referring to the execution time t; we will refer to $a[t]$ as a *timed (action) operator*;
- G the goal towards which the action contributes (i.e., the action belongs to a *plan* for the goal G). G may be a post-condition for A (but there may be other such post-conditions, as given within the knowledge base of the computee).
- C are the *preconditions* which should hold in order for the action to take place successfully; syntactically, C is a conjunction of timed fluent literals;
- Tc is a (possibly empty) temporal constraints which typically refers to the time t of the action.

In the next Section, we will introduce actions which a computee may perform when *reacting* to external stimuli. These actions are named *top-level reactive* actions and will be denoted by a 4-tuple $\langle a[t], \bot^r, C, Tc \rangle$. The meaning of the second component, $\bot^r$ will be clarified in the next Section 7.2. Notice however that, differently from ordinary actions where the second component is a goal, top-level reactive actions are not introduced as part of a plan to achieve some goal.

Note that we are assuming that actions are atomic and do not have a duration. The temporal constraints specify a time window over which the time of the action can be instantiated, at execution time. If the temporal constraints are empty then the action can be executed at any time.

As an example, we may have an action

$$\langle tell(c_1, c_2, request(r1), d, , t''), G', \{\}, 5 \leq t'' \leq t' \rangle$$

where G' is given as above.

Note that, in practice, actions can be seen as special kinds of goals which are directly *executable*. In the sequel *sensing actions* are actions whose action operator is a sensing action operator, *non-sensing actions* are actions whose action operator is a non-sensing action operator. We distinguish two types of sensing operators, denoted by $sense(l[t])$ and $sense\_precondition(l[t])$. The latter is a sensing action which a computee may explicitly introduce in its plan in order to check whether or not a precondition of a certain action holds at the current time.

Time variables

In both a timed fluent literal $l[t]$ and a timed operator $a[t]$, the time t may be a time constant (in which case the associated temporal constraint will be empty) or a time variable. This variable is treated as an existentially quantified variable, the scope of which is the whole *state* of the computee (see Section 7.2). Whenever a goal (resp. action) is introduced within a state, the time variable associated with the goal (resp. action), is a distinguished, fresh variable. When a time variable is instantiated (e.g., at action execution time) the actual instantiation is recorded in the state of the computee. This allows us to keep different instances of the same action (resp. goal) distinguished.

Finally, note that the temporal constraints of a goal/action might be empty even though the time of the goal/action is a variable.

7.2 State of a computee

At any given time, the state of a computee is a triple

$$\langle KB, Goals, Plan \rangle$$

where $Goals$ is a set of goals and $Plan$ is a set of actions, as defined in section 7.1.

The $Goals$ and $Plan$ components of the state can be represented as a tree where:

- the root is $\perp$;
- the nodes of the tree other than the root are labelled by goals in $Goals$ or actions in $Plan$;
- the children of the root are the top-level goals in $Goals$;
- actions can label only leaf nodes;
- for each non-leaf node labelled by a goal G, the children of the node are all the actions in $Plan$ and goals in $Goals$ whose parent is G.

As an example, the tree for the following $Goals$ and $Plan$ of computee c_1

$$\begin{aligned} Goals &= \{G, G_1, G_2\}, \text{ where} \\ \\ G &= \langle problem\_fixed(p2, t), \perp, 5 \leq t \leq 10 \rangle \\ G_1 &= \langle get\_resource(r1, t_1), G, 5 \leq t_1 \leq t \rangle \\ G_2 &= \langle get\_resource(r2, t_2), G, 5 \leq t_2 \leq t \rangle \\ \\ Plan &= \{\langle tell(c_1, c_2, request(r1), d, t'), G_1, \{\}, 5 \leq t' \leq t_1 \rangle\} \end{aligned}$$

is the tree in Figure 2.

It is worth pointing out that all variables occurring in the tree (and in particular the time variables t, t_1, t_2 and t' in the example above) are implicitly existentially quantified with scope the whole tree.

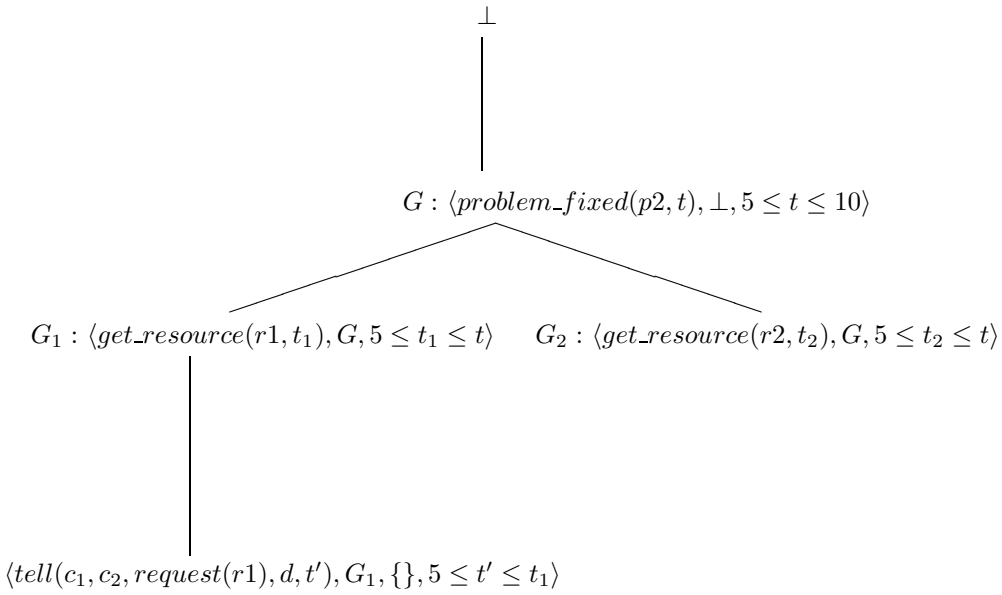


Figure 2: The tree view of the state of computee c_1

In the revised D4, we have introduced two slight extensions of the state, in order to handle reactivity, i.e. the ability of the computee to react to external stimuli and to its own internal decisions. Firstly, we distinguish between *ordinary* top-level goals, introduced by Goal Decision and *reactive* top-level goals introduced in the state because of reactions of the computee to external stimuli and internal decisions as represented within goals and plans. Each of these reactive goals will be represented as a tuple $\langle l[t], \bot^r, Tc \rangle$. Secondly, we will also have *top-level actions*, namely actions which are not introduced as part of a plan for a goal, but instead are introduced directly in reaction to some external stimuli. Top-level actions will be represented as tuples $\langle a[t], \bot^r, C, Tc \rangle$.

Given such extension, the state of the computee can be seen as a pair of trees. The first tree is the non-reactive component of the state and is constructed as sketched above. The second tree represents the reactive component of the state: the root of this tree is labelled by $\bot^r$ and the children of the root may be top-level (reactive) goals as well as top-level (reactive) actions.

For simplicity, we will assume that, given a state $\langle KB, Goals, Plan \rangle$, all occurrences of variables in $Goals$ and $Plan$ are time variables (implicitly existentially quantified globally within the $Goals$ and $Plan$). In other words, our goals and actions are ground except for the time parameters. In an extension, variables other than time variables in goals and actions can be dealt with in the same way as time parameters. We concentrate on time variables as time plays a fundamental role in our model. We avoid dealing with the other variables to keep the model simple.

The KB component of the state is the union of various (not necessarily disjoint) knowledge

bases. Among them we distinguish KB_0 which records the actions which have been executed and their time of execution as well as the properties (i.e. fluents and their negation) which have been observed and the time of the observation. Formally, KB_0 contains assertions of the form:

> $executed(a[t], \tau)$ where $a[t]$ is a timed operator and τ is a time constant. This sentence means that an action a has been executed at time $t = \tau$ by the computee.
>
> $observed(l[t], \tau)$ where $l[t]$ is a timed fluent literal and τ is a time constant. This fact means that the property l has been observed to hold at time $t = \tau$.
>
> $observed(c, a[\tau'], \tau)$ where c is a computee's name, different from the name of the computee whose state we are defining, τ and τ' are time constants, and a is an action operator. This sentence means that the given computee has observed at time τ that computee c has executed the action a at time τ' (of course $\tau' \leq \tau$).

Note that assertions in KB_0 of the third kind are variable-free. These are intended to represent reception of communication from other computees. These type of assertions have no variable as they represent actions executed by other computees, whose (internal) time variables are of no interest to the computee in question.

Instead, assertions of the first two kinds refer explicitly to the time t. This representation with explicit variables allows us to link the record in KB_0 of observed properties and execution of actions by the computee with the time of actions in $Plan$ and goals in $Goals$. Thus, the time of an action and of a goal serves implicitly as an identifier, uniquely identifying the action or goal (due to the assumption on the uniqueness of such variables, as indicated in section 7.1). Note that, as a consequence, the time variables in KB_0 are not properly speaking variables as such. Rather, they can be equated to "named variables" as in [3].

However, since KB_0 will be used in all the remaining knowledge bases that form KB, and these are represented in a logic programming style, we are not allowed to have assertions with existentially quantified variables. Hence, the various knowledge bases will include a *variant* of KB_0 containing:

- an assertion $executed(a[\tau])$, for each $executed(a[t], \tau) \in KB_0$
- an assertion $observed(l[\tau]$ for each $observed(l[t], \tau) \in KB_0$
- all assertions $observed(c, a[\tau'], \tau)$ in KB_0.

In other words, the variant of KB_0, which we still refer to as KB_0 by abuse of notation, contained in each knowledge base contains only variable-free facts, without referring explicitly to time variables.

Note that at this stage of the development of the KGP model we assume that the computee trusts its environment absolutely and therefore considers that the information in KB_0 is irrevocable.

7.3 Operations and notations on states

In D4 and here, we use extensively some useful notations and operations on a state

$$\langle KB, Goals, Plan \rangle$$

- First of all, we will refer to the union (conjunction) of all temporal constraints in all actions in $Plan$ and all goals in $Goals$ as TCS.
- Given $G \in Goals$ and $A \in Plan$ we write:
  - $parent(G) = G'$ if $G = \langle \_, G', \_ \rangle$ (G' can be $\bot$ or $\bot^r$ if G is a top-level goal) [7]
  - $parent(A) = G'$ if $A = \langle \_, G', \_, \_ \rangle$ (G' can be $\bot^r$ if A is a top-level action).
  - $children(G, Goals) = \{G' \in Goals \mid G' = \langle \_, G, \_ \rangle\}$
  - $descendants(G, Goals) = children(G, Goals) \cup$ $\{G' \in Goals \mid \exists G'' \in descendants(G, Goals).G' \in descendants(G'', Goals)\}$
  - $ancestor(G, Goals) = \{G' \in Goals | G \in descendants(G', Goals)\}$.
- We say that $GA, GA' \in Goals \cup Plan$ are *siblings* if $parent(GA) = parent(GA')$.

The following notations are introduced in order to distinguish the non-reactive and the reactive component of a state $\langle KB, Goals, Plan \rangle$.

- $Goals^r$ and $Plan^r$ denote the set of goals and actions in $Goals$ and $Plan$ which are either top-level reactive goals and actions, or descendants of top-level reactive goals, respectively. Namely,

$$\begin{array}{rcl} Goals^r & = & \{G \in Goals \,|\, G = \langle \_, \bot^r, \_ \rangle\} \ \cup \\ & & \{G \in Goals \,|\, G \in descendants(G', Goals) \text{ and } G' \in Goals^r\} \\ Plans^r & = & \{A \in Plan \,|\, A = \langle \_, \bot^r, \_, \_ \rangle\} \ \cup \\ & & \{A \in Plan \,|\, parent(A) \in Goals^r\} \end{array}$$

- Similarly, $Goals^{nr}$ and $Plan^{nr}$ denote the set of non-reactive goals and actions in $Goals$ and $Plan$, respectively. Namely

$$\begin{array}{rcl} Goals^{nr} & = & Goals \setminus Goals^r \\ Plan^{nr} & = & Plan \setminus Plan^r \end{array}$$

Valuation of temporal constraints

Given a state $S = \langle KB, Goals, Plan \rangle$, we will denote by $\Sigma(S)$ the valuation obtained as follows.

$$\Sigma(S) = \{t = \tau \,|\, executed(a[t], \tau) \in KB_0\} \cup \{t = \tau \,|\, observed(l[t], \tau) \in KB_0\}$$

When the state S we are referring to is clear from the context, we will write simply Σ instead of $\Sigma(S)$. Intuitively, Σ extracts from KB_0 the instantiation of the (existentially quantified) time variables in $Plan$ and $Goals$, derived from having executed (some of the) actions in $Plan$ and

[7] We use "_" to denote a component which is not relevant in the context, inheriting the Prolog tradition of denoting by "_" an anonymous variable.

having observed that (some of the) fluents in *Goals* hold (or do not hold). KB_0 provides a "virtual" representation of Σ.

Below, $\Sigma(t)$, for some time variable t, will return the value of t in Σ, if there exists one, namely, if $t = \tau \in \Sigma$, then $\Sigma(t) = \tau$.

The valuation of temporal constraints associated with goals or actions in a state S will always take Σ into account. Let Tc be a (set of) temporal constraint with (temporal) variables $t_1, \ldots, t_n, t_{n+1}, \ldots, t_m$, $m \geq n \geq 1$, such that:

- for each $i = 1, \ldots, n$, $\Sigma(t_i) = \tau_i$ for some time point τ_i;
- for each $i = n+1, \ldots, m$, there exists no τ_i such that $\Sigma(t_i) = \tau_i$.

Then, a *total* $\Sigma-$*valuation* σ for Tc is a valuation for the time variables in Tc which are not evaluated by Σ already, namely σ is a valuation $\{t_{n+1} = \tau_{n+1}, \ldots, t_m = \tau_m\}$, for some time points $\tau_{n+1}, \ldots, \tau_m$.

Given a temporal constraint Tc and a total $\Sigma-$valuation σ for Tc, we will write

$$\sigma \models_{\Re} Tc$$

if Tc is satisfied by the valuation $\sigma \cup \Sigma$.
Finally, we will write $F\sigma$ to denote the application of the valuation $\sigma \cup \Sigma$ to a formula F.

7.4 Other notations

$\models^{\tau}_{Env}$ as in D4 will represent the sensing capability of computees. We will not provide a computational counterpart for it, since in practice it is external to the model.

8 Cycle computational model

The operation of computees is dictated by their *cycle theory* [8], which is a logic program with priorities. Cycle theories define in a declarative way the possible alternative behaviors of a computee depending on the particular circumstances of the (perceived) external environment in which the computee is situated and of the internal state of the computee at the time of the operation.

In D4, the cycle theory is used to define the *operational trace* of computees, which is seen as a sequence of (applications of) *transitions*, decided via *cycle steps*, each one dictated by the cycle theory together with the definition of the *selection functions* used within the cycle theory. Each transition may call one or more *capabilities* (or no capability at all). Transitions are responsible for changes to the state of computees. Indeed, the operational trace of a computee gives rise to a sequence of states.

In this section, we provide computational counterparts for the operational trace (section 8.3) of computees, with respect to suitable computational counterparts of transitions, selection functions (both given in section 9), and of cycle steps whose computational counterpart is provided

[8] In D4 we have shown that computees with *fixed cycles* can be obtained as special cases of computees equipped with cycle theories. Therefore, we will not explicitly deal with the case of fixed cycles here.

by the techniques given in section 10.3. Thus, this section provides the "glue" for the computational models described in the remainder of the document, for the remaining components (selection functions, transitions and capabilities) of the computee model in D4.

Below, we will assume (as in D4) that the computee is equipped with a cycle theory $\mathcal{T}_{cycle}$ consisting of

- an *initial* part $\mathcal{T}_{initial}$ that determines the transitions that the computee should perform when it starts to operate (initial cycle step),
- a *basic* part $\mathcal{T}_{basic}$ that determines the possible transitions (cycle steps) following other transitions,
- an *interrupt* part $\mathcal{T}_{interrupt}$ that specifies the possible transitions (cycle steps) that can follow a POI (Passive Observation Introduction) transition, i.e. an interrupt with new information,
- a *behaviour* part $\mathcal{T}_{behaviour}$ that determines preferences on the alternative transitions given in the basic and interrupt parts.

More details on the components of $\mathcal{T}_{cycle}$ are given in section 8.1.1.

Here, differently from D4, we will allow for the behaviour part $\mathcal{T}_{behaviour}$ to express preferences also on the initial transition. This is a natural extension of the original model in D4, which exploits the possibility of performing preferential reasoning at all levels.

Note that in D4 we have tacitly assumed that the interrupt part of $\mathcal{T}_{cycle}$ always overrides the basic part. However, we have pointed out that this (tacit) restriction could be relaxed (see section 9.2.5 in D4), so that the behaviour part of $\mathcal{T}_{cycle}$ could decide whether the computee should accept to be interrupted by a POI or not. In this document we will provide a computational counterpart for the generalisation of the model in D4, where $\mathcal{T}_{behaviour}$ decides *whichever* transitions should follow any other transition, independently on whether there is a potential POI pending or not. In particular, $\mathcal{T}_{behaviour}$ might decide to "suspend", temporarily or even indefinitely, a passive observation if there are urgent matters (actions to perform or goals to achieve) to deal with. To get a conservative generalisation of the D4 model, we will assume that $\mathcal{T}_{basic}$ contains rules sanctioning that POI is a potential follow-up of any transition if there is a passive observation pending. Note that we can get the model of D4 as a special instance of the generalised model by giving higher priority to these latter rules.

Within the generalised model, POI is treated like any other transition, and this allows us to actually simplify the definition of operational trace. We will give the simplified operational trace in section 8.2.1.

Finally, note that in D4 we impose that $\mathcal{T}_{cycle}$ satisfies conditions to the effect that a unique next transition is always concluded by $\models_{pr}$ (see section 6.2) in a single cycle step. Here, we use $\models_{pr}^{cred}$ (see section 6.2) instead for $\models_{pr}$ to conclude a set of possible next transitions, and choose at random one of these as the next transition. The generation of multiple potential next transitions by $\models_{pr}^{cred}$ is the first step to extend our computee (formal and computational) model to allow concurrent execution of transitions, as discussed in D4, in section 11.4. We will give the generalised operational trace in section 8.2.2.

8.1 The cycle theory $\mathcal{T}_{cycle}$

8.1.1 Syntax

$\mathcal{T}_{cycle}$ is a logic program (given by $\mathcal{T}_{initial} \cup \mathcal{T}_{basic} \cup \mathcal{T}_{interrupt}$) with dynamic priorities (represented within $\mathcal{T}_{behaviour}$), equipped with a notion of entailment $\models_{pr}$ (which, in D4, is based upon the $LPwNF$ framework for LPP, see section 6.2).

More concretely, $\mathcal{T}_{initial}$ consists of rules of the form

$$T(S_0, X) \leftarrow C(S_0, \tau, X), time\_now(\tau)$$

sanctioning that, if the conditions C are satisfied in the initial state S_0 at the current time τ, then the initial transition should be T, applied to state S_0 and input X, if required. Note that, here and everywhere in this document, transitions are (implicitly) taken from the set $\{GI, PI, RE, SI, POI, AOI, AE, GR, PR\}$, where:

- GI stands for Goal Introduction
- PI stands for Plan Introduction
- RE stands for Reactivity
- SI stands for Sensing Introduction
- POI stands for Passive Observation Introduction
- AOI stands for Active Observation Introduction
- AE stands for Action Execution
- GR stands for Goal Revision
- PR stands for Plan revision

We will define these transitions formally (recapping from the revised D4) in section 9.

Note also that $C(S_0, \tau, X)$ may be empty, and $\mathcal{T}_{initial}$ might simply indicate a fixed initial transition T_1.

$\mathcal{T}_{basic}$ and $\mathcal{T}_{interrupt}$ consist of rules of the form

$$T'(S', X', \tau) \leftarrow T(S, X, S', \tau', \tau), EC(S', \tau, X'), time\_now(\tau)$$

which we often represent in short as

$$T'(S', X') \leftarrow T(S, X, S'), EC(S', \tau, X')$$

and refer to via the "name" $r_{T|T'}$. These rules sanction that, if at the current time τ, which is the time at which the current transition T has finished (having started at time τ'), the conditions EC evaluated in the resulting state S' are satisfied, then transition T' should follow transition T and applied with inputs the state S' and the set of items X', if required. Note that evaluating the conditions EC allows us to compute X' from S'. In the case of $\mathcal{T}_{interrupt}$, the transition T will be necessarily POI. The conditions EC are called *enabling conditions* as they determine when a cycle-step from the transition T to the transition T' can be applied. In

particular, they determine the input X', if any is required, of the next transition T'. Such input may be determined by calls to the appropriate *core selection functions*, when appropriate.

$\mathcal{T}_{behaviour}$ contains rules describing dynamic priorities amongst rules in $\mathcal{T}_{basic}$ and $\mathcal{T}_{interrupt}$ and $\mathcal{T}_{initial}$. Rules in $\mathcal{T}_{behaviour}$ are of the (abbreviated) form

$$h\_p(r_{T|T'}(S, X'), r_{T|T''}(S, X'')) \leftarrow BC(S, X', X'', \tau), time\_now(\tau)$$

which we often write simply as

$$h\_p(r_{T|T'}(S, X'), r_{T|T''}(S, X'')) \leftarrow BC(S, X', X'', \tau)$$

where $r_{T|T'}$ and $r_{T|T''}$ are (names of) rules in $\mathcal{T}_{basic} \cup \mathcal{T}_{interrupt} \cup \mathcal{T}_{initial}$. [9] We refer to such rule as $R^T_{T'|T''}$. These rules in $\mathcal{T}_{behaviour}$ sanction that, at the current time τ, after transition T, if the conditions BC hold, then we prefer the next transition to be T' over T'', namely doing T' has *higher priority* (h_p) than doing T'', after T. The conditions BC are called *behaviour conditions* and determine when the preferences apply. These conditions depend on the state of the computee after T and on the parameters chosen in the two cycle steps represented by $r_{T|T''}$ and $r_{T|T''}$. Behaviour conditions are *heuristic* conditions, defined in terms of the *heuristic selection functions*, where appropriate.

Finally, $\mathcal{T}_{cycle}$ contains an *auxiliary part* including rules of the form

$$incompatible(T(S, X), T'(S, X'))$$

for all T, T' such that $T \neq T'$ and for all T, T' such that $T = T'$ and $X \neq X'$. These rules state that all different transitions are incompatible with each other and that different calls to the same transition but with different items are incompatible with each other. The auxiliary part of $\mathcal{T}_{cycle}$ also include definitions for any predicates occuring in the enabling and behaviour conditions.

8.1.2 Example of $\mathcal{T}_{cycle}$

As a concrete example for $\mathcal{T}_{cycle}$, we give the following *normal cycle theory*, specifying a pattern of operation where the computee prefers to follow a sequence of transitions that allows it to achieve its goals in a way that matches an expected "normal" behaviour. Basically, the "normal" computee first introduces goals (if it has none to start with) via GI, then reacts to them, via RE, and then repeats the process of planning for them, via PI, executing (part of) the chosen plans, via AE, revising its state, via GR and PR, until all goals are dealt with (succesfully or revised away). At this point the computeee returns to introducing new goals via GI and repeating the process above. Whenever in this process the computee is interrupted via a passive observation, via POI, it chooses to introduce new goals via GI, to take into account any changes in the world. Whenever it has actions which are "unreliable", in the sense that their preconditions definitely need to be checked, the computee senses them (via SI) before executing the action. Whenever it has actions which are "unreliable", in the sense that their effects definitely need to be checked, the computee actively introduces actions that aim at sensing these effects, via AOI, after having executed the original actions.

[9] Note that, with an abuse of notation, and with respect to the generalisation we will discuss in section 8.2.1, T could be 0 in the case that $h\_p(r_{T|T'}(S, X'), r_{T|T''}(S, X''))$ is used to specify a priority over the *first* transition to take place in an operational trace, in other words, when the priority is over rules in $\mathcal{T}_{initial}$.

- $\mathcal{T}_{initial}$ consists of

  $r_{0|GI}(S_0) : GI(S_0) \leftarrow empty\_goals(S_0)$

  $r_{0|PI}(S_0) : PI(S_0, Gs) \leftarrow Gs = c_{GS}(S_0, \tau), Gs \neq \{\}$

  $r_{0|POI}(S_0) \leftarrow poi\_pending(\tau)$

  This last rule is only needed with the generalisation of operational trace that we will give in the following section 8.2.1 (see theorem 8.1).

- $\mathcal{T}_{basic}$ consists of the following parts.

  - The rules for deciding what might follow an AE transition are as follows:

    $r_{AE|PI}(S', Gs) : PI(S', Gs) \leftarrow AE(S, As, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$

    $r_{AE|AE}(S', As') : AE(S', As') \leftarrow AE(S, As, S'), As' = c_{AS}(S', \tau), As' \neq \{\}$

    $r_{AE|AOI}(S', Fs) : AOI(S', Fs) \leftarrow AE(S, As, S'), Fs = c_{FS}(S', \tau), Fs \neq \{\}$

    $r_{AE|PR}(S') : PR(S') \leftarrow AE(S, S')$

    $r_{AE|GI}(S') : GI(S') \leftarrow AE(S, S')$

    Namely, AE could be followed by another AE, or by a PI, or by an AOI, or by a PR, or by a GI, or by a POI. Any other possibility, e.g. for GR to follow AE, is excluded within this particular $\mathcal{T}_{basic}$ theory.

  - The rules for deciding what might follow GR are as follows

    $r_{GR|PR}(S') : PR(S') \leftarrow GR(S, S')$

    Namely, GR can only be followed by PR. Indeed, GR and PR are naturally coupled, since removing some goals in the state might lead to removing some actions.

  - The rules for deciding what might follow PR are as follows

    $r_{PR|PI}(S') : PI(S', Gs) \leftarrow PR(S, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$

    $r_{PR|GI}(S') : GI(S') \leftarrow PR(S, S'), Gs = c_{GS}(S', \tau), Gs = \{\}$

    Namely, PR can only be followed by PI or GI, depending on whether there are goals to plan for or not in the state.

  - The rules for deciding what might follow PI are as follows

    $r_{PI|AE}(S', As) : AE(S', As) \leftarrow PI(S, Gs, S'), As = c_{AS}(S', \tau), As \neq \{\}$

    $r_{PI|SI}(S', Ps) : SI(S', Ps) \leftarrow PI(S, Gs, S'), Ps = c_{PS}(S', \tau), Ps \neq \{\}$

    The second rule is here to allow the possibility of sensing the preconditions of an action before its execution.

  - The rules for deciding what might follow GI are as follows

    $r_{GI|RE}(S', Gs) : RE(S') \leftarrow GI(S, S')$

    $r_{GI|PI}(S', Gs) : PI(S', Gs) \leftarrow GI(S, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$

    Namely, GI can only be followed by RE or PI, if there are goals to plan for.

  - The rules for deciding what might follow RE are as follows

    $r_{RE|PI}(S', Gs) : PI(S', Gs) \leftarrow RE(S, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$

    $r_{RE|SI}(S', Ps) : SI(S', Ps) \leftarrow RE(S, S'), Ps = c_{PS}(S', \tau), Ps \neq \{\}$

  - The rules for deciding what might follow SI are as follows

    $r_{SI|AE}(S', As) : AE(S', As) \leftarrow SI(S, Ps, S'), As = c_{AS}(S', \tau), As \neq \{\}$

    $r_{SI|PR}(S') : PR(S') \leftarrow SI(S, Ps, S')$

– The rules for deciding what might follow AOI are as follows

$$r_{AOI|AE}(S', As) : AE(S', As) \leftarrow AOI(S, Fs, S'), As = c_{AS}(S', \tau), As \neq \{\}$$
$$r_{AOI|GR}(S') : GR(S') \leftarrow AOI(S, Fs, S')$$
$$r_{AOI|SI}(S', Ps) : SI(S', Ps) \leftarrow AOI(S, Fs, S')Ps = c_{PS}(S', \tau), Ps \neq \{\}$$

– The rules for deciding when POI should take place are as follows

$$r_{T|POI}(S') : POI(S', \tau) \leftarrow T(S, X, S', \tau', \tau), poi\_pending(\tau),$$

for all transitions T, namely POI is always an option if there is an input from the environment (observation) pending. These rules are needed only for the generalised version of operational trace given in the following section 8.2.1 (see theorem 8.1), if we want to enforce that passive observations are always taken into account when occurring (as in the original notion of operational trace in D4).

- $\mathcal{T}_{interrupt}$ consists of the following rules

$$r_{POI|GI}(S') : GI(S') \leftarrow POI(S, S')$$
$$r_{POI|RE}(S') : RE(S') \leftarrow POI(S, S')$$
$$r_{POI|GR}(S') : GR(S') \leftarrow POI(S, S')$$
$$r_{POI|POI}(S') : POI(S') \leftarrow POI(S, S')$$

This last rule is only needed with the generalisation of operational trace that we will give in the following section 8.2.1 (see theorem 8.1).

- $\mathcal{T}_{behaviour}$ consists of the following rules (besides its auxiliary part, including the definitions for *incompatible* given above as well as any other definitions for predicates used in $\mathcal{T}_{behaviour}$, such as $empty\_goals, unreliable\_pre$ etc):

  – GI should be given higher priority if there are no goals in *Goals* and actions in *Plan* in the state: [10]

  $$R^T_{GI|T'} : h\_p(r_{T|GI}(S), r_{T|T'}(S, X)) \leftarrow empty\_goals(S), empty\_plan(S)$$

  for all transitions T, T', $T' \neq GI$, and with T possibly 0 (indicating that if there are no goals and plans in the initial state of a computee, then GI should be its first transition).

  – GI is also given higher priority after a POI:

  $$R^{POI}_{GI|T} : h\_p(r_{POI|GI}(S'), r_{POI|T}(S, S'))$$

  for all transitions $T \neq GI$.

  – After GI, the transition RE should be given higher priority:

  $$R^{GI}_{RE|T} : h\_p(r_{GI|RE}(S), r_{GI|T}(S, X))$$

  for all transitions $T \neq RE$.

  – After RE, the transition PI should be given higher priority:

  $$R^{RE}_{PI|T} : h\_p(r_{RE|PI}(S, Gs), r_{RE|T}(S, X))$$

  for all transitions $T \neq PI$.

[10] Instead of the conditions $empty\_goals(S), empty\_plan(S)$ in this rule, we could have these conditions in each rule in $\mathcal{T}_{basic}$ which indicates as viable any transition that could be a competitor of GI after any given transition.

– After PI, the transition AE should be given higher priority, unless there are actions in the actions selected for execution whose preconditions are "unreliable" and need checking, in which case SI will be given higher priority:

$$R^{PI}_{AE|T} : h\_p(r_{PI|AE}(S, As), r_{PI|T}(S, X)) \leftarrow not\, unreliable\_pre(As)$$

for all transitions $T \neq AE$;

$$R^{PI}_{SI|T} : h\_p(r_{PI|SI}(S, Ps), r_{PI|T}(S, As)) \leftarrow unreliable\_pre(As)$$

for all transitions $T \neq SI$;

Here we assume that the auxiliary part of $\mathcal{T}_{cycle}$ specifies whether a given set of actions contains any "unreliable" action, in the sense described above.

– After SI, the transition AE should be given higher priority

$$R^{SI}_{AE|T} : h\_p(r_{SI|AE}(S, As), r_{SI|T}(S, X))$$

for all transitions $T \neq AE$.

– After AE, the transition AE should be given higher priority until there are no more actions to execute in $Plan$, in which case either AOI or GR should be given higher priority, depending on whether there are actions which are "unreliable", in the sense that their effects need checking, or not:

$$R^{AE}_{AE|T} : h\_p(r_{AE|AE}(S, As), r_{AE|T}(S, X))$$

for all transitions $T \neq AE$. Note that, by definition of $\mathcal{T}_{basic}$, the transition AE is applicable only if there are still actions to be executed in the state.

$$R^{AE}_{AOI|T} : h\_p(r_{AE|AOI}(S, Fs), r_{AE|T}(S, X)) \leftarrow BC^{AE}_{AOI|T}(S, Fs)$$

for all transitions $T \neq AOI$, where the behaviour condition $BC^{AE}_{AOI|T}(S, Fs)$ is defined (in the auxiliary part) by:

$$BC^{AE}_{AOI|T}(S, FS) \leftarrow empty\_executable\_plan(S), unreliable\_post(S)$$

Similarly, we have:

$$R^{AE}_{GR|T} : h\_p(r_{AE|GR}(S), r_{AE|T}(S, X)) \leftarrow BC^{AE}_{GR|T}(S)$$

for all transitions $T \neq GR$ where:

$$BC^{AE}_{GR|T}(S) \leftarrow empty\_executable\_plan(S), not\, unreliable\_post(S)$$

Here, we assume that the auxiliary part of $\mathcal{T}_{cycle}$ specifies whether a given set of actions contains any "unreliable" action, in the sense expressed by $unreliable\_post$, and defines the predicate $empty\_executable\_plan$. Intuitively, $empty\_executable\_plan(S)$ succeeds if all the actions which can be selected for execution have already been executed.

– After GR, the transition PR should have higher priority:

$$R^{GR}_{PR|T} : h\_p(r_{GR|PR}(S, \_), r_{GR|T}(S))$$

for all transitions $T \neq PR$.

– After PR, the transition PI should have higher priority:

$$R^{PR}_{PI|T} : h\_p(r_{PR|PI}(S, Gs), r_{PR|T}(S))$$

for all transitions $T \neq PI$. Note that, by definition of $\mathcal{T}_{basic}$, the transition PI is applicable only if there are still goals to plan for in the state. If there are no actions and goals left in the state, then rule $R^{T}_{GI|T'}$ would apply.

– After any transition, POI is preferred over all other transitions:

$$R^T_{PI|T'} : h\_p(r_{T|POI}(S), r_{T|T'}(S, X))$$

for all transitions T, T', $T' \neq POI$, with T possibly 0 (indicating that, if applicable by $\mathcal{T}_{initial}$, POI should be the first transition).

This priority rule is only needed with the generalisation of operational trace that we will give in the following section 8.2.1 (see theorem 8.1).

– In the initial state, PI should be given higher priority:

$$R^0_{PI|T} : h\_p(r_{0|PI}(S, Gs), r_{0|T}(S, X))$$

for all transitions $T \neq PI$. Note that PI, by definition of $\mathcal{T}_{initial}$, the transition PI is applicable initially only if there are goals to plan for in the initial state.

Note that this is just an example of cycle theory, and many others are possible. To obtain different patterns of behaviour we can restrict the rules in $\mathcal{T}_{basic}$ either by adding extra conditions in the body of its rules or by dropping some rules completely. For example, we could add the conditions $empty\_goals(S'), empty\_plan(S')$ in all the rules in $\mathcal{T}_{basic}$ introducing GI, namely all rules $r_{*|GI}(S', Gs)$, thus allowing GI only if the state is empty. This would in turn allow us to drop the conditions $empty\_goals(S'), empty\_plan(S')$ in the rule $R^T_{GI|T'}$ in $\mathcal{T}_{behaviour}$. As another example, we may want to drop the last rule in the definition of what might follow AE in $\mathcal{T}_{basic}$ as we may never want to do GI unless we have new observational input.

8.2 Operational trace

In this section, we define the two generalisations of operational trace described earlier on, on which the computational counterpart will be based.

In D4, the operational trace of a computee is seen as a (typically infinite) sequence

$$\sigma_1, \ldots, \sigma_j, \ldots$$

of sequences σ_i of (applications of) transitions. Each (application of a) transition has the form

$$T(S, X, S', \tau)$$

where

- T is a transition name, chosen from within $\{GI, PI, RE, SI, POI, AOI, AE, GR, PR\}$,
- S, S' are states of the computee (namely $S = \langle KB, Goals, Plan\rangle$, for some KB, $Goals$ and $Plan$, and $S' = \langle KB', Goals', Plan'\rangle$, obtained from S via the application of transition T), and
- τ is the time given by the clock of the system when T is applied (namely the current time).

The first sequence σ_1 starts with the transition chosen via $\models_{pr}$ on $\mathcal{T}_{initial}$, applied on some given initial state S_0 of the computee. We will refer to this choice as the **initial step**. All other sequences starts with a POI transition applied on the state resulting from applying the last transition in the previous sequence. In each sequence, $\models_{pr}$ is used to decide which transition should follow any given one, by means of a **cycle step**.

As mentioned earlier, this definition of operational trace assumes that no reasoning is applied to decide the initial transition and that the POI transition is always executed as soon as there is a (passive) observation that requires attention. Thus, this definition (tacitly) assumes that the interrupt part $\mathcal{T}_{interrupt}$ of $\mathcal{T}_{cycle}$ always overrides the basic part $\mathcal{T}_{basic}$. These assumptions can be dropped to get a less restricted computee behaviour, by allowing the behaviour part $\mathcal{T}_{behaviour}$ of $\mathcal{T}_{cycle}$ to decide the initial transition and whether the computee should accept to be interrupted by a POI or not.

8.2.1 Generalisation 1: Initial transition and POI controlled by $\mathcal{T}_{behaviour}$

The *(generalised) operational trace* given by $\mathcal{T}_{cycle}$ is a (typically infinite) sequence of transitions

$$T_1(S_0, X_1, S_1, \tau_1), \ldots, T_i(S_{i-1}, X_i, S_i, \tau_i), T_{i+1}(S_i, X_{i+1}, S_{i+1}, \tau_{i+1}), \ldots$$

(where each of the X_i may be empty), such that

- S_0 is the given initial state;
- (**Generalised Initial Step**) $\mathcal{T}_{initial} \wedge \mathcal{T}_{behaviour} \models_{pr} T_1(S_0, X_1)$;
- for each $i \geq 1$, τ_i is given by the clock of the system at the time that T_i is applied, namely $time\_now(\tau_i)$ holds when T_i is applied, with the property that $\tau_i < \tau_{i+i}$, namely time increases;
- (**Generalised Cycle Step**) for each $i \geq 1$

  $$\mathcal{T}_{basic} \wedge \mathcal{T}_{interrupt} \wedge \mathcal{T}_{behaviour} \wedge T_i(S_{i-1}, X_i, S_i, \tau_i, \tau_{i+1}) \models_{pr} T_{i+1}(S_i, X_{i+1}, \tau_{i+1})$$

  namely each (non-final) transition in a sequence is followed by the most preferred transition, as specified by $\mathcal{T}_{cycle}$.

Thus, within the generalised model, POI is treated like any other transition, and this allows us to simplify the definition of operational trace to just a sequence of transitions (rather than a sequence of sequences of transitions). [11]

Note that, with this generalisation, the delay in executing the transition POI might bring about a delay in recording observations into KB_0, so that the time associated with the observations in KB_0 might actually be later than the actual time of the observation, and that observations made at different times are recorded as observations made at the same time. This feature, however, is already present in the original model in D4, as POI cannot interrupt transitions, but only interrupt their flow in the operational trace.

Note that, for the generalised model to actually generalise the model of D4, we need to make sure that POI is taken into account as a candidate next transition whenever a passive observation is lined up. To do so, we need to assume that $\mathcal{T}_{initial}$ and $\mathcal{T}_{basic}$ and $\mathcal{T}_{interrupt}$ contain rules sanctioning that POI is a potential follow-up of any transition if there is a passive observation pending. Then, we get the original model of operational trace and computee behaviour of D4 as a special instance of the generalised model by giving higher priority to the aforementioned added rules. This is formally stated in the following theorem, whose proof is straightforward.

[11] The generalisation to reason about the initial step is non-controversial, and will be taken for granted in the sequel.

Theorem 8.1. *Let*

$$\mathcal{T}_{cycle} = \mathcal{T}_{initial} \cup \mathcal{T}_{basic} \cup \mathcal{T}_{interrupt} \cup \mathcal{T}_{behaviour}$$

be a cycle theory and

$$\mathcal{T}^g_{cycle} = \mathcal{T}^g_{initial} \cup \mathcal{T}^g_{basic} \cup \mathcal{T}^g_{interrupt} \cup \mathcal{T}^g_{behaviour}$$

be the extension of $\mathcal{T}_{cycle}$ *where*

- $\mathcal{T}^g_{initial}$ *is given by* $\mathcal{T}_{initial}$ *extended with the rule*
  $$r_{0|POI}(S_0) \leftarrow poi\_pending(\tau)$$
- $\mathcal{T}^g_{basic}$ *is given by* $\mathcal{T}_{basic}$ *extended with the rules* $r_{T|POI}$*:*
  $$POI(S') \leftarrow T(S, X, S', \tau', \tau), poi\_pending(\tau)$$
  for all T*,*
- $\mathcal{T}^g_{interrupt}$ *is given by* $\mathcal{T}_{interrupt}$ *extended with*
  $$r_{POI|POI}(S') : POI(S') \leftarrow POI(S, S')$$
  and
- $\mathcal{T}^g_{behaviour}$ *is given by* $\mathcal{T}_{behaviour}$ *extended with the priority rules:*
  $$h\_p(r_{T|POI}(S), r_{T|T'}(S, X, \tau))$$
  for all T, T', $T' \neq POI$*.*

Then, there is a one-to-one correspondence between any operational trace as in D4, wrt $\mathcal{T}_{cycle}$*, and the generalised operational trace, wrt* $\mathcal{T}^g_{cycle}$*.*

In the sequel we will refer to the generalised operational trace above simply as operational trace.

8.2.2 Generalisation 2: Cycle step with set of alternatives

In D4 we impose that $\mathcal{T}_{cycle}$ satisfies conditions to the effect that a unique next transition is always concluded by $\models_{pr}$ in a single cycle step. One possible pool of such conditions is the following, given that a *family* of rules in $\mathcal{T}_{behaviour}$ is the set of all rules to decide the follow-up of the same current transition:

1. the conditions of all rules in $\mathcal{T}_{initial}$ are *exclusive*, namely no two such conditions can hold at the same time;
2. the enabling conditions for rules in $\mathcal{T}_{basic}$ and $\mathcal{T}_{interrupt}$ are *exclusive*, as above;
3. the behaviour conditions of rules in $\mathcal{T}_{behaviour}$ which belong to the same family are *exclusive*, as above;
4. the conditions of all rules in $\mathcal{T}_{initial}$ are *exhaustive*, namely one such condition is always satisfied;
5. the enabling conditions for rules in $\mathcal{T}_{basic}$ and $\mathcal{T}_{interrupt}$ are *exhaustive*, as above;

6. the behaviour conditions of rules in $\mathcal{T}_{behaviour}$ which belong to the same family are *exhaustive*, as above.

Here, the first three conditions guarantee that there is at most one transition possible at each time, whereas the last three conditions guarantee that there is at least one transition possible at each time.

These conditions (and conditions to the same effect) are cumbersome to test and complicate writing $\mathcal{T}_{cycle}$. More importantly, enforcing that a unique next transition is always concluded by $\models_{pr}$ prevents from the onset the concurrent execution of transitions, which can instead be beneficial in many cases. (For example, it might be useful to be able to plan for one goal, via PI, while executing actions in a plan for another goal, via AE.) Below, we perform the first step towards a model of computees where transitions can be executed concurrently, by allowing a pool of possible next transitions to be returned within the same cycle step, and the next transition to be chosen randomly. (In order to get a fully-fledged concurrent model for computees we would need to execute all or some of the transitions in the pool, as well as say how the state is updated through the concurrent execution of transitions.)

The (further) generalised model is obtained as follows.

- We relax the conditions that all transitions are incompatible, so that $\mathcal{T}_{behaviour}$ contains the rules

  $$incompatible(T(S,X), T(S,X'))$$

  for all $X \neq X'$ and rules

  $$incompatible(T(S,X), T'(S,X'))$$

  for *some* $T, T', T \neq T'$, stating that some transitions are incompatible with some others (but allowing for some transitions to be compatible).

- We define a *(further generalised) operational trace*, by modifying the initial step and the cycle step as follows

  - (**Further Generalised Initial Step**)
    if $Set = \{T(S_0, X) \mid \mathcal{T}_{initial} \wedge \mathcal{T}_{behaviour} \models_{pr} T(S_0, X)\}$
    then $T_1(S_0, X_1) \in Set$.
  - (**Further Generalised Cycle Step**) for each $i \geq 1$,
    if $Set = \{T(S_i, X, \tau_{i+1}) \mid$
    $\mathcal{T}_{basic} \wedge \mathcal{T}_{interrupt} \wedge \mathcal{T}_{behaviour} \wedge T_i(S_{i-1}, X_i, S_i, \tau_i, \tau_{i+1}) \models_{pr} T(S_i, X, \tau_{i+1})\}$
    then $T_{i+1}(S_i, X_{i+1}, \tau_{i+1}) \in Set$.
    Namely, in the initial step, we first take the set of all possible first transitions, and then choose at random one of these as the initial transition. Also, in the cycle step, we first take the set of all possible next transitions, and then choose at random one of these as the next transition. In other words, each (non-final) transition in a sequence is followed by one of the transitions in the set of the most preferred transitions, as specified by $\mathcal{T}_{cycle}$. Note that this set may be empty. In this case the computee would stay idle.

Below we give a computationally more viable reformulation of the above generalised operational trace, by using the credulous version $\models_{pr}^{cred}$ of $\models_{pr}$ to conclude a randomly chosen next transition directly (without having to construct a full set first). This reformulation is not equivalent to the one above, except in very special cases (see theorem 8.3 below). In this reformulation the initial and cycle step become, respectively:

- (**Reformulation of the Further Generalised Initial Step**)
  $\mathcal{T}_{initial} \wedge \mathcal{T}_{behaviour} \models_{pr}^{cred} T_1(S_0, X_1)$
- (**Reformulation of the Further Generalised Cycle Step**) for each $i \geq 1$,
  $\mathcal{T}_{basic} \wedge \mathcal{T}_{interrupt} \wedge \mathcal{T}_{behaviour} \wedge T_i(S_{i-1}, X_i, S_i, \tau_i, \tau_{i+1}) \models_{pr}^{cred} T_{i+1}(S_i, X_{i+1}, \tau_{i+1})$.

Obviously, since every sceptical conclusion can also be obtained credulously, the following theorem holds.

Theorem 8.2. *Every operational trace wrt the original definition in D4 is a further generalised operational trace reformulated via* $\models_{pr}^{cred}$.

However, the converse of this theorem does not hold in general. Thus, this reformulation of the second generalisation is not a conservative extension of the original operational trace of D4. Trivially, though, if $\mathcal{T}_{cycle}$ satisfies conditions to the effect that a unique next transition is always concluded by $\models_{pr}$ in the initial step and each single cycle step, then the further generalised operational trace (and its reformulation in terms of $\models_{pr}^{cred}$) is equivalent to the earlier operational trace, as stated by the following theorem, whose proof is trivial:

Theorem 8.3. *If* $\mathcal{T}_{cycle}$ *is such that there exists a single transition* T_1 *such that*
$\mathcal{T}_{initial} \wedge \mathcal{T}_{behaviour} \models_{pr}^{cred} T_1(S_0, X_1)$
if and only if
$\mathcal{T}_{initial} \wedge \mathcal{T}_{behaviour} \models_{pr} T_1(S_0, X_1)$
and for each transition T *there exists a single transition* T' *such that*
$\mathcal{T}_{basic} \wedge \mathcal{T}_{interrupt} \wedge \mathcal{T}_{behaviour} \wedge T(S, X, S', \tau, \tau') \models_{pr}^{cred} T'(S', X', \tau')$
if and only if
$\mathcal{T}_{basic} \wedge \mathcal{T}_{interrupt} \wedge \mathcal{T}_{behaviour} \wedge T(S, X, S', \tau, \tau') \models_{pr} T'(S', X', \tau')$
then there is a one-to-one correspondence between any operational trace and the (reformulation via $\models_{pr}^{cred}$ *of the) further generalised operational trace.*

The computational techniques presented in section 10.3 can be used directly to provide a computational counterpart of the cycle step in the reformulation in terms of $\models_{pr}^{cred}$ of the further generalised operational trace, which we will refer to simply as operational trace in the sequel.

8.3 Computational counterpart of the operational trace

In this section we provide the computational counterpart of the (further generalised) operational trace. This is obtained simply by replacing $\models_{pr}^{cred}$ in the initial step and in the cycle step by a suitable computational counterpart $\vdash_{pr}^{cred}$ (see section 10.3 for its definition), and transitions and selection functions by suitable computational counterparts (see section 9 for their definition). Namely, given $\mathcal{T}_{cycle}$ and suitable computational counterparts of the (core and heuristic) selection functions, to evaluate conditions in $\mathcal{T}_{cycle}$, the *computational operational trace* is a (typically infinite) sequence of transitions

$$T_1^c(S_0, X_1, S_1, \tau_1), \ldots, T_i^c(S_{i-1}, X_i, S_i, \tau_i), T_{i+1}^c(S_i, X_{i+1}, S_{i+1}, \tau_{i+1}), \ldots$$

(where each of the Xs may be empty), such that T_i^c is the computational counterpart of T_i and

- S_0 is the given initial state;
- (**Computational Initial Step**) $\mathcal{T}_{initial} \cup \mathcal{T}_{behaviour} \vdash_{pr}^{cred} T_1(S_0, X_1)$;
- for each $i \geq 1$, τ_i is given by the clock of the system at the time that T_i is applied, with the property that $\tau_i < \tau_{i+i}$, namely time increases;
- (**Computational Cycle Step**) for each $i \geq 1$

$$\mathcal{T}_{basic} \wedge \mathcal{T}_{interrupt} \wedge \mathcal{T}_{behaviour} \wedge T_i(S_{i-1}, X_i, S_i, \tau_i, \tau_{i+1}) \vdash_{pr}^{cred} T_{i+1}(S_i, X_{i+1}, \tau_{i+1})$$

In the sequel we will ignore the initial step, as it can be dealt with similarly to the cycle step.

The following theorem, stating the correctness of the computational operational trace wrt the (reformulation of the generalised) operational trace *given* correct computational models for the cycle step, the selection functions and the transitions, trivially holds.

Theorem 8.4. *(Conditional correctness of the computational operational trace)*
Let

- $\vdash_{pr}^{cred}$ *be a correct computational model for* $\models_{pr}^{cred}$,
- $\{\vdash_{GI}, \vdash_{PI}, \vdash_{RE}, \vdash_{SI}, \vdash_{POI}, \vdash_{AOI}, \vdash_{AE}, \vdash_{GR}, \vdash_{PR}\}$ *be correct computational counterparts of the transitions* $\{GI, PI, RE, SI, POI, AOI, AE, GR, PR\}$, *and*
- $c_{AS}^c, c_{GS}^c, c_{FS}^c, c_{PS}^c, h_{AS}^c, h_{GS}^c, h_{FS}^c, h_{PS}^c$ *be correct computational counterparts of the (core and heuristic) selection functions (for action, goal, fluent and precondition selection).*

Then, any computational operational trace, wrt some given cycle theory $\mathcal{T}_{cycle}$ *and initial state* S_0*, is an operational trace, wrt the same cycle theory and initial state.*

We will see suitable computational counterparts for $\vdash_{pr}^{cred}$ in definition 10.14 and for the transitions and selection functions in section 9.

9 Computational model for selection functions and transitions

In the previous section 8, we have provided a notion of computational operational trace of a computee, responsible for its behaviour, parametric on computational models for selection functions and transitions. We have indicated the computational models for the transitions

$$\{GI, PI, RE, SI, POI, AOI, AE, GR, PR\}$$

as

$$\{\vdash_{GI}, \vdash_{PI}, \vdash_{RE}, \vdash_{SI}, \vdash_{POI}, \vdash_{AOI}, \vdash_{AE}, \vdash_{GR}, \vdash_{PR}\}$$

and the computational models for the selection functions

c_{AS} (core action selection)
c_{GS} (core goal selection)
c_{FS} (core fluent selection)
c_{PS} (core precondition selection)
h_{AS} (heuristic action selection)
h_{GS} (heuristic goal selection)
h_{FS} (heuristic fluent selection)
h_{PS} (heuristic precondition selection)

as

$$c^c_{AS}, c^c_{GS}, c^c_{FS}, c^c_{PS}, h^c_{AS}, h^c_{GS}, h^c_{FS}, h^c_{PS}.$$

Here, we provide concretely the computational models for selection functions and transitions, and give conditional results on their correctness, assuming correct computational counterparts for

1. the capabilities that are "called" from within the selection functions and the transitions (goal decision, planning, reactivity, temporal reasoning) and

2. the constraint satisfaction $\models_{\Re}$, that deals with the temporal constraints.

The computational counterparts for the capabilities (goal decision $\vdash^{\tau}_{GD}$, planning $\vdash^{\tau}_{plan}$, reactivity $\vdash^{\tau}_{react}$, temporal reasoning $\vdash_{TR}$) are given later on in the document. The computational counterpart of $\models_{\Re}$, referred to as $\vdash_{\Re}$, has been described in section 6.3. Here and in the rest of the document we will assume to have a correct and complete $\vdash_{\Re}$. In particular, we will assume that, for any (set of) temporal constraints TC, defined as in D4,

$\vdash_{\Re} TC$ if and only if
there exists a (total) substitution σ for the variables in TC such that $\sigma \models_{\Re} TC$

Below, we will ignore the heuristic selection functions and concentrate solely on the core selection functions. Indeed, the definition of heuristic selection functions is linked to concrete domains of applications and categories of behaviour, and does not lend itself to a general formulation.

9.1 Computational counterparts for the (core) selection functions

At a high-level of description, the core selection functions can all be seen as returning the set of all items from a given initial set that satisfy a certain number of conditions. For example, given a state $\langle KB, Goals, Plan\rangle$, the action selection function returns the set of all actions in $Plan$ that satisfy some conditions; the goal selection function returns the set of all goals in $Goals$ that satisfy some conditions; the fluent selection function returns the set of all fluents which are effects of actions already executed (as recorded in KB_0) that satisfy some conditions; the precondition selection function returns the set of all pairs, each consisting of a timed fluent literal (occurring somewhere in the precondition part of some action in $Plan$) and a goal in $Goals$ (the parent of the action), that satisfy some conditions.

Below, for each selection function, we will provide a computational counterpart for each of the given conditions. These computational counterparts of checking the required conditions are given in terms of appropriate calls to the computational counterparts of the capabilities and of the constraint solver that are invoked from within the formal specification of the selection functions.

The fully-fledged computational model for each of the selection functions can then be obtained by identifying a procedure for checking each of the conditions on the set of all candidate items, until the required (maximal) set of items passing all checks is returned. Below, we will not commit to any such procedure. However, note that a concrete procedure could perform the checks *sequentially*, by incrementally shrinking the set of candidate items by removing one by one the items that do not pass all the checks, or *concurrently*, by taking the intersection of all sets of candidate items separately passing the individual ckecks. Trivially, whichever order of execution of the checks we adopt, the resulting computational model will be correct with respect to the specification, provided that the computational counterparts of checking the conditions are correct.

Note that all items in any given initial set will have to be considered in performing these checks, and some of them will have to be considered as many times as the number of checks to be performed (in the case in which all items in the initial set pass the checks, then each item in the initial set will have been considered as many times as the checks). This may be rather costly. In the implementation (see deliverable D9 [5]), we have taken the view that the first item that is found to satisfy all required conditions is returned as the *sole* selected item, namely the sets returned are singleton sets. This provides a more (computationally) viable counterpart of the selection functions, even though an approximation.

Finally, note that below we summarise the specification of all core selection functions as given in the revised D4. With respect to the original D4:

- we have simplified the action and goal selection functions so that no call to the planning capability is made within them: calling such capability to simply check the existence of plans (as in the original specification) amounted to performing planning without actually keeping any record of it, and so it was wasteful and computationally not viable;
- temporal constraints of individual actions/goals are checked for satisfiability with respect to the overall set of temporal constraints in the state, rather than in isolation (as in the original specification): testing temporal constraints in isolation turns out to be insufficient in practice.

9.1.1 Action selection function

Specification of c_{AS}

Informally, the set of conditions for the core action selection function is as follows. Given a state $S = \langle KB, Goals, Plan \rangle$ and a time-point τ, the set of all actions selected by c_{AS} is the set of all actions A in $Plan$ such that at time τ:

1. A is executable at τ, e.g. it is not timed out,
2. no ancestor or sibling of A in $Goals$ and $Plan$ is timed out at τ,
3. no ancestor of A in $Goals$ is already satisfied at τ, given S,

4. no precondition of A is known to be false at τ, given S,

5. A has not already been executed.

Formally, given a state $S = \langle KB, Goals, Plan\rangle$, with overall set of temporal constraints TCS, and a time-point τ, the set of all actions selected by c_{AS} is the set of all actions

$$A = \langle a[t], G, C, Tc\rangle \in Plan$$

such that:

1. there exists a total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t = \tau \wedge TCS$,

2. there exists no action $\langle a'[t'], G^*, C', Tc'\rangle \in Plan$ and there exists no goal $\langle l[t'], G^*, Tc'\rangle \in Goals$ such that

   - $G^* = G$ or $G^* \in ancestors(G, Goals)$, and
   - there exists no total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t' \geq \tau \wedge TCS$,

3. there exists no $\langle l[t'], G^*, Tc'\rangle \in Goals$ such that

   - $G^* = G$ or $G^* \in ancestors(G, Goals)$, and
   - there exists a total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t' \leq \tau \wedge TCS$ and $KB \models_{TR} l[t']\sigma$,

4. let $C = l_1[t] \wedge \ldots \wedge l_n[t]$; if $n > 0$, then it is not the case that for some $i = 1, \ldots, n$ there exists a total valuation σ for the variables in TCS such that $\sigma \models_{\Re} TCS \wedge t = \tau$ and $KB \models_{TR} \overline{l_i[t]}\sigma$,

5. $executed(a[t], t') \notin KB_0$.

c^c_{AS}: computational counterpart of c_{AS}

The computational counterpart for this selection function can be obtained by having computational counterparts for the above conditions appropriately combined (either sequentially or concurrently checked, as discussed earlier).

1. $\vdash_{\Re} t = \tau \wedge TCS$

2. for each $\langle a'[t'], G^*, C', Tc'\rangle \in Plan$ and $\langle l[t'], G^*, Tc'\rangle \in Goals$ such that $G^* = G$ or $G^* \in ancestors(G, Goals)$:

$$\vdash_{\Re} t' \geq \tau \wedge TCS,$$

3. for each $\langle l[t'], G^*, Tc'\rangle \in Goals$ such that $G^* = G$ or $G^* \in ancestors(G, Goals)$:

$$\vdash_{TR} \text{ (finitely) fails to prove that } l[t'] \wedge t' \leq \tau \wedge TCS$$

4. let $C = l_1[t] \wedge \ldots \wedge l_n[t]$; if $n > 0$, then for every $i = 1, \ldots, n$

$$\vdash_{TR} \text{ (finitely) fails to prove that } \overline{l_i[t]} \wedge TCS \wedge t = \tau,$$

5. $executed(a[t], t') \notin KB_0$.

Trivially, if $\vdash_{TR}$ and $\vdash_{\Re}$ are correct computational counterparts of $\models_{TR}$ and $\models_{\Re}$, then the given computational counterparts of the checks 1-5 in the specification of c_{AS} are correct wrt the specification itself, and thus an overall correct computation counterpart of c_{AS} can be obtained.

9.1.2 Goal selection function

Specification of c_{GS}

Informally, the set of conditions for the core goal selection function is as follows. Given a state $S = \langle KB, Goals, Plan \rangle$ and a time-point τ, the set of all goals selected by c_{GS} is the set of all goals G in $Goals$ such that at time τ:

1. G is not timed out at τ,
2. no ancestor or sibling of G in $Goals$ and $Plan$ is timed out at τ,
3. no ancestor of G in $Goals$ is satisfied in S at τ,
4. G is not satisfied in S at τ.

Formally, given a state $S = \langle KB, Goals, Plan \rangle$ with overall set of temporal constraints TCS, and a time-point τ, the set of all goals selected by c_{GS} is the set of all goals

$$G = \langle l[t], G', Tc \rangle \in Goals$$

such that:

1. there exists a total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t > \tau \wedge TCS$,
2. there exists no $\langle a[t'], G', C, Tc' \rangle \in Plan$, and there exists no $\langle l'[t'], G^*, Tc' \rangle \in Goals$ such that
   - $G^* = G'$ or $G^* \in ancestors(G', Goals)$, and
   - there exists no total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t' \geq \tau \wedge TCS$,
3. there exists no $G^* = \langle l'[t'], \_, Tc' \rangle \in Goals$ such that
   - $G^* \in ancestors(G, Goals)$, and
   - there exists a total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t' \leq \tau \wedge TCS$ and $KB \models_{TR} l'[t']\sigma$,
4. there exists no total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t' \leq \tau \wedge TCS$ and $KB \models_{TR} l[t]\sigma$.

c^c_{GS}: computational counterpart of c_{GS}

The computational counterpart for this selection function can be obtained by having computational counterparts for the above conditions appropriately combined (either sequentially or concurrently checked, as discussed earlier).

1. $\vdash_{\Re} t > \tau \wedge TCS$
2. for each $\langle a[t'], G', C, Tc' \rangle \in Plan$ and $\langle l'[t'], G^*, Tc' \rangle \in Goals$ such that $G^* = G'$ or $G^* \in ancestors(G', Goals)$:

$$\vdash_{\Re} t' \geq \tau \wedge TCS$$

3. for each $G^* = \langle l'[t'], \_, Tc' \rangle \in Goals$ such that $G^* \in ancestors(G, Goals)$:

$$\vdash_{TR} \text{ (finitely) fails to prove that } l'[t'] \wedge t' \leq \tau \wedge TCS$$

4. $\vdash_{TR}$ (finitely) fails to prove that $l[t] \wedge t' \leq \tau \wedge TCS$.

Trivially, if $\vdash_{TR}$ and $\vdash_{\Re}$ are correct computational counterparts of $\models_{TR}$ and $\models_{\Re}$, then the given computational counterparts of the checks 1-4 in the specification of c_{GS} are correct wrt the specification itself, and thus an overall correct computation counterpart of c_{GS} can be obtained.

9.1.3 Fluent selection function

Specification of c_{FS}

Informally, the set of conditions for the core fluent selection function is as follows. Given a state $S = \langle KB, Goals, Plan \rangle$ and a time-point τ, the set of all (timed) fluents selected by c_{FS} is the set of all (timed) fluents $f[t]$ such that:

1. f or $\neg f$ is one of the effects of some action A in $Plan$,
2. this action A has recently been executed.

Note that such $f[t]$ (or $\neg f[t]$) may not occur in $Goals$ but could be some other (observable) effect of the executed action which is not necessarily the same as the goal that the action contributes to achieving.

Formally, given a state $S = \langle KB, Goals, Plan \rangle$, with overall set of temporal constraints TCS, and a time-point τ, the set of all (timed) fluents selected by c_{FS} is the set of all (timed) fluents $f[t]$ such that:

1. there exists an action $\langle a[t], G, C, Tc \rangle \in Plan$ and a rule in KB_{plan} with head $initiates(a, \_, f)$ or $terminates(a, \_, f)$, and
2. $KB_0 \models_{LP} executed(a[\_], \tau')$ and $\tau - \epsilon < \tau' < \tau$, where ϵ is a sufficiently small number.

In this document, we choose $\models_{LP}$ to be truth wrt the completion semantics (see section 6.1). Moreover, we integrate constraint satisfiability $\models_{\Re}$ within this semantics, indicated as $Comp(\_) \models^3_{\Re}$ (see section 6.3). Thus, the second condition above can be rewritten as

2. $Comp(KB_0) \models^3_{\Re} executed(a[\_], \tau') \wedge \tau - \epsilon < \tau' < \tau$.

c^c_{FS}: computational counterpart of c_{FS}

The computational counterpart for this selection function can be obtained by having computational counterparts for the above conditions 1-2, appropriately combined.

1. there exists an action $\langle a[t], G, C, Tc\rangle \in Plan$ and a rule in KB_{plan} with head $initiates(a, \_, f)$ or $terminates(a, \_, f)$, and
2. $KB_0 \vdash executed(a[\_], \tau') \wedge \tau - \epsilon < \tau' \wedge \tau' < \tau$, where ϵ is a sufficiently small number and $\vdash$ is some appropriate computational mechanism for $Comp(\_) \models^3_{\Re}$.

We will see (in section 10.1) that C-IFF is a correct computational counterpart of $Comp(\_) \models^3_{\Re}$ and can be used in place of $\vdash$ in condition 2 above. Thus, condition 2 above can be rewritten as

2. $\langle KB_0, \{\}, \{\}\rangle$, $executed(a[\_], \tau') \wedge \tau - \epsilon < \tau' < \tau \vdash_{\text{ciff}} (\{\}, \_)$, namely there exists a successful derivation for $executed(a[\_], \tau') \wedge \tau - \epsilon < \tau' < \tau$, given the abductive logic program $\langle KB_0, \{\}, \{\}\rangle$.

The given computational counterparts of the check in the specification of c_{FS} are correct wrt the specification itself by using (the correct) C-IFF. Thus, an overall correct computation counterpart of c_{FS} can be obtained from the computational counterparts of checks 1-2 above.

9.1.4 Precondition selection function

Specification of c_{PS}

Informally, the set of conditions for the core precondition selection function is as follows. Given a state $S = \langle KB, Goals, Plan\rangle$ and a time-point τ, the set of preconditions (of actions in $Plan$) selected by c_{PS} is the set of all pairs (C, G) of (timed) preconditions C and goals G such that:

1. there exists an action A in $Plan$ such that C is a precondition of A and G is the parent of A,
2. C is not known to be true in S at τ,
3. $A \in c_{AS}(S, \tau)$.

The reasons why this selection function returns pairs, rather then simply preconditions, are that the transition SI, which makes use of the outputs of this selection function, needs to know which parent to associate to the sensing actions it will introduce, for each precondition returned. This parent is the element associated to the precondition in the pair. This will become clearer later on, in section 9.2.4.

Formally, given a state $S = \langle KB, Goals, Plan\rangle$, where TCS is the set of all temporal constraints in S, and a time-point τ, the set of all preconditions of actions selected by c_{PS} is the set of all pairs (C, G) of (timed) preconditions C and goals G such that:

1. there exists $A = \langle a[t], G, Cs, Tc\rangle \in Plan$ such that C is a conjunct in Cs,
2. there exists no total valuation σ for the variables in TCS such that $\sigma \models_{\Re} t = \tau \wedge TCS$ and $KB \models_{TR} C\sigma$,
3. $A \in c_{AS}(S, \tau)$.

c^c_{PS}: computational counterpart of c_{PS}

The computational counterpart for this selection function can be obtained by having computational counterparts for the above conditions 1-3 appropriately combined (either sequentially or concurrently checked, as discussed earlier).

1. there exists $A = \langle a[t], G, Cs, Tc\rangle \in Plan$ such that C is a conjunct in Cs,
2. $\vdash_{TR}$ (finitely) fails to prove that $C \wedge t = \tau \wedge TCS$
3. $A \in c^c_{AS}(S, \tau)$.

Trivially, if $\vdash_{TR}$ and $\vdash_{\Re}$ are correct computational counterparts of $\models_{TR}$ and $\models_{\Re}$, and c^c_{AS} is a correct computational counterpart of c_{AS}, then the given computational counterparts of the checks 1-3 in the specification of c_{PS} are correct wrt the specification itself, and thus an overall correct computation counterpart of c_{PS} can be obtained.

9.2 Computational counterparts for the transitions

The computational counterparts of the transitions are defined via transition rules themselves, obtained from the specifications by replacing calls to capabilities appropriately by calls to their computational counterparts. Below, for each transition, we first summarise the formal specification, as given in the revised D4, and then provide the computational countertpart. With respect to the original D4:

- in the new specification of GI and RE we have taken into account the new split of the state into a non-reactive and reactive parts;
- in the new specification of GI, RE, PI we have taken into account the adjustments to the capabilities that they rely upon;
- in the new specification of GR and PR we have dropped any condition calling the planning capability, similarly to the action and goal selection functions.

9.2.1 Goal Introduction

This transition takes no input (except for the knowledge base of the computee) and returns a completely new set of non-reactive goals, the empty set of reactive goals, and empty sets of (reactive and non-reactive) actions.

Specification of GI

$$(\mathbf{GI}) \qquad \frac{\langle KB, Goals, Plan\rangle}{\langle KB, Goals', \{\}\rangle}\,\tau$$

where

(i) If $KB \models^{\tau}_{GD} \{\}$, then

- $Goals' = Goals^{nr}$

(ii) otherwise, if $KB \models^{\tau}_{GD} Gs$ and $Gs \neq \{\}$, then

- $Goals^{nr'} = \{\langle g[t], \bot, Tc\rangle \mid \langle g[t], Tc\rangle \in Gs\}$
- $Goals' = Goals^{nr'}$

$\vdash_{GI}$: computational counterpart of GI

The computational model for GI consists of two components, dealing with cases (i) and (ii) of the specification, respectively.

$$\frac{\begin{array}{lll} KB_0 \cup KB_{TR} \cup KB_{GD} & \vdash^{\tau}_{GD} & Gs \\ Gs & = & \{\} \\ Goals_1 & = & Goals_1^{nr} \\ Goals_1^{nr} & = & Goals^{nr} \\ Plan_1 & = & \{\} \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{GI} \langle KB, Goals_1, Plan_1\rangle}(\vdash^{\tau}_{GI})$$

$$\frac{\begin{array}{lll} KB_0 \cup KB_{TR} \cup KB_{GD} & \vdash^{\tau}_{GD} & Gs \\ Gs & \neq & \{\} \\ Goals_1 & = & Goals_1^{nr} \cup Goals_1^{r} \\ Goals_1^{nr} & = & \{\langle G, \bot^{nr}, Tc\rangle \mid (G, Tc) \in Gs\} \\ Goals_1^{r} & = & \{\} \\ Plan_1 & = & \{\} \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{GI} \langle KB, Goals_1, Plan_1\rangle}(\vdash^{\tau}_{GI})$$

Clearly, if $\vdash^{\tau}_{GD}$ is correct wrt $\models^{\tau}_{GD}$, then the computational counterpart $\vdash_{GI}$ is correct wrt GI. In section 11.5 we will present a correct realisation of $\vdash^{\tau}_{GD}$.

9.2.2 Reactivity

This transition takes as input the non-reactive part of a state and produces a new state by calling the computee's reactivity and identification of preconditions capabilities. The new state will have a completely new reactive part and the unchanged non-reactive part.

Specification of RE

$$(\mathbf{RE}) \qquad \frac{\langle KB, Goals, Plan\rangle}{\langle KB, Goals', Plan'\rangle}\,\tau$$

where $Goals', Plan'$ are defined as follows.

- If $KB, Goals^{nr}, Plan^{nr} \models^{\tau}_{react} \bot, \bot$, then
  - $Goals' = Goals^{nr}$, and
  - $Plan' = Plan^{nr}$
- otherwise, if $KB, Goals, Plan \models^{\tau}_{react} RGs, RAs$, then let

  $$NewGs^{r} = \{\langle g[t], \bot^{r}, Tc\rangle \mid \langle g[t], Tc\rangle \in RGs\}$$

$$NewAs^r = \{\langle a[t], \bot^r, C, Tc\rangle \mid \langle a[t], Tc\rangle \in RAs \ \wedge \ KB, a[t] \models_{pre} C\}.$$

Then:

- $Goals' = Goals^{nr} \cup NewGs^r$, and
- $Plan' = Plan^{nr} \cup NewGs^r$

$\vdash_{RE}$: computational counterpart of RE

The computational model consists of two components. The first component deals with the case in which no consistent set of reactions is returned by the reactive capability represented as $\langle \bot \ \bot \rangle$ (first bullet of the specification), whereas the second component deals with the case in which a (consistent) set of reactions is returned by the reactive capability (second bullet of the specification).

$$\frac{\begin{array}{lll} Goals & = & Goals^{nr} \cup Goals^r \\ Plan & = & Plan^{nr} \cup Plan^r \\ KB_0 \cup KB_{react}, Goals^{nr}, Plan^{nr} & \vdash^{\tau}_{react} & \langle \bot, \bot \rangle \\ Goals_1 & = & Goals^{nr} \cup Goals^r_1 \\ Plan_1 & = & Plan^{nr} \cup Plan^r_1 \\ Goals^r_1 & = & \{\} \\ Plan^r_1 & = & \{\} \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{RE} \langle KB, Goals_1, Plan_1\rangle}(\vdash^{\tau}_{RE})$$

$$\frac{\begin{array}{lll} Goals & = & Goals^{nr} \cup Goals^r \\ Plan & = & Plan^{nr} \cup Plan^r \\ KB_0 \cup KB_{react}, Goals^{nr}, Plan^{nr} & \vdash^{\tau}_{react} & \langle Gs, As\rangle \\ Gs & \neq & \bot \\ As & \neq & \bot \\ Goals_1 & = & Goals^{nr} \cup Goals^r_1 \\ Plan_1 & = & Plan^{nr} \cup Plan^r_1 \\ Goals^r_1 & = & \{\langle G, \bot^r, Tc\rangle \mid (G, Tc) \in Gs\} \\ Plan^r_1 & = & \{\langle A, \bot^r, P, Tc\rangle \mid (A, Tc) \in As \text{ and } KB_{plan}, A \vdash_{pre} P\} \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{RE} \langle KB, Goals_1, Plan_1\rangle}(\vdash^{\tau}_{RE})$$

Clearly, if $\vdash^{\tau}_{react}$ is correct wrt $\models^{\tau}_{react}$ and $\vdash_{pre}$ is correct wrt $\models_{pre}$, then the computational counterpart $\vdash_{RE}$ is correct wrt RE. In sections 11.3 and 11.2 we will present correct realisarions of $\vdash^{\tau}_{react}$ and $\vdash_{pre}$, reespectively.

9.2.3 Plan Introduction

This transition takes as input a state and a set of goals in the state (that have been selected by the goal selection function) and produces a new state by calling the computee's planning and identification of preconditions capabilities.

Specification of PI

$$(\mathbf{PI}) \qquad \frac{\langle KB, Goals, Plan\rangle \qquad SGs}{\langle KB, Goals', Plan'\rangle}\,\tau$$

where SGs is a non-empty set of goals selected for planning (see section 9.1), and

$Goals' = Goals \cup \bigcup_{G\in SGs} Subg(G)$

$Plan' = Plan \cup \bigcup_{G\in SGs} Pplan(G)$

where, for each $G \in SGs$, the sets $Subg(G)$ and $Pplan(G)$ are obtained as follows.

(i) **Mental goals**: let $\{G_1, \ldots, G_n\} \subseteq SGs$, $n \geq 0$, be the set of all mental goals in SGs. If $n > 0$, let

$$KB, Plan, \{G_1, \ldots, G_n\} \models^{\tau}_{plan} \quad \{\langle G_1, \mathcal{A}_1 s, \mathcal{G}_1 s\rangle \atop \ldots \atop \langle G_n, \mathcal{A}_n s, \mathcal{G}_n s\rangle\}$$

Then for each $i = 1, \ldots, n$,

(i.1) either $\mathcal{G}_i s = \mathcal{A}_i s = \bot$ and $Subg(G_i) = Pplan(G_i) = \{\}$,

(i.2) or $\mathcal{G}_i s \neq \bot, \mathcal{A}_i s \neq \bot$ and
$Subg(G_i) = \{\langle l[t], G_i, T\rangle | (l[t], T) \in \mathcal{G}_i s\}$, and
$Pplan(G_i) = \{\langle a[t], G_i, C, T\rangle | (a[t], T) \in \mathcal{A}_i s \wedge KB, a[t] \models_{pre} C\}$.

(ii) **Sensing goals**: for each sensing goal $G = \langle l[t], G', Tc\rangle \in SGs$,

- $Subg(G) = \{\}$, and
- $Pplan(G) = \langle sense(l[t']), G', C, t' \leq t\rangle$,
  where $KB, sense(l[t']) \models_{pre} C$.

$\vdash_{PI}$: computational counterpart of PI

The computational model for PI consists of one component, dealing jointly with mental and sensing actions.

G_i mental goals		$i = 1, \ldots, n$
G_i	$=$	$\langle l_i[t_i], \_, \_\rangle, i = 1, \ldots, n$
$KB_0 \cup KB_{plan}, Plan, Goals$	$\vdash^{\tau}_{plan}$	$New = \langle\{(l_1[t_1], NAs_1, NGs_1), \ldots, (l_n[t_n], NAs_n, NGs_n)\}\rangle$
$Goals_1$	$=$	$Goals \cup NGs$
$Plan_1$	$=$	$Plan \cup NAs$
NGs	$=$	$\{\langle G, G_i, Tc\rangle \mid (l_i[t_i], NAs_i, NGs_i) \in New$ and $NAs_i, NGs_i \neq \bot$ and $(G, Tc) \in NGs_i\}$
NAs	$=$	$\{\langle A, G_i, P, Tc\rangle \mid (l_i[t_i], NAs_i, NGs_i) \in New$ and $NAs_i, NGs_i \neq \bot$ and $(A, Tc) \in NAs_i$ and $KB_{plan}, A \vdash_{pre} P\}$
SG_j sensing goals		$j = 1, \ldots, m$
SG_j	$=$	$\langle l_i[t_i], G'_i, \_\rangle, j = 1, \ldots, m$
$Plan_2$	$=$	$\{\langle sense(l_i[t'_i]), G'_i, P_i, t'_i \leq t_i) \mid KB_{plan}, sense(l_i[t'_i]) \vdash_{pre} P_i\}$

$$\frac{}{\langle KB, Goals, Plan\rangle, \{G_1, \ldots, G_n\} \cup \{SG_1, \ldots, SG_m\} \vdash^{\tau}_{PI} \langle KB, Goals_1, Plan_1 \cup Plan_2\rangle} (\vdash^{\tau}_{PI})$$

Clearly, if $\vdash^{\tau}_{plan}$ is correct wrt $\models^{\tau}_{plan}$ and $\vdash_{pre}$ is correct wrt $\models_{pre}$, then the computational counterpart $\vdash_{PI}$ is correct wrt PI. In sections 11.1 and 11.2 we will provide correct realisations of $\vdash^{\tau}_{plan}$ and $\vdash_{pre}$.

9.2.4 Sensing Introduction

This transition takes as input a state and a set of fluent literals that are preconditions of some actions in the state (these fluent literals have been selected by the precondition selection function) and produces a new state by adding sensing actions to its *Plan* component. To each sensing action is associated its preconditions obtained by calling the computee's identification of preconditions capability.

Specification of SI

$$(\mathbf{SI}) \qquad \frac{\langle KB, Goals, Plan\rangle \qquad SPs}{\langle KB, Goals, Plan'\rangle}\,\tau$$

where SPs is a non-empty set of preconditions of actions, associated with the parent of these actions, selected for sensing (see section 9.1), and

$$Plan' = Plan \cup \{\langle sense\_precondition(c[t'], G, D, Tc\rangle \mid (c[t], G) \in SPs\}$$

where, for each $(c[t], G)$,

(i) $Tc = (t' < t)$, and

(ii) $KB, sense\_precondition(c[t']) \models_{pre} D$.

$\vdash_{SI}$: computational counterpart of SI

$$\frac{\begin{array}{lll} Plan_1 & = & Plan \cup As \\ As & = & \{\langle sense\_precondition(l_1[nt_1]), G_1, P_1, \{nt_1 < t_1\}\rangle, \\ & & \ldots \\ & & \langle sense\_precondition(l_n[nt_n]), G_n, P_n, \{nt_n < t_n\}\rangle\} \\ KB_{plan}, sense\_precondition(l_i[nt_i]) & \vdash_{pre} & P_i \end{array}}{\langle KB, Goals, Plan\rangle\{(l_1[t_1], G_1)\ldots,(l_n[t_n], G_n)\} \vdash^{\tau}_{SI} \langle KB, Goals, Plan_1\rangle} (\vdash^{\tau}_{SI})$$

Clearly, if $\vdash_{pre}$ is correct wrt $\models_{pre}$, then the computational counterpart $\vdash_{SI}$ is correct wrt SI. In section 11.2 we will provide a correct realisation of $\vdash_{pre}$.

9.2.5 Passive Observation Introduction

This transition updates KB_0 by adding new observed facts reflecting changes in the environment.

Specification of POI

$$(\textbf{POI}) \qquad \frac{\langle KB, Goals, Plan\rangle}{\langle KB', Goals, Plan\rangle}\,\tau$$

where, if $\models^{\tau}_{Env} l_1 \wedge \ldots \wedge l_n, c_1 : a_1[\tau_1] \wedge \ldots \wedge c_k : a_k[\tau_k]$, $n, k \geq 0$, either $n > 0$ or $k > 0$, each l_i being a fluent $g_i[\_]$ or the negation of a fluent $\neg g_i[\_]$, each c_j being the name of a computee and each $a_j[\tau_j]$ being a timed action operator,

$$\begin{aligned} KB'_0 = KB_0 \cup \{&observed(l_1, \tau), \ldots, observed(l_n, \tau)\} \\ \cup\{&observed(c_1, a_1[\tau_1], \tau), \ldots, observed(c_k, a_k[\tau_k], \tau)\}. \end{aligned}$$

$\vdash_{POI}$: computational counterpart of POI

$$\frac{\begin{array}{lll} & \models^{\tau}_{Env} & l_1, \ldots, l_n \\ & \models^{\tau}_{Env} & c_1 : a_1[\tau_1], \ldots, c_m : a_m[\tau_m], \\ & & n \geq 0, m \geq 0, n + m > 0, \\ KB_0^{new} & = & \{observed(l_1, \tau), \ldots, observed(l_n, \tau), \\ & & observed(c_1, a_1[\tau_1], \tau), \ldots, observed(c_m, a_m[\tau_m], \tau)\} \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{POI} \langle KB \cup KB_0^{new}, Goals, Plan\rangle} (\vdash^{\tau}_{POI})$$

Clearly, as this transition relies upon no capability or other functionality (such as constraint solving) except for the sensing capability, and since we assume that the sensing capability serves as its computational counterpart (being external to the model anyway), then the computational counterpart $\vdash_{POI}$ is trivially correct wrt POI.

9.2.6 Active Observation Introduction

This transition updates KB_0 by adding new facts deliberately observed by the computee, which seeks to establish whether some given set of fluents hold or not at a given time. These fluents are selected by the fluent selection function and given as input to the transition.

Specification of AOI

$$(\textbf{AOI}) \qquad \frac{\langle KB, Goals, Plan\rangle \qquad SFs}{\langle KB', Goals, Plan\rangle}\,\tau$$

where $SFs = \{f_1, \ldots, f_n\}$, $n > 0$, is a set of fluents selected for being actively sensed (by the fluent selection function)

$$KB'_0 = KB_0 \cup \bigcup_{i=1,\ldots,n} \mathcal{S}_i$$

where, for each $i = 1, \ldots, n$:

- $\mathcal{S}_i = \{observed(f_i[t_i], \tau)\}$ if $\models^{\tau}_{Env} f_i[\tau]$
- $\mathcal{S}_i = \{observed(\neg f_i[t_i], \tau)\}$ if $\models^{\tau}_{Env} \neg f_i[\tau]$
- $\mathcal{S}_i = \{\}$ if neither $\models^{\tau}_{Env} f_i[\tau]$ nor $\models^{\tau}_{Env} \neg f_i[\tau]$

and each t_i is a fresh time variable.

$\vdash_{AOI}$: computational counterpart of AOI

$$\frac{\begin{array}{lcl} Fs & = & \{f_1, \ldots, f_n\} \\ S_i & = & \{observed(f_i[t_i], \tau)\} \text{ if } \models^{\tau}_{Env} f_i[\tau] \\ S_i & = & \{observed(\neg f_i[t_i], \tau)\} \text{ if } \models^{\tau}_{Env} \neg f_i[\tau] \\ S_i & = & \{\} \text{ if } neither \models^{\tau}_{Env} f_i[\tau] \text{ nor } \models^{\tau}_{Env} \neg f_i[\tau] \\ KB_0^{new} & = & \bigcup_{i=1,\ldots,n} S_i \end{array}}{\langle KB, Goals, Plan\rangle, Fs \vdash^{\tau}_{AOI} \langle KB \cup KB_0^{new}, Goals, Plan\rangle}(\vdash^{\tau}_{AOI})$$

Clearly, as this transition relies upon no capability or other functionality (such as constraint solving) except for the sensing capability, and since we assume that the sensing capability serves as its computational counterpart (being external to the model anyway), then the computational counterpart $\vdash_{AOI}$ is trivially correct wrt AOI.

9.2.7 Action Execution

This transition updates KB_0 recording the execution of actions by the computee. The actions to be executed are selected by the action selection function prior to the transition, and given as input to the transition.

Specification of AE

$$(\textbf{AE}) \qquad \frac{\langle KB, Goals, Plan\rangle \qquad SAs}{\langle KB', Goals, Plan\rangle}\,\tau$$

where SAs is a non-empty set of actions selected for execution (by the action selection function) and for each $A \in SAs$

(i) If A is not a sensing action then

$$KB'_0 = KB_0 \cup \{executed(a[t], \tau)\}.$$

(ii) if A is a sensing action, namely if $A = p(l[t])$ where $p = sense$ or $p = sense\_precondition$, then:

$KB'_0 = KB_0 \cup \{executed(p(l[t]), \tau)\} \cup \mathcal{S}$

where:

- $\mathcal{S} = \{observed(l[t], \tau)\}$ if $\models^{\tau}_{Env} l[\tau]$
- $\mathcal{S} = \{observed(\overline{l[t]}, \tau)\}$ if $\models^{\tau}_{Env} \overline{l[\tau]}$
- $\mathcal{S} = \{\}$ if neither $\models^{\tau}_{Env} l[\tau]$ nor $\models^{\tau}_{Env} \overline{l[\tau]}$

$\vdash_{AE}$: computational counterpart of AE

$$\frac{\begin{array}{lcl} As & = & \{A_1, \ldots, A_n\} \cup \{SA_1, \ldots, SA_m\} \\ A_i & = & \langle a_i[t_i], \_, \_, \_\rangle \text{(non-sensing action)}, i = 1, \ldots, n \\ SA_j & = & \langle p(l_j[s_j]), \_, \_, \_\rangle, p = sense \text{ or } p = sense\_precondition \text{(sensing action)}, j = 1, \ldots, m \\ K_i & = & \{executed(a_i[t_i], \tau)\}, i = 1, \ldots, n \\ H_j & = & \{executed(sense(l_j[s_j]), \tau)\} \cup S_j, j = 1, \ldots, m \\ S_j & = & \{observed(l_j[s_j], \tau)\} \text{ if } \models^{\tau}_{Env} l_j[\tau] \\ S_j & = & \{observed(\overline{l_j[s_j]}, \tau)\} \text{ if } \models^{\tau}_{Env} \overline{l_j[\tau]} \\ S_j & = & \{\} \text{ if } neither \models^{\tau}_{Env} l_j[\tau] \text{ nor } \models^{\tau}_{Env} \overline{l_j[\tau]} \\ KB_0^{new} & = & \bigcup_{i=1,\ldots,n} K_i \cup \bigcup_{j=1,\ldots,m} H_j \end{array}}{\langle KB, Goals, Plan\rangle, As \vdash^{\tau}_{AE} \langle KB \cup KB_0^{new}, Goals, Plan\rangle} (\vdash^{\tau}_{AE})$$

Clearly, as this transition relies upon no capability or other functionality (such as constraint solving) except for the sensing capability, and since we assume that the sensing capability serves as its computational counterpart (being it external to the model anyway), then the computational counterpart $\vdash_{AE}$ is trivially correct wrt AE.

9.2.8 Goal Revision

This transition revises the state by modifying its *Goals* component so that only goals that are still worth achieving and are not achieved yet are kept. It calls the computee's temporal reasoning capability and also the constraint solver.

Specification of GR

$$(\mathbf{GR}) \qquad \frac{\langle KB, Goals, Plan\rangle}{\langle KB, Goals', Plan\rangle}\, \tau$$

where $Goals'$ is the biggest subset of *Goals* consisting of all goals $G = \langle l[t], G', Tc\rangle \in Goals$ such that, given that TCS is the set of all temporal constraints in $Goals'$ and $Plan$,

(i) either $G' = \bot$ or $G' \in Goals'$, and

(ii) there is no total valuation σ such that $\sigma \models_{\Re} Tc \wedge TCS \wedge t \leq \tau$ and $KB \models_{TR} l[t]\sigma$, and

(iii) there exists a total valuation σ such that $\sigma \models_{\Re} Tc \wedge TCS \wedge t > \tau$.

$\vdash_{GR}$: computational counterpart of GR

$$\frac{\begin{array}{c} TCS^{Plan} \text{ is the set of all temporal constraints in } Plan \\ \text{there exists a GR-derivation computing } Goals_1 \text{ from } Goals, TCS^{Plan}, \text{ at time } \tau \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{GR} \langle KB, Goals_1, Plan\rangle}(\vdash^{\tau}_{GR})$$

where a *GR-derivation* computing Gs' from Gs, TCS^{Plan}, at τ, with

- Gs (and thus Gs') a set of goals in the state of a computee,
- TCS^{Plan} a set of temporal constraints (the set of all temporal constraints in the actions in the state), and
- τ a ground time,

is defined as a sequence

$$(Gs_1, TCS_1) \ldots, (Gs_i, TCS_i) \ldots, (Gs_n, TCS_n) \quad n \geq 1$$

such that

- $Gs_1 = \{\langle G[t], G', Tc\rangle \in Gs | G' = \bot^{nr}$ or $G' = \bot^{r}$,
  $\vdash_{TR}$ (finitely) fails to prove $G[t] \wedge Tc \wedge TCS^{Plan} \wedge t \leq \tau$ and
  $\vdash_{\Re} Tc \wedge TCS^{Plan} \wedge t > \tau\}$,

  $TCS_1 = TCS^{Plan}$,

- $Gs_i = \{\langle G[t], G', Tc\rangle \in Gs | G' \in Gs_{i-1}$,
  $\vdash_{TR}$ (finitely) fails to prove $G[t] \wedge Tc \wedge TCS_{i-1} \wedge t \leq \tau$ and
  $\vdash_{\Re} Tc \wedge TCS \wedge t > \tau\}$,

  $TCS_i = TCS_{i-1} \cup TCS(Gs_i)$, where $TCS(S)$ is the set of all temporal constraints in the set S of goals,

- for each $i < n$, $Gs_i \neq \{\}$, and $G_n = \{\}$,

- $Gs' = \bigcup_{i=1,\ldots,n} Gs_i$.

Basically, a derivation extracts from the given (reactive and non-reactive) goal-trees Gs all the goals that are not achieved yet (condition calling $\vdash_{TR}$) and that can still be achieved (condition calling $\vdash_{\Re}$), starting from the roots of the goal-trees, down the trees layer by layer, till reaching the leaves or finding a sub-tree whose root should not be kept. In this process, temporal constraints of goals which are being kept are taken into account as the goals are looked at.

It is easy to see that, if $\vdash_{TR}$ is correct wrt $\models_{TR}$ and $\vdash_{\Re}$ is correct wrt $\models_{\Re}$, then the computational counterpart $\vdash_{GR}$ is correct wrt GR. In section 11.4 we will present a correct realisation of $\vdash_{TR}$.

9.2.9 Plan Revision

This transition revises the state by modifying its $Plan$ component so that only actions that are still relevant and executable are kept. It calls the constraint solver.

Specification of PR

$$(\mathbf{PR}) \qquad \frac{\langle KB, Goals, Plan\rangle}{\langle KB, Goals, Plan'\rangle}\,\tau$$

where $Plan'$ is the biggest subset of $Plan$ consisting of all actions $\langle a[t], G, C, Tc\rangle \in Plan$ such that, given that TCS is the set of all temporal constraints in $Goals$ and $Plan'$,

(i) either $G \in Goals$ or $G = \bot$, and

(ii) there exists a total valuation σ such that $\sigma \models_{\Re} Tc \wedge TCS \wedge t > \tau$, and

(iii) if $a[t] = sense\_precondition(C)$ then $Tc = t < t'$ and there exists an action

$$\langle a'[t'], G', Cs, Tc'\rangle \in Plan'$$

such that C is a conjunct of Cs.

$\vdash_{PR}$: computational counterpart of PR

$$\frac{\begin{array}{l} TCS^{Goals} \text{ is the set of all temporal constraints in } Goals \\ \text{there exists a PR-derivation computing } Plan_1 \text{ from } Plan, Goals, TCS^{Goals}, \text{ at time } \tau \end{array}}{\langle KB, Goals, Plan\rangle \vdash^{\tau}_{PR} \langle KB, Goals, Plan_1\rangle}(\vdash^{\tau}_{PR})$$

where a *PR-derivation* computing As' from As, Gs, TCS^{Goals}, at τ, with

- Gs a set of goals in the state of a computee,
- As (and thus As') a set of actions in the state of a computee,
- TCS^{Goals} a set of temporal constraints (the set of all temporal constraints in the goals in the state), and
- τ a ground time,

is defined as a sequence

$$(As_1, TCS_1) \ldots, (As_i, TCS_i) \ldots, (As_n, TCS_n) \quad n \geq 1$$

such that

- $As_1 = As_1^* \setminus As_1^{sp}$ where
  - $As_1^* = \{\langle A[t], G, P, Tc\rangle \in As | G = \bot^r, \vdash_{\Re} Tc \wedge TCS^{Goals} \wedge t > \tau\}$
  - $As_1^{sp} = \{\langle A[t], G, P, Tc\rangle \in As | G = \bot^r, A = sense\_precondition(C[t]), Tc = t < t', \text{ there exists no } \langle A[t'], G, P, Tc'\rangle \in As_1^* \text{ such that } C[t'] \text{ is a conjunct of } P\}$,

  $TCS_1 = TCS^{Goals}$,

- $As_i = As_i^* \setminus As_i^{sp}$ where
  - $As_i^* = \{\langle A[t], G, P, Tc\rangle \in As | G \in Gs \text{ and } G \text{ has level } i-1, \vdash_{\Re} Tc \wedge TCS_{i-1} \wedge t > \tau\}$
  - $As_i^{sp} = \{\langle A[t], G, P, Tc\rangle \in As | G \in Gs \text{ and } G \text{ has level } i-1, A = sense\_precondition(C[t]), Tc = t < t', \text{ there exists no } \langle A[t'], G, P, Tc'\rangle \in As_i^* \text{ such that } C[t'] \text{ is a conjunct of } P\}$,

  $TCS_i = TCS_{i-1} \cup TCS(As_i)$, where $TCS(\_)$ is defined as for GR,

- for each $i < n$, $As_i \neq \{\}$, and $As_n = \{\}$,

- $As' = \bigcup_{i=1,\ldots,n} As_i$.

Basically, a derivation extracts from the given (reactive and non-reactive) trees all the actions that are still executable and still relevant (namely whose parent goal is still in the state) starting from the roots of the trees, down the trees layer by layer, till reaching the leaves or finding a sub-tree whose root should not be kept. In this process, temporal constraints of actions which are being kept are taken into account as the actions are looked at.

It is easy to see that, if $\vdash_{\Re}$ is correct wrt $\models_{\Re}$, then the computational counterpart $\vdash_{PR}$ is correct wrt PR.

10 Proof Procedures: building blocks for the computational model of the capabilities

The language of the KGP model comprises of the language of Abductive Logic Programming (ALP, see section 6.1) and the language of Logic Programming with Priorities (LPP, see section 6.2). All components of the KGP model are specified within these two extensions of Logic Programming. Hence any computational model of this will be based on computational models of these two underlying frameworks.

In this section, we present a study of the proof theory for each of ALP and LLP with emphasis on the particular proof theory and procedure that we will adopt in the development of the computational models of the different components of the model. We will adopt an extension of the IFF proof procedure (see section 6.1.3) for the ALP components of the model, and an argumentation proof procedure extending [75] for $LPwNF$ (see section 6.2), our chosen framework for the LPP components of the model.

10.1 C-IFF: IFF with handling of constraint predicates

An important feature of the model of the individual computee as given in D4 and of all the components of this model (as well as the society model given in D5) is the need to handle temporal constraints, which are defined by means of constraint predicates.

There are several solutions for handling constraint predicates in abductive logic programming. Among these, one is to see temporal constraints as abducible atoms and to handle them as usual in abductive logic programming using a set of integrity constraints which allow to simplify and solve such constraints. An example of this treatment has been implemented in [78], which is an extension of the IFF-procedure. Another one is the use of an underlying constraint solver seen as a *black box* by the abductive proof procedure and called during a computation. This solution has been implemented in the ACLP system [66] using the theory of CLP (see Section 6.3). This is the solution which we have opted for while extending the IFF proof procedure for our purposes.

10.1.1 C-IFF: Syntax

The syntax of an abductive framework suitable for the C-IFF procedure must take constraint predicates into account. In principle, the exact specification of the constraint language is independent from the definition of the C-IFF procedure, because we are going to use the constraint solver as a *black box* component. However, we are going to restrict ourselves to binary constraint predicates. The constraint language has to include a relation symbol for equality (we are going to write $t_1 \#= t_2$) and it must be closed under complements. A suitable set of constraint predicate symbols would be $\{\#=, \#\neq, \#<, \#\leq, \#>, \#\geq\}$[12]. The range of admissible arguments to constraint predicates again depends on the specifics of the chosen constraint solver $\vdash_{\Re}$ (see Section 6.3). For the above set of predicates, any (arithmetic) term built from variable names, integers, and the function symbols $+$, $-$ and $*$ would be appropriate. Note that we distinguish the *constraint atom* $t_1 \#= t_2$ from the *equality atom* $t_1 = t_2$.[13] Below, we refer to the given set of constraint predicates as C.

[12] Notice that this language is conformant to the one introduced in the KGP model in Section 7

[13] Note that the predicate symbol $=$ is understood as in CET (in practice, t is a Herbrand term) and is distinct from the predicate $\#=$ which is is a constraint atom, interpreted wrt the constraint theory $\Re$.

Atoms are equality atoms, constraint atoms, and atoms "in the usual sense of the word" (e.g. a formula such as $p(X)$). The set of literals is the set of atoms (also called positive literals) together with the set of negated atoms (also called negative literals). Where necessary, we are going to use the terms non-constraint or non-equality atoms or literals to refer to atoms and literals where the predicate symbol is not a constraint predicate or not $=$, respectively.

As with the original IFF procedure, we start from an abductive framework $\langle P, A, IC\rangle$ where P is now a normal logic program where clauses of P and integrity constraints in IC may also contain constraint atoms. However, differently from IFF, we transform the program P into $Comp_{\overline{A \cup C}}(P)$, indicated with P^c, rather than simply $Comp_{\overline{A}}(P)$ (see Section 6.1).

Due to the presence of constraint predicates, a new set of allowedness restrictions must be defined to handle constraint predicates correctly. Indeed, the allowedness restriction for the IFF-procedure given in Section 6.1 is not enough if we take into account constraint predicates. Let us consider the following simple example.

P^c $\quad q(X) \leftrightarrow X \#> 3$

IC $\quad q(X) \Rightarrow p(X)$

Then, in a derivation, e.g. for the empty query, we can obtain

$X \#> 3 \Rightarrow p(X)$

by the application of the unfolding rule. The problem is that X is universally quantified with scope the whole implication. In such a way we obtain an "active rule" meaning "for all values of X such that $X \#> 3$, $p(X)$ must hold". Considering that constraints predicates are handled by the underlying constraint solver and thus are not treated as ordinary, non-abducible predicates, the IFF is not able to handle this kind of implication properly. Simply, we can modify the allowedness condition given for the IFF-procedure in Section 6.1 as follows.

Definition 10.1 (Allowedness restriction for C-IFF).
The allowedness of predicate definitions, integrity constraints and queries is:

1. **CIFF allowed definitions.** *A definition of the form*

$$p(X_1, \ldots, X_n) \leftrightarrow D_1 \vee \ldots \vee D_k,$$

is allowed if for each D_i, every variable distinct from $X_1, \ldots, X_n$ occurring in D_i must also occur in a positive non-constraint/non-equality atom in D_i. Moreover, every variable in a constraint literal must also occur in a positive non-constraint/non-equality atom in D_i.

2. **CIFF allowed integrity constraints.** *An integrity constraint is allowed if every variable occurring in an equality atom, in a constraint atom, in a negative literal in the body or anywhere in the head also occurs in a positive non-constraint/non-equality atom in the body.*

3. **CIFF allowed queries.** *A query is allowed if every variable in it occurs in a non-constraint/non-equality atom of the query itself.*

10.1.2 New rules for the C-IFF procedure

With respect to the IFF-procedure, we need two new rules to handle constraint atoms: *Case Analysis for Constraints* and *Constraint Solving*. The first rule handles constraints occurring in the body of an implication and it is in direct correspondence with the original Case Analysis

rule of the IFF procedure. The second rule handles constraints occurring as atoms in a node and checks their consistency using the underlying CLP solver $\vdash_{\Re}$. The new rules, as the IFF original ones, are equivalence preserving in the sense that a frontier F_{i+1} obtained from a frontier F_i by the application of a rule is logically equivalent to F_i, in the underlying $P^c \cup CET$. This feature, crucial for the soundness, will be discussed in the Appendix (see Section B.1.)

Case Analysis for Constraints

Let F_i be a frontier and let

$$Rest \wedge [(C \wedge B) \Rightarrow A]$$

be a node of F_i where C is a constraint atom. with all variables existentially quantified (or free), B is the rest of the body of the implication and *Rest* is the rest of the node.[14] Then the new frontier F_{i+1} is F_i with the node being replaced by:

$$[Rest \wedge C \wedge [B \Rightarrow A]] \vee [Rest \wedge \overline{C}]$$

where $\overline{C}$ is the complement of C, e.g. if C is $(X\#> 3)$ then $\overline{C}$ is $(X\#\leq 3)$. □

Constraint Solving

Let N be a node in the form of a conjunction (i.e. splitting is not applicable to N), in a frontier F_i and let

- C_N be the set of all constraint atoms which are conjuncts of N, and
- $Eq_N = \{t_1\#= t_2 \mid t_1 = t_2 \text{ is a conjunct of } N\} \cup$
  $\{t_1\#\neq t_2 \mid t_1 = t_2 \rightarrow false \text{ is a conjunct of } N\}$.

Then the frontier F_{i+1} is F_i with the node N replaced by *false* if $C_N \cup Eq_N$ is unsatisfiable by the application of the constraint solver $\vdash_{\Re}$. □

10.1.3 Extracted answers in C-IFF

We use the same notion of (failure and non-failure) *leaf node* as in IFF (see Section 6.1.3). Given an abductive logic program $\langle P, A, IC\rangle$, a query Q and a non-failure leaf node N (obtained from a derivation for Q), we can extract from N an answer (Δ, Γ) where:

- the set Δ composed of all the (possibly non-ground) abducible atoms occurring as conjuncts in N;
- the set Γ composed of all the (possibly non-ground) constraint literals occurring as conjuncts in N and by all the elements of the set Eq_N as defined in the Constraint Solving Rule.[15]

[14] Note that for the allowedness restriction we have that all the variables in C are free or existentially quantified.
[15] With an abuse of notation, we will denote by Γ both the set and the conjunction of the elements of the set.

10.1.4 Correctness of C-IFF

The correctness of C-IFF is stated as follows.

Theorem 10.1 (Success Soundness). *Given an abductive framework $\langle P, A, IC\rangle$, a query Q and a non-failure leaf node N (obtained from a derivation for Q). Let (Δ, Γ) be a computed answer for Q extracted from N and let V be the set of all the variables which occur in Q, Δ and Γ. Then*

1. *there exists a substitution σ over V such that:*

   $\sigma \models_{\Re} \Gamma$, *and*

2. *for each such substitution σ:*

   - $Comp_{\overline{C}}(P \cup \Delta\sigma) \cup CET \models^{3}_{\Re} Q\sigma$
   - $Comp_{\overline{C}}(P \cup \Delta\sigma) \cup CET \models^{3}_{\Re} IC$

The proof of this theorem is a direct consequence of the soundness of IFF, which carries through with respect to the new notion of allowedness, the correctness and the completeness of the constraint solver and thus the equivalence preserving of the newly introduced rewrite rules. We will not give this proof explicitly in this document, as we will use another version of CIFF with dynamic allowedness, introduced in the next section.
We also state another type of soundness in the case that all the leaf nodes of a derivation are failure nodes.

Theorem 10.2 (Failure Soundness). *Given an abductive framework $\langle P, A, IC\rangle$ and a query Q, if there is a finite derivation for Q terminating with no success leaf node, then there exists no answer for Q, namely:*
$Comp_{\overline{C}}(P) \cup CET \cup IC \models^{3}_{\Re} Q \leftrightarrow false.$

10.2 C-IFF with dynamic allowedness

The C-IFF procedure defined in section 10.1 handles constraints and, hence, it caters for a large portion of the abductive event calculus as defined in D4, and used in KB_{plan}, KB_{react} and KB_{TR}. Nonetheless, some parts of the abductive event calculus are not allowed according to the definition given earlier. For example, the following clause

$holds\_at(G, T_2) \leftarrow happens(A, T_1), initiates(A, T_1, G), T_1 < T_2, not\ clipped(T_1, G, T_2)$

is not allowed and therefore cannot be handled by the C-IFF procedure. Fortunately, the allowedness restriction can be relaxed. The original allowedness for the IFF-procedure was proposed with the purpose of avoiding *floundering*, which happens if universally quantified variables are introduced in atomic conjuncts in nodes within a derivation. In the IFF-procedure the problem of floundering may occur (for non-allowed programs) due to the fact that a negative literal is moved as a positive disjunct into the head of an integrity constraint, or the fact that an equality atom may occur in the body of an integrity constraint with universally quantified variables. In both cases, the allowedness restriction ensures that variables which may cause floundering occur also in a positive literal in the body of the integrity constraint. In the C-IFF procedure, the same must be done for the constraint atoms which may cause floundering in the same way as equality atoms.

Instead of defining a more relaxed notion of syntactic allowedness we relax the allowedness restriction in a dynamic way, i.e. we check the allowedness of an integrity constraint during a computation. This dynamic use of the allowedness restriction extends the class of logic programs handled by the procedure and in particular covers the abductive event calculus used in D4.

The dynamic allowedness that we propose avoids the floundering which can occur due to equality or constraint atoms but it does not avoid the floundering negation problem. To deal with this problem, we still impose a (limited form of) static allowedness, by modifying the allowedness restriction given in Definition 10.1, imposing a less restrictive condition on the definitions occurring in P^c. In particular, we keep the parts about Integrity Constraints and Queries in the definition and we modify the part about Definitions in the following way:

Definition 10.2 (Allowedness restriction for dynamic C-IFF).
The allowedness of predicate definitions, integrity constraints and queries is obtained by replacing case 1. of definition 10.1 by the following:

1′. A definition of the form

$$p(X_1, \ldots, X_n) \leftrightarrow D_1 \vee \ldots \vee D_k,$$

is allowed if for each D_i, every variable distinct from $X_1, \ldots, X_n$ occurring in a negative literal in D_i also occurs in a positive non-equality atom in D_i.

In C-IFF the rules which handle variables in an integrity constraint and consider their quantification are the *Case Analysis* rules (the original one and that for constraints) and the *Rewrite Rules for Equality* (in particular case 8).

Both Case Analysis rules are applied only to equality/constraint atoms $X = t$ where X is an existentially quantified variable and so we can trivially argue that such rules are safe and we do not need to modify them.

The only rule that handles universally quantified variables in the condition of an integrity constraint is the Rewrite Rule for Equality (Case 8). This rule states:

> If $X = t$ occurs in the body of an implication, t does not contain X and X is universally quantified, then apply the substitution X/t to the entire implication, deleting the conjunct $X = t$.

As explained above, the original allowedness condition for the IFF-procedure ensures that this rule is never applied to an integrity constraint of the form:

$X = t \wedge B \Rightarrow H$

where X occurs only in the atom $X = t$. With a non-allowed theory we are not able to ensure such a safety condition and we could apply, for example, the Rewrite Rule for Equality (Case 8) to an integrity constraint of the form:

$X = Y \Rightarrow r(X)$

where X, Y are universally quantified variables and r is an abducible predicate, obtaining as result the sentence $\forall Y.r(Y)$ which causes a soundness problem when extracted from a node.

The safety condition obtained by the original allowedness can be also obtained in a dynamic way modifying the Rewrite Rule for Equality (Case 8) as follows:

> If $X = t$ occurs in the body of an implication, t does not contain X, each universally quantified variable in t also occurs in a positive non-constraint/non-equality atom in the body of the implication, and X is universally quantified, then apply the substitution X/t to the entire implication, deleting the conjunct $X = t$.

Now, we do not apply the Rewrite Rule for Equality (Case 8) in the example above because Y is universally quantified and does not occur in a positive non-constraint/non-equality atom elsewhere in the body of the implication.
In addition to incorporating this dynamic allowedness condition, for the purpose of sound answer extraction, we need to define a third type of leaf node.

Definition 10.3 (Don't know node). *A don't know node is a leaf node which contains an implication with only equality/constraint atoms in the body (except when there is just a single equality atom in the body and $false$ in the head).*

The concept of *Don't know node* is not present in the original IFF-procedure. Intuitively, don't know nodes represent computations with abductive frameworks which are not allowed with respect to the IFF allowedness condition of Section 6.1.3. In summary, we distinguish now three types of leaf nodes:

1. Don't know leaf node - as defined in Definition 10.3
2. Failure leaf node - a leaf node which coincides with $false$
3. Success leaf node - a leaf node which is not of type 1. or type 2.

10.2.1 Extracted answers in C-IFF with dynamic allowedness

Using the C-IFF procedure with dynamic allowedness, abductive answers must be extracted only from success leaf nodes, rather than non-failure leaf nodes as in static C-IFF, to cater for the presence of don't know nodes.

Formally, given an abductive logic program $\langle P, A, IC\rangle$, a query Q and a success leaf node N (obtained from a derivation for Q), we can extract a computed answer (Δ, Γ) for Q where Δ and Γ are defined as in Definition 10.1.3.

10.2.2 Correctness of dynamic C-IFF

The following theorems are the analogous of theorems 10.1 and 10.2 for static C-IFF.

Theorem 10.3 (Success Soundness). *Given an abductive framework $\langle P, A, IC\rangle$, a query Q and a success leaf node N (obtained from a derivation for Q), let (Δ, Γ) be a computed answer for Q extracted from N and let V be the set of all the variables which occur in Δ, Γ and Q. Then*

1. *there exists a substitution σ over V such that:*

   $\sigma \models_{\Re} \Gamma$

2. *for each such substitution σ:*

   - $Comp_{\overline{C}}(P \cup \Delta\sigma) \cup CET \models^{3}_{\Re} Q\sigma$
   - $Comp_{\overline{C}}(P \cup \Delta\sigma) \cup CET \models^{3}_{\Re} IC$

Notice that the only difference between the above Theorem and Theorem 10.1 is that "success leaf node" in the former replaces "non-failure leaf node" in the latter. Similarly, the next Theorem is the analogous of Theorem 10.2. Here we talk about finite derivations terminating in leaf nodes which are all failre nodes, as opposed to Theorem 10.2 where we talk about finite derivations terminating with no success leaf node.

Theorem 10.4 (Failure Soundness). *Given an abductive framework $\langle P, A, IC\rangle$ and a query Q, if there is a finite derivation for Q terminating in leaf nodes all of which are failure leaf nodes, then there exists no answer for Q, namely:*

$Comp_{\overline{C}}(P) \cup CET \cup IC \models^{3}_{\Re} Q \leftrightarrow false.$

Finally, we must deal with the final don't know nodes. Given an abductive framework $\langle P, A, IC\rangle$ and a query Q, if there is at least a don't know leaf node, we can safely say that the input is not allowed in the sense of the original static allowedness for C-IFF. Nevertheless, if there is a success leaf node in the derivation, the answer extracted from that node is still a correct answer.

The other case to be considered is the case in which, given an abductive framework $\langle P, A, IC\rangle$ and a query Q, in a derivation for Q there is:

- at least a don't know leaf node
- no success leaf node and
- at least a non-leaf node.

In such case we can say that the input is not allowed in the sense of the original static allowedness for C-IFF but we are not able to say if there is a correct answer or not.

The following theorem shows that if an abductive framework is allowed in the sense of definition 10.1, dynamic C-IFF coincides with static C-IFF, in the sense that no don't know leaf nodes are ever generated.

Theorem 10.5. *Given an abductive framework $\langle P, A, IC\rangle$ and a query Q which are allowed according to Definition 10.1, then any dynamic C-IFF derivation is a static C-IFF derivation and viceversa.*

10.2.2.1 Using C-IFF: notations

In the rest of this document, we will use the C-IFF procedure to realize (part of) some of the computational models of the capabilities of a computee. Given an abductive logic program $\langle P, A, IC\rangle$ and a query Q, we will denote as follows the possible outcomes of (finite) C-IFF derivations.

- $\langle P, A, IC\rangle, Q \vdash_{\text{ciff}} (\Delta, \Gamma)$
  denotes that (Δ, Γ) is an extracted answer for Q, i.e. there exists a derivation for Q with a success leaf node N such that (Δ, Γ) is the answer extracted from N.
- $\langle P, A, IC\rangle, Q \vdash_{\text{ciff}} fail$
  denotes that there is a derivation for Q terminating in leaf nodes all of which are failure leaf nodes.
- $\langle P, A, IC\rangle, Q \vdash_{\text{ciff}} flounder$
  denotes that there is a derivation for Q with no success leaf nodes and at least one don't know node.

10.2.2.2 Implementation of C-IFF

We have implemented C-IFF in SICStus Prolog, using a labeling technique for the constraint solving step. The implementation is described in detail in deliverable D9 [5]. Here, it is worth pointing out that in the implementation we have taken into account some heuristics which help in reducing as much as possible the number of don't know leaf nodes encountered in a derivation. In particular, propagation is given priority with respect to unfolding of atoms in the body of an integrity constraint, so that potential universally quantified variables are instantiated by constants or existentially quantified variables. Some of the benefits of this technique is clarified by the following example of using C-IFF.

10.2.3 C-IFF example

We illustrate C-IFF using the following example, by considering a subset of the abductive event calculus theory used for planning (see Section 11.1). First, we consider the following domain independent rules.

$holds\_at(G, T_2) \leftarrow happens(A, T_1), T_1 < T_2, initiates(A, T_1, G), not\ clipped(T_1, G, T_2)$
$holds\_at(\neg G, T_2) \leftarrow happens(A, T_1), T_1 < T_2, terminates(A, T_1, G), not\ declipped(T_1, G, T_2)$
$clipped(T_1, G, T_2) \leftarrow happens(A,\ T), terminates(A, T,\ G), T_1 \leq T < T_2$
$declipped(T_1, G, T_2) \leftarrow happens(A,\ T), initiates(A, T,\ G), T_1 \leq T < T_2$

Moreover we also consider the following integrity constraint of the abductive event calculus used in D4.

$holds\_at(F, T), holds\_at(\neg F, T) \Rightarrow false$

To get a more readable C-IFF derivation we simplify the above rules and integrity constraints in the following way.

P :
$h(G, T_2) \leftarrow ei(T_1, G), T_1 < T_2, not\ c(T_1, G, T_2)$
$h(\neg G, T_2) \leftarrow et(T_1, G), T_1 < T_2, not\ d(T_1, G, T_2)$
$c(T_1, G, T_2) \leftarrow et(T,\ G), T_1 \leq T < T_2$
$d(T_1, G, T_2) \leftarrow ei(T,\ G), T_1 \leq T < T_2$

IC :
$h(F, T), h(\neg F, T) \Rightarrow false$

Now we transform the logic program P into a completed logic program P^C suitable for the C-IFF procedure accomplishing the semantics given by $Comp_{\overline{A} \cup \overline{C}}(P)$.
P^C :
$h(X, Y) \leftrightarrow ei(T_1, G),\ T_1 < T_2,\ not\ c(T_1, G, T_2),\ X = G,\ Y = T_2$
$h(X, Y) \leftrightarrow et(T_1, G),\ T_1 < T_2,\ not\ d(T_1, G, T_2),\ X = \neg G,\ Y = T_2$
$c(X, Y, Z) \leftrightarrow et(T,\ G),\ T_1 \leq T < T_2,\ X = T_1,\ Y = G,\ Z = T_2$
$d(X, Y, Z) \leftrightarrow ei(T,\ G),\ T_1 \leq T < T_2,\ X = T_1,\ Y = G,\ Z = T_2$

Note that the above completed program is not allowed with respect to definition 10.1. Hence, we need to use the C-IFF with dynamic allowedness. We start a C-IFF derivation from the starting query $[h(p,4), ei(1,p), et(2,p)]$ conjoined with the IC given above (variables in IC are renamed before to conjoin them with the starting query).

$G_0 =$ $h(p,4), ei(1,p), et(2,p),$
$[h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false]$

by Unfolding:

$G_1 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,G),\ T_1 < T_2,\ not\ c(T_1,G,T_2),\ Fs = G,\ Ts = T_2,$
$h(\neg Fs,Ts) \Rightarrow false]$

by Unfolding:

$G_2 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,G),\ T_1 < T_2,\ not\ c(T_1,G,T_2),\ Fs = G,\ Ts = T_2,$
$et(T_3,G1),\ T_3 < T_4,\ not\ d(T_3,G1,T_4),\ Fs = G1,\ Ts = T_4 \Rightarrow false]$

by Rewrite Rules for Eq (twice):

$G_3 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,Fs),\ T_1 < T_2,\ not\ c(T_1,Fs,T_2),\ Ts = T_2,$
$et(T_3,Fs),\ T_3 < T_4,\ not\ d(T_3,Fs,T_4),\ Ts = T_4 \Rightarrow false]$

by Rewrite Rules for Eq (twice):

$G_4 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,Fs),\ T_1 < Ts,\ not\ c(T_1,Fs,Ts),$
$et(T_3,Fs),\ T_3 < Ts,\ not\ d(T_3,Fs,Ts),\ \Rightarrow false]$

by Negation Elimination (twice):

$G_5 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,Fs),\ T_1 < Ts,\ et(T_3,Fs),\ T_3 < Ts \Rightarrow$
$c(T_1,Fs,Ts) \vee d(T_3,Fs,Ts)]$

by Propagation:

$G_6 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,Fs),\ T_1 < Ts,\ et(T_3,Fs),\ T_3 < Ts \Rightarrow$
$c(T_1,Fs,Ts) \vee d(T_3,Fs,Ts)],$
$[T_1 = 1,\ Fs = p,\ T_1 < Ts,\ et(T_3,Fs),\ T_3 < Ts \Rightarrow$
$c(T_1,Fs,Ts) \vee d(T_3,Fs,Ts)]$

by Rewrite Rules for Eq (twice):

$G_7 =$ $h(p,4), ei(1,p), et(2,p),$
$[ei(T_1,Fs),\ T_1 < Ts,\ et(T_3,Fs),\ T_3 < Ts \Rightarrow$
$c(T_1,Fs,Ts) \vee d(T_3,Fs,Ts)],$
$[1 < Ts,\ et(T_3,p),\ T_3 < Ts \Rightarrow$
$c(1,p,Ts) \vee d(T_3,p,Ts)]$

by Propagation:

$$
\begin{aligned}
G_8 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [ei(T_1,Fs),\ T_1 < Ts,\ et(T_3,Fs),\ T_3 < Ts \Rightarrow \\
& \ c(T_1,Fs,Ts) \vee d(T_3,Fs,Ts)], \\
& [1 < Ts,\ et(T_3,p),\ T_3 < Ts \Rightarrow \\
& \ c(1,p,Ts) \vee d(T_3,p,Ts)], \\
& [1 < Ts,\ T_3 = 2, p = p,\ T_3 < Ts \Rightarrow \\
& \ c(1,p,Ts) \vee d(T_3,p,Ts)]
\end{aligned}
$$

by Rewrite Rules for Eq (twice):

$$
\begin{aligned}
G_9 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [ei(T_1,Fs),\ T_1 < Ts,\ et(T_3,Fs),\ T_3 < Ts \Rightarrow \\
& \ c(T_1,Fs,Ts) \vee d(T_3,Fs,Ts)], \\
& [1 < Ts,\ et(T_3,p),\ T_3 < Ts \Rightarrow \\
& \ c(1,p,Ts) \vee d(T_3,p,Ts)], \\
& [1 < Ts,\ 2 < Ts \Rightarrow \\
& \ c(1,p,Ts) \vee d(2,p,Ts)]
\end{aligned}
$$

As we can see, in the frontier $G9$ (consisting of a single node) there is an implication with only constraint atoms in the body which contain a universally quantified variable, namely Ts. The node will eventually lead to a "Don't know" leaf node due to the fact that the completed program P^C is not allowed according with the definition 10.1.
Now, we show another derivation which terminates with a "success" leaf node.

$$
\begin{aligned}
G_0 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false]
\end{aligned}
$$

by Propagation:

$$
\begin{aligned}
G_1 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false], \\
& [Fs = p,\ Ts = 4,\ h(\neg Fs,Ts) \Rightarrow false]
\end{aligned}
$$

by Rewrite Rules for Eq (twice):

$$
\begin{aligned}
G_2 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false], \\
& [h(\neg p,4) \Rightarrow false]
\end{aligned}
$$

by Unfolding:

$$
\begin{aligned}
G_3 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false], \\
& [et(T_1,G),\ T_1 < T_2,\ not\ d(T_1,G,T_2),\ G = p,\ T_2 = 4 \Rightarrow false]
\end{aligned}
$$

by Rewrite Rules for Eq (twice):

$$
\begin{aligned}
G_4 = \quad & h(p,4), ei(1,p), et(2,p), \\
& [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false], \\
& [et(T_1,p),\ T_1 < 4,\ not\ d(T_1,p,4) \Rightarrow false]
\end{aligned}
$$

by Negation Elimination:

$$G_5 = \quad h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)]$$

by Propagation

$$G_6 = \quad h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)],\\ [T_1 = 2,\ p = p,\ T_1 < 4\ \Rightarrow d(T_1,p,4)]$$

by Rewrite Rules for Eq (twice):

$$G_7 = \quad h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)],\\ [2 < 4\ \Rightarrow d(2,p,4)]$$

by Case Analysis for Constraints:

$$G_8 = \quad h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)],\\ [2 \geq 4\ \vee d(2,p,4)]$$

by Splitting:

$$G_9 = \quad h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)],\\ 2 \geq 4$$

$$\vee$$

$$h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)],\\ d(2,p,4)$$

by Constraint Solving:

$$G_{10} = \quad false$$

$$\vee$$

$$h(p,4), ei(1,p), et(2,p),\\ [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false],\\ [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)],\\ d(2,p,4)$$

by Logical Simplification:

$$
\begin{aligned}
G_{11} = \quad & h(p,4), ei(1,p), et(2,p), \\
& [h(Fs,Ts), h(\neg Fs,Ts) \Rightarrow false], \\
& [et(T_1,p),\ T_1 < 4\ \Rightarrow d(T_1,p,4)], \\
& d(2,p,4)
\end{aligned}
$$

This time, the frontier G_{11} (consisting of a single node) does not contain any implication with only constraint atoms in the body and universally quantified variables and the frontier $G9$ will eventually lead to a "success" leaf node.
This oversimplified example shows the importance of using an appropriate order of application of the rewrite rules for C-IFF with dynamic allowedness.

10.3 Proof Procedures for $LPwNF$

This subsection studies the computational model of the framework of $LPwNF$ which in turn will provide computational counterparts of (a) the cycle operation of computees (see section 8) and (b) the goal decision capability of a computee (see section 11.5). Its purpose is to define a sound and complete derivability relation, $\vdash_{pr}$, for the preference reasoning $\models_{pr}$ of $LPwNF$ as defined in [63].

We will build $\vdash_{pr}$ for $LPwNF$ directly from its argumentation based semantics following a standard method for computing argumentation. To do so we first develop, following [75], an abstract argumentation based computational framework that can be realized via different parametric variations of a simple proof theory. This proof theory is given in terms of derivations of trees where each node in a tree contains an argument against its corresponding parent node. The abstract proof theory forms the basis for developing concrete proof procedures for query evaluation, obtained by adopting specific ways of computing attacks in the particular argumentation framework. This then gives us a way to realize $\vdash_{pr}$. The proposed argumentation framework (and its integration with abduction) has been implemented in the *Gorgias* system (see [5]). From this an implementation for $\vdash_{pr}$ is obtained.

The original $LPwNF$ framework and its computational model are extended with *dynamic priorities* as required by the computee model. We also incorporate *abduction* within $LPwNF$, despite the fact that this is not required to realise the basic computee model as currently specified in [63], since it can provide a useful extension of the model to cope more fully with the demands of the Global Computing environment in which computees operate.

10.3.1 Computing $\models_{pr}$ via argumentation: Argumentation frameworks

We first recap some of the basic theory of abstract argumentation and show how we can extend the existing framework of $LPwNF$ with dynamic priorities (and abduction) as required by the KGP model.

Argumentation has recently been shown to be a useful framework for formalizing non-monotonic reasoning and other forms of reasoning (see e.g. [10, 22, 36, 71, 95, 96]). In its abstract form, argumentation can be seen as a form of preference reasoning under some given *relative strength* of the arguments.

In general, an argumentation framework is a pair of a set of arguments and a binary attacking relation between conflicting arguments. Here, an argument is defined as a set of sentences whose role is primarily determined by its relations to other arguments.

Definition 10.4 (Abstract Argumentation Framework). *An argumentation framework*

is a pair $(\mathcal{T}, \mathcal{A})$ *where* $\mathcal{T}$ *is a theory in some background (monotonic) logic, and* $\mathcal{A}$ *is a binary attacking relation on* $2^{\mathcal{T}}$*, i.e.* $\mathcal{A} \subseteq 2^{\mathcal{T}} \times 2^{\mathcal{T}}$*.*

We require that no set of sentences attacks the empty set and that the attacking relation is monotonic and compact [75]. The basic notion of an admissible subset of a theory within the abstract argumentation framework is given as follows.

Definition 10.5 (Admissibility). *Let* $(\mathcal{T}, \mathcal{A})$ *be an argumentation framework. A subset* Δ *of* $\mathcal{T}$ *is admissible iff* Δ *is not self-attacking and for all sets of sentences* A*, if* A *attacks* Δ*, then* Δ *attacks* A*.*

A theory may admit several admissible sets which may be incompatible with each other. Any proof theory for the admissibility extension semantics would therefore need to allow a non-deterministic choice among such sets.

One way to realize the attacking relation is via a notion of *conflict* together with a notion of *strength* of an argument. Conflicting arguments, i.e. arguments that have incompatible (e.g. negative) conclusions between them, may coexist in a given theory. A concrete scheme of the abstract attacking relation can employ an irreflexive strength (or qualification) relation that specifies the scope of the conflict and the strength of the arguments under the given logical framework.

We will study argumentation frameworks which can be formulated in this way via conflicts and strength of arguments, as follows:

Definition 10.6 (Abstract Attacking Relation). *Given* $\phi, \psi \subseteq \mathcal{T}$ *and a strength relation* $\mathcal{Q} \subseteq 2^{\mathcal{T}} \times 2^{\mathcal{T}}$*,* ϕ *attacks* ψ *with respect to* $\mathcal{Q}$*, denoted by* $\phi \xrightarrow{\mathcal{Q}} \psi$*, iff* $conflict(\phi, \psi)$ *and* $(\phi, \psi) \in \mathcal{Q}$*, denoted by* $\phi \succeq_{\mathcal{Q}} \psi$ *or* $\psi \preceq_{\mathcal{Q}} \phi$*.*

Furthermore, we can consider the problem of recovering the strength relation in terms of a given priority relation $<$ on the individual sentences in a theory, whose sets make up the arguments, where $r < r'$ means that r has lower priority than r'. The role of this priority relation is to encode locally the relative strength of rules in the theory. As we will see in the next section, the priority relation can be reasoned about, just like any other predicate, and thus it can be classified as either static or dynamic and first- or higher-order.

10.3.2 Logic programming without negation as failure

The framework of $LPwNF$ can be seen as an argumentation framework of the form described above. We will extend the original framework in two ways: (a) generalize the attacking relation to be dynamic, and (b) integrate abduction. The first extension allows the strength of the arguments to be non static depending on factors that can change and that themselves can form part of the argumentative reasoning. The second extension allows us to deal with missing information that can prevent us from constructing fully supported arguments. The integration of abduction is done by extending the sets of arguments to include assumptions on abducible predicates as additional arguments and assigning them an appropriate strength with respect to the other arguments in the theory.

The background logic of $LPwNF$ is given as follows.

Definition 10.7 (Background Logic). *Formulae in the background logic* $\mathcal{L}$ *of the LPwNF framework are defined as labelled rules of the form,* $label : l \leftarrow l_1, ..., l_n$*, where* $l, l_1, ..., l_n$ *are*

positive or explicit negative literals and label is a functional term. The derivability relation, $\vdash$, of the background logic is given by the single inference rule of modus ponens. This is also denoted by $\models_H$ in some places.

Its strength relation extended so that it can capture dynamic priorities is defined as follows:

Definition 10.8 (Strength Relation via Priorities). *Let $\phi, \psi \subseteq \mathcal{T}$ be two conflicting arguments*[16]*. Then, $\psi \succeq_{DYN} \phi$ iff $(\exists r \in \phi, r' \in \psi : \phi \vdash r' < r) \Rightarrow (\exists r \in \phi, r' \in \psi : \psi \vdash r < r')$.*

Note that in many cases we define the $conflict$ relation between two sets of sentences via an *incompatibility* relation between predicates, that extends explicit negation, so that two sets are in conflict for every pair of incompatible conclusions that they derive respectively. More specific details of this extension to dynamic priorities can be found in [73, 74] together with some general computational properties and examples.

10.3.3 An Example theory of $LPwNF$

Consider the following part of the goal decision knowledge base KB_{GD} of a computee describing its simple policy for deciding how to respond to requests. This has two object-level generation rules one for responding yes and the other for responding no (we are using here some of the notation of the KGP model as described in deliverable [63]):

$r_y(Asker, RequestedNeed, yes)$:
$respond(Myself, Asker, need(RequestedNeed), yes)[ResponseTime] \leftarrow$
$holds\_at(request(Asker, Myself, need(RequestedNeed), RequestTime), T_{now})$,
$currently\_satisfiable(Myself, RequestedNeed)$,
$response\_time(RequestTime, T_{now}, ResponseTime)$.

$r_n(Asker, RequestedNeed, no)$:
$respond(Myself, Asker, need(RequestedNeed), no)[ResponseTime] \leftarrow$
$holds\_at(request(Asker, Myself, need(RequestedNeed), RequestTime), T_{now})$
$incompatible\_current\_needs(Myself, RequestedNeed)$,
$response\_time(RequestTime, T_{now}, ResponseTime)$.

Here the ResponseTime in both rules can be either ground or an existentially quantified time with some constraint to respond within a certain time from the request time. Note that we have:

$incompatible(respond(M, A, need(RN), yes)), respond(M, A, need(RN), no))$.

Left as it is this theory will be unable to decide how to respond when for a particular request both $currently\_satisfiable(Myself, RequestedNeed)$ and $incompatible\_current\_needs(Myself, RequestedNeed)$ hold. Both responses will be equally admissible. In the original $LPwNF$ framework we can express static (i.e. unconditional) priorities on the rules and so we could include in the theory:

$p_1(A, RN) : r_n(A, RN, no) > r_y(A, RN, yes)$

[16] Arguments in $LPwNF$ are taken to be *closed*: i.e. a subset Δ of $\mathcal{T}$ is closed iff it contains no rule whose conditions are not derived (by the background logic) in Δ.

to express the policy of "always refuse if you currently need the requested need yourself".

This though would be too inlfexible and so dynamic priorities could be employed. For example, we can condition the above with the urgency of the need for the requested need. So we would have:

$p_1(A, RN) : r_n(A, RN, no) > r_y(A, RN, yes) \leftarrow urgent(Myself, RN).$
$p_2(A, RN) : r_y(A, RN, yes) > r_n(A, RN, no) \leftarrow urgent(A, RN).$
$p_3(A, RN) : r_y(A, RN, yes) > r_n(A, RN, no) \leftarrow \neg urgent(Myself, RN).$

expressing the policy to "prefer to refuse requests for needs that you urgently need and prefer to accept requests for needs that you do not need urgently or that are urgently needed by the asker". When the requested need is urgent for both the asker and yourself then the preference to refuse is stronger unless the asker is your manager. This is captured with the higher-order priorities:

$c_1(A, RN) : p_1(A, RN) > p_2(A, RN).$
$c_2(A, RN) : p_2(A, RN) > p_1(A, RN) \leftarrow manager(Asker, Myself).$
$d_1(A, RN) : c_2(A, RN) > c_1(A, RN).$

Note that the same admissibility semantics is applied for arguments that involve the priority rules where for any pair of rule names $Rule1$ and $Rule2$ we have: $incompatible(Rule1 > Rule2, Rule2 > Rule1)$.

Finally, consider the above example where the computee does not know if a requested need is urgent for the asker who is its manager. Then the computee would decide, given that it knows that this is urgent to it to refuse the request. But it maybe interested in finding out if there is some possible situation under which he should accept. This can be done by incorporating abduction on some predicates, such as $urgent(Asker, RN)$, for which the computee cannot be expected to have complete information. In the example, the computee would then derive that under the condition (or hypothesis) that this is an urgent need for its manage, it should accept the request. It can then check if this condition holds in the world.

10.3.4 Integrating Abduction

As we have illustrated above with the example, in several cases the admissibility of an argument depends on whether we have or not some background information about the specific case in which we are reasoning. However, this information may be just unknown (or incomplete) and thus we need to find assumptions related to the unknown information under which we can build an admissible argument. Furthermore, this type of information may itself be dynamic and change while the rest of the theory remains fixed. To address this problem we can incorporate abductive reasoning within our argumentation framework.

As usual for abduction in logic programming, we separate a distinguished set of predicates in the language of the theory, called abducible predicates, that carry the incomplete information of the given domain of discourse. Then, given a goal, abduction extends the theory with ground abducibles so that it can satisfy the goal.

To integrate abduction with argumentation, we can consider the abducible facts (ground literals) as a special type of sentences in our theory with the following strength relation on abductive arguments:

Definition 10.9 (Strength Relation via Assumptions). *Let α be an abducible fact and $\phi, \psi \subseteq \mathcal{T}$. Then, $\phi \preceq_{ABD} \psi$ iff $\alpha \in \psi$ for some $\neg\alpha \in \phi$.*

This clearly is a symmetric relation thus showing that amongst them the abducible arguments have equal strength.

Furthermore, we extend the framework to allow us to split the theory into a definitional part consisting of the logic program and an assertional part consisting of a set of *integrity constraints* on the abducible predicates. These are rules in the same form as any other rule, but with an abducible literal in its head. Then the arguments that are given by the integrity constraints are stronger than any individual opposing abductive literal. This is expressed by extending the strength relation as follows.

Definition 10.10 (Strength Relation via Integrity Constraints). *Let α be an abducible literal and $\phi, \psi \subseteq \mathcal{T}$ such that $\alpha \notin \psi$. Then, $\phi \preceq_{IC} \psi$ if and only if $\psi \vdash \alpha$ for some $\neg\alpha \in \phi$.*

10.3.5 Computing Argumentation

In this section we develop, following closely [75], a proof theory for the abstract argumentation framework introduced in the previous section. A proof theory can be used to decide whether a given (variable-free or existentailly quantified with a time variable over some constraints) query holds with respect to the given semantics. The different proof theories are based upon a common computational framework, parametric with respect to the attacking relation. Different instances of the attacking relation correspond to different semantics.

The proof theory then aims to detect whether a given query admits solutions. In this section we will assume that an argument Δ_0 for $\mathcal{G}$ is given and we will concetrate on the questions "Is Δ_0 admissible?", "If not, can Δ_0 be made admissible?", or "Is there an admissible superset Δ of Δ_0?".

The proof theory is based upon the construction of admissible trees via derivation of partial trees. In the trees we will consider, nodes are sets of sentences with each node labelled as 'attack' or 'defence'. In the sequel, we will use the term node to refer both to a location in the tree and to the set of sentences at this location.

Let us now define the notion of derivation of partial trees formally. In this definition we assume a selection strategy identifying a node (in the current partial tree) to be handled next, and we mark nodes that should not be selected further. Moreover, we record culprits chosen in selected (and marked) attack nodes.

Definition 10.11 (Derivation). *A derivation for a set of sentences Δ_0 is a sequence of partial trees $\mathcal{T}_0, ..., \mathcal{T}_n$ such that $\mathcal{T}_0$ consists only of the (unmarked) root Δ_0 labelled as defence, and, given $\mathcal{T}_i (i \geq 0)$, if N is the selected (unmarked) node in $\mathcal{T}_i$ then $\mathcal{T}_{i+1}$ is obtained as follows:*

(α) if N is an attack node, choose a culprit $c \in closure(N)$ and a minimal argument D against c such that D attacks N with respect to some qualification relation. Then $\mathcal{T}_i + 1$ is $\mathcal{T}_i$ where N is marked, c is recorded as the culprit of N, and D is added as the (unmarked) defence node child of N.

(δ) if N is a defence node, $closure(N) \cap culprits(T_i) = \emptyset$, then $\mathcal{T}_{i+1}$ is $\mathcal{T}_i$ where N is marked, the root is extended by N, and if $A_1, ..., A_m (m \geq 0)$ are all minimal attacks against N then $A_1, ..., A_m$ are added as additional (unmarked) attack nodes children of the root.

Definition 10.12. *A* **successful derivation** *for a set of sentences Δ_0 is a derivation of trees $T_1, T_2, ..., T$ for Δ_0 such that all nodes in T are marked and all leaves in T are labelled as defence. If the root of T is Δ, then we say that the derivation computes Δ from Δ_0.*

Theorem 10.6 (Soundness). *Let Δ_0 be a closed set of sentences. If there exists a successful derivation computing Δ from Δ_0, then Δ is admissible and $\Delta_0 \subseteq \Delta$.*

Theorem 10.7 (Completeness). *Let Δ_0 be a closed set of sentences. If Δ is an admissible set of sentences such that $\Delta_0 \subseteq \Delta$, then there exists a successful derivation computing Δ' from Δ_0, such that $\Delta_0 \subseteq \Delta' \subseteq \Delta$ and Δ' is admissible.*

Sketch proofs of these results can be found in Appendix B.2.

10.3.6 A proof procedure for $LPwNF$ and $\models_{pr}$

The proof theory developed in the previous section can be specialized to give a proof procedure for LPwNF by incorporating a specific way of computing minimal attacks.

The construction of an admissible set Δ is done incrementally, starting from a given set of sentences Δ_0, by adding to Δ_0 suitable defences for it. The existence of several admissible sets reflects itself on the existence of several defences for a given Δ_0, and imposes a non-deterministic choice among defences in the proof procedure. However, not every potential defence can be promoted to Δ_0 as shown in [75].

We will assume that, actual nodes result from reducing (by resolution) a query $\mathcal{G}$ into a closed and minimal set that concludes $\mathcal{G}$. Moreover, during computation, we use the special predicate "*not*" to record the non-existence of a priority constraint between rules in an argument and its counterargument. This permits the monotonic growth of the admissible set Δ during the computation. We construct a counterargument for a given node, as formalized below:

Definition 10.13. *Given an argument N, a counterargument $N' \cup S' \cup N'' \cup S''$ of N is obtained as follows:*

> *Basis: Choose a literal $c \in closure(N)$ and construct a closed and minimal set N' such that $N' \cup S' \vdash c'$ where c' is in conflict with c and S' is a minimal set of assumptions.*
>
> *Qualification: Choose any of the following such that $N'' \cup S''$ is a non-empty set:*
>
> - *If there exists an abducible $\alpha \in N$ then $S'' = \{\neg\alpha\}$.*
> - *If there exists an abducible $\alpha \in N$ then N'' is a closed and minimal subset of $\mathcal{T}$ such that $N'' \cup S'' \vdash \neg\alpha$ where S'' is a minimal set of assumptions, $\neg\alpha \notin N''$, and $\neg\alpha \notin S''$.*
> - *If there exist rules $r \in N$ and $r' \in N'$ such that $N \vdash r' < r$ then N'' is a closed and minimal subset of $\mathcal{T}$ such that $N'' \cup S'' \vdash \tau < \tau''$, where S'' is a minimal set of assumptions, $\tau \in N$, and $\tau'' \in N'$. Otherwise, $N'' = \{not(r' < r) \mid r \in N \text{ and } r' \in N'\}$.*
> - *If there exists $not(L) \in N$ for some functional term L then N'' is a closed and minimal subset of $\mathcal{T}$ such that $N'' \cup S'' \vdash L$, where S'' is a minimal set of assumptions.*

Note that, the first three options in the qualification step correspond to the definitions of $\preceq_{ABD}, \preceq_{IC}$, and $\preceq_{DYN}$.

Definition 10.14 (Derivability relations in $LPwNF$). *Let T be a theory in $LPwNF$. Then $T \vdash_{pr}^{cred} L$ iff there exists a subset Δ_0 of T such that $\Delta_0 \vdash L$ and there is a successful derivation of the above proof procedure for Δ_0. The skeptical derivability relation, $T \vdash_{pr} L$, for $LPwNF$ is defined by $T \vdash_{pr}^{cred} L$ and $T \not\vdash_{pr}^{cred} \overline{L}$ for any $\overline{L}$ such that incompatible$(L, \overline{L})$ holds from T under the background logic, $\vdash$, of $LPwNF$.*

The soundness and completeness of these derivability relations for theories of LPwNF whose ground rules that they represent are finite, follow from the equivalence of the above definition of attack with the original attacking relation in $LPwNF$ and the Theorems 10.6 and 10.7 [17].

Theorem 10.8 (Soundness and completeness of $\vdash_{pr}^{cred}$ and $\vdash_{pr}$). *Let T be a finite theory of $LPwNF$. Then derivability relation $\vdash_{pr}^{cred}$ is sound and complete with respect to $\models_{pr}^{cred}$. Hence the skeptical relation $\vdash_{pr}$ is also sound and complete with respect to $\models_{pr}$.*

10.3.7 Implementation of the Proof Procedure for LPwNF

The proposed argumentation-based computational model for $LPwNF$ has been implemented in a general system, namely *Gorgias*, for argumentative deliberation and is available at `http://www.cs.ucy.ac.cy/ nkd/gorgias/.` The computation of an argument in the Gorgias system can be seen as an interleaving of two phases: (a) the reduction of the query to an initial set of rules[18] together with hypotheses on the abducible predicates, and (b) the incremental growth of the initial set to defence against all attacking arguments according to the proof procedures presented above. Details of this implementation and the system can be found in [5].

11 Capabilities computational models

Here we specify the computational models for the capabilities. Proofs of theoretical results regarding these computational models are given in the Appendix.

11.1 Planning

11.1.1 KB_{plan} and specification of $\models_{plan}^{\tau}$: recap

As the knowledge base KB_{plan} used to represent the knowledge required for partial planning we adopt (a variant of) the abductive event calculus, namely $KB_{plan} = \langle P_{plan}, A_{plan}, I_{plan} \rangle$. In the following specification of KB_{plan} we adopt the notational conventions of the abductive event calculus. In particular, an atom of the form $holds\_at(G, T)$ stands for *the fluent G holds at time T* and an atom of the form $happens(A, T)$ stands for *the action A takes place at time T*. In order to link these event calculus formulation to our time fluent and timed operators, we extend the basic theory, originating from the conventional event calculus, with suitable *bridge rules*.

- P_{plan} consists of two parts: *domain-independent rules* and *domain-dependent rules*. In the sequel, we assume that 0 is the initial time.

[17] We are also assuming here that we are using with the above proof procedure for $\vdash_{pr}^{cred}$ a sound and complete realization, $\vdash$, of the background monotonic relation $\models_H$ of $LPwNF$.

[18] The rules are identified by their labels and a partial valuation of the variables for each label.

Domain independent rules

$holds\_at(G, T_2) \leftarrow happens(A, T_1), T_1 < T_2, initiates(A, T_1, G), not\ clipped(T_1, G, T_2)$

$holds\_at(\neg G, T_2) \leftarrow happens(A, T_1), T_1 < T_2, terminates(A, T_1, G), not\ declipped(T_1, G, T_2)$

$holds\_at(G, T) \leftarrow holds\_initially(G), 0 < T, not\ clipped(0, G, T)$

$holds\_at(\neg G, T) \leftarrow holds\_initially(\neg G), 0 < T, not\ declipped(0, G, T)$

$clipped(T_1, G, T_2) \leftarrow happens(A,\ T), terminates(A, T,\ G), T_1 \leq T < T_2$

$declipped(T_1, G, T_2) \leftarrow happens(A,\ T), initiates(A, T,\ G), T_1 \leq T < T_2$

$clipped(T_1, G, T_2) \leftarrow observed(\neg G[\_],\ T), T_1 \leq T < T_2$

$declipped(T_1, G, T_2) \leftarrow observed(G[\_],\ T), T_1 \leq T < T_2$

$holds\_at(G, T_2) \leftarrow observed(G[\_], T_1), T_1 \leq T_2, not\ clipped(T_1, G, T_2)$

$holds\_at(\neg G, T_2) \leftarrow observed(\neg G[\_], T_1), T_1 \leq T_2, not\ declipped(T_1, G, T_2)$

$happens(A, T) \leftarrow executed(A[T'], T), T' = T$

$happens(A(C), T) \leftarrow observed(C, A[\_], T)$

$happens(A, T) \leftarrow assume\_happens(A, T)$

Domain dependent rules

P_{plan} also contains domain-dependent rules defining the predicates *holds_initially*, *initiates*, *terminates*, and *precondition* e.g.

$holds\_initially(at(c, (1, 1)))$

$initiates(go(X, L_1, L_2), T, at(X, L_2)) \leftarrow holds\_at(mobile(X), T)$

$terminates(go(X, L_1, L_2), T, at(X, L_1) \leftarrow holds\_at(mobile(X), T), L_1 \neq L_2$

$precondition(go(X, L_1, L_2), at(X, L_1))$

- A_{plan} consists of the predicate *assume_happens*.
- I_{plan} contains the following domain-independent integrity constraints:

  $holds\_at(F, T), holds\_at(\neg F, T) \Rightarrow false$

  $assume\_happens(A, T) \wedge precondition(A, P) \Rightarrow holds\_at(P, T)$

Given a plan *Plan*, we denote by $\mathcal{EC}(Plan)$ its representation in the event calculus formulation, that is the conjunction

$$\bigwedge_{\langle a[t], \_, \_, Tc\rangle \in Plan} (happens(a, t) \wedge Tc)$$

Similarly, given a set of goals *Goals*, we denote by $\mathcal{EC}(Goals)$ its representation in the event calculus formulation, that is the conjunction

$$\bigwedge_{\langle l[t], \_, Tc\rangle \in Goals} (holds\_at(l, t) \wedge Tc)$$

11.1.1.1 Specification of $\models^{\tau}_{plan}$

Let $S = \langle KB, Goals, Plan\rangle$ be a state, and $\mathcal{G}s$ be the (non-empty) set of mental goals $\{\langle G_1, G'_1, T_1\rangle, \ldots, \langle G_n, G'_n, T_n\rangle\}$. Then:

$$KB, Plan, Goals, \mathcal{G}s \models^{\tau}_{plan} \quad \{\langle G_1, \mathcal{A}_1 s, \mathcal{G}_1 s\rangle, \ldots, \langle G_n, \mathcal{A}_n s, \mathcal{G}_n s\rangle\}$$

where, for each $j = 1, \ldots, n$

- either $\mathcal{A}_j s = \mathcal{G}_j s = \bot$,
- or
  $\mathcal{A}_j s = \{(a^j_1[t^j_1], T^j_1), \ldots, (a^j_{m_j}[t^j_{m_j}], T^j_{m_j})\}$, $m_j \geq 0$, each $a^j_i[t^j_i]$ is a timed operator and T^j_i are temporal constraints and
  $\mathcal{G}_j s = \{(l^j_1[s^j_1], S^j_1), \ldots, (l^j_{k_j}[s^j_{k_j}], S^j_{k_j})\}$, $k_j \geq 0$, each $l^j_i[s^j_i]$ is a timed literal, and S^j_i are temporal constraints,

such that:

(i) if $\mathcal{T}$ is the set of all temporal constraints in $\mathcal{G}_s, \mathcal{A}_1 s, \ldots, \mathcal{G}_1 s, \ldots, \mathcal{A}_n s, \ldots, \mathcal{G}_n s$, together with additional constraints ensuring that each new action must be executable in the future, namely
$\mathcal{T} = \bigcup_{j=1,\ldots,n} T_j \cup \bigcup_{j=1,\ldots,n, \mathcal{A}_j s \neq \bot, \mathcal{G}_j s \neq \bot} \bigcup_{i=1,\ldots m_j} T^j_i \cup \bigcup_{j=1,\ldots,n, \mathcal{A}_j s \neq \bot, \mathcal{G}_j s \neq \bot} \bigcup_{i=1,\ldots k_j} S^j_i \cup \bigcup_{j=1,\ldots,n,\, i=1,\ldots m_j} t^j_i > \tau$
then there exists a total $\Sigma-$valuation σ such that $\sigma \models_{\Re} \mathcal{T} \cup TCS$

(ii) for each $j = 1, \ldots, n$, with $G_j = l_j[t_j]$, such that the sets $\mathcal{A}_j s, \mathcal{G}_j s$ are not $\bot$,
$P_{plan} \wedge [\bigwedge_{i=1,\ldots,m_j} assume\_happens(a^j_i, t^j_i) \wedge \mathcal{EC}(Plan) \wedge \bigwedge_{\ell=1,\ldots,k_j} holds(l^j_\ell, s^j_\ell) \wedge \mathcal{EC}(Goals \setminus Gs)]\sigma \models_{LP} holds\_at(l_j, t_j)\sigma$

(iii) $P_{plan} \wedge [\bigwedge_{j=1,\ldots,n, \mathcal{A}_j s \neq \bot, \mathcal{G}_j s \neq \bot} \bigwedge_{i=1,\ldots,m_j} assume\_happens(a^j_i, t^j_i) \wedge \mathcal{EC}(Plan)]\sigma$
$\wedge [\bigwedge_{j=1,\ldots,n, \mathcal{A}_j s \neq \bot, \mathcal{G}_j s \neq \bot} \bigwedge_{i=1,\ldots,k_j} holds\_at(l^j_i, s^j_i) \wedge \mathcal{EC}(Goals)]\sigma \models_{LP} I_{plan}$

(iv) for each $j = 1, \ldots, n$, such that $\mathcal{A}_j s = \bot$ and $\mathcal{G}_j s = \bot$, there exists no sets X_j, Y_j with $X_j \neq \bot$, $Y_j \neq \bot$ such that the assignment $\mathcal{A}_j s = X_j$ and $\mathcal{G}_j s = Y_j$ satisfies (i), (ii) and (iii) above.

11.1.2 $\vdash^{\tau}_{plan}$: Computational model for planning

The computational model for planning amounts at devising how an abductive proof procedure can be used to generate partial plans for given goals. The basic proof procedure that we are going to use is the C-IFF proof procedure introduced earlier in this document (see Section 10.1). We need to properly state here what is the abductive framework which is going to be used by C-IFF for planning.

Given KB_{plan} as above, we first construct the abductive logic program KB^{+}_{plan} as follows:

$$KB^+_{plan} = \langle P^+_{plan}, A^+_{plan}, I^+_{plan} \rangle$$

where:

- P^+_{plan} is obtained from P_{plan} by adding the following rules:
  $holds\_at(G,T) \leftarrow assume\_holds(G,T)$
  $holds\_at(not\ G,T) \leftarrow assume\_holds(not\ G,T)$
- $A^+_{plan} = A_{plan} \cup \{assume\_holds\} \cup \{time\_now\}$
- I^+_{plan} is obtained from I_{plan} by adding the following constraints:
  (i) $holds\_at(G,T), assume\_holds(not\ G,T) \rightarrow \bot$
  (ii) $assume\_holds(G,T), holds\_at(not\ G,T) \rightarrow \bot$
  (iii) $assume\_happens(A,T), not\ executed(A,T), time\_now(T') \rightarrow T > T'$

The purpose of adding the constraints (i) and (ii) is to make sure that no fluent is assumed to hold at a time when the contrary of the fluent holds. The addition of (iii) ensures that the actions in the resulting plan that have not been executed yet will be executable in the future.

11.1.2.1 $\vdash^{\tau}_{plan}$ for a single goal

For clarity sake, we start from the case in which the set of goals to plan for is a singleton. In this case, the computational model for planning amounts at running the C-IFF procedure using the above abductive logic program KB^+_{plan}, starting from an initial query which depends on:

- the goal to plan for, and
- the current state $\langle KB, Goals, Plan \rangle$

Initial query

Let $S = \langle KB, Goals, Plan \rangle$ be a state, τ be a time point and $G = \langle l[t], \_, Tc \rangle$ be a mental goal. Then, in order to compute the sets $\mathcal{A}s, \mathcal{G}s$ such that

$$KB, Plan, Goals, \{G\} \models^{\tau}_{plan} \quad \{\langle G, \mathcal{A}s, \mathcal{G}s \rangle\ \}$$

we run the C-IFF proof procedure with an initial query composed of the following conjuncts:

1. $holds\_at(l,t) \wedge Tc \wedge not\ assume\_holds(l,t)$
2. $time\_now(\tau)$
3. $assume\_holds(l',t') \wedge Tc'$
   for all goals $\langle l'[t'], \_, Tc' \rangle$ in $Goals \setminus \{G\}$
4. $assume\_happens(a',t') \wedge Tc'$
   for all actions $\langle a[t], \_, Tc \rangle$ in $Plan$.
5. an equality $t' = \tau'$ for each assertion in KB_0 of the form:

$executed(a[t'], \tau')$, or
$observed(l'[t'], \tau')$.

The conjunct 1. represents the goal we are planning for, along with its temporal constraints. Notice that the sub-conjunct *not assume_holds*(l, t) ensures that the naif plan for G consisting of assuming it is not returned (though this is not strictly needed given the specification of $\models^{\tau}_{plan}$). The conjunct 2. represents the current time at which the capability takes place. The conjuncts 3. and 4. represent the current state: notice that in 4. the goal G we are planning for is not assumed to hold.
Finally, the conjuncts in 5. introduce the explicit bindings of the temporal variables corresponding to actions which have already been executed and fluents which have already been observed. These bindings can be retrieved from the KB_0 component of the current knowledge base.
Given a state S and a goal G, we denote by

$$\mathcal{Q}_{plan}(S, G)$$

the query constructed as above.

Application of C-IFF

Let $S = \langle KB, Goals, Plan\rangle$ be a state and G be a goal.

(i) if $KB^{+}_{plan}, \mathcal{Q}_{plan}(S, G) \vdash_{\text{ciff}} fail$ or $KB^{+}_{plan}, \mathcal{Q}_{plan}(S, G) \vdash_{\text{ciff}} flounder$ then

$$\{G\} \vdash^{\tau}_{plan} \bot, \bot$$

(ii) if $KB^{+}_{plan}, \mathcal{Q}_{plan}(S, G) \vdash_{\text{ciff}} (\Delta, \Gamma)$, then:

$$\{G\} \vdash^{\tau}_{plan} \mathcal{G}s, \mathcal{A}s$$

where

- $\mathcal{G}s = \{\langle p[t], Tc\rangle \mid assume\_holds(p, t) \in \Delta \ \wedge \ Tc = \Gamma \downarrow t\}$
  $\setminus \{assume\_holds(p, t) \in \Delta \mid \langle p[t], \_, \_\rangle \in Goals\}$
- $\mathcal{A}s = \{\langle a[t], Tc\rangle \mid assume\_happens(a, t) \in \Delta \ \wedge \ Tc = \Gamma \downarrow t\}\}$
  $\setminus \{assume\_happens(a, t) \in \Delta \mid \langle a[t], \_, \_, \_\rangle \in Plan\}$

where $\Gamma \downarrow t$ is obtained as follows.

$$\Gamma \downarrow t = \bigcup_{i \geq 0} \Gamma_i$$

where

- $\Gamma_0 = \{c \in \Gamma \mid c$ is atomic and t occurs in $c\}$
- $\Gamma_{i+1} = \{c \in \Gamma \mid c$ is atomic and c contains a variable t' occurring in $\Gamma_i\}$

11.1.2.2 $\vdash^{\tau}_{plan}$ for multiple goals

When the planning capability is requested to plan for a set of goals $\{G_1, \ldots, G_n\}$, we run in turn the C-IFF procedure for each goal G_i by taking into account, at each stage i, the partial plans already constructed for the goals up to $i-1$. More precisely, given a state $\langle KB, Goals, Plan\rangle$, a time point τ and a set of goals $\{G_1, \ldots, G_n\}$,

$$\{G_1, \ldots, G_n\} \vdash^{\tau}_{plan} \{PP_1, \ldots, PP_n\}$$

where each partial plan PP_i is obtained as follows:

(i) if $KB^{+}_{plan}, \mathcal{Q}^{i}_{plan}(S, G_i) \vdash_{\text{ciff}} fail$ or $KB^{+}_{plan}, \mathcal{Q}^{i}_{plan}(S, G) \vdash_{\text{ciff}} flounder$ then

$$PP_i = \langle \bot, \bot \rangle$$

(ii) if $\mathcal{Q}^{i}_{plan}(S, G_i) \vdash_{\text{ciff}} (\Delta, \Gamma)$, then:

$$PP_i = \mathcal{G}s^i, \mathcal{A}s^i$$

where

- $\mathcal{G}^i s = \{\langle p[t], Tc\rangle \mid assume\_holds(p,t) \in \Delta \ \wedge \ Tc = \Gamma \downarrow t\}$
  $\setminus \{assume\_holds(p,t) \in \Delta \mid \langle p[t], \_, \_\rangle \in Goals \cup \bigcup_{j<i} \mathcal{G}s^j\}$
- $\mathcal{A}^i s = \{\langle a[t], Tc\rangle \mid assume\_happens(a,t) \in \Delta \ \wedge \ Tc = \Gamma \downarrow t\}\}$
  $\setminus \{assume\_happens(a,t) \in \Delta \mid \langle a[t], \_, \_, \_\rangle \in Plan \cup \bigcup_{j<i} \mathcal{A}s^j\}$

where $\Gamma \downarrow t$ is defined as in the previous section.

The above specification of $\vdash^{\tau}_{plan}$ for multiple goals refers to queries $\mathcal{Q}^{i}_{plan}(S, G_i)$ which we define as follows.

Initial query at stage i

The initial query $\mathcal{Q}^{i}_{plan}(S, G_i)$ that we use to run C-IFF at stage i, depends on the current state $\langle KB, Goals, Plan\rangle$ and on the partial plans already constructed at the previous stages. Each such query contains a fixed subquery which is obtained as in the case of a single goal, namely $\mathcal{Q}_{plan}(S, G_i)$. The further conjuncts of the query are constructed as follows.

6. $assume\_holds(l', t') \wedge Tc'$
   for all goals $\langle l'[t'], \_, Tc'\rangle$ in $\bigcup_{j<i} \mathcal{G}s^j$

7. $assume\_happens(a', t') \wedge Tc'$
   for all actions $\langle a[t], \_, Tc\rangle$ in $\bigcup_{j<i} \mathcal{A}s^j$.

where, for each $j < i$, $PP_j = \langle \mathcal{G}s^j, \mathcal{A}s^j\rangle$ is the partial plan already constructed for G_j. Notice that, whenever $PP_j = \langle \bot, \bot\rangle$, in 6. and 7. above we implicitly view $\bot$ as the empty set of goals and actions, respectively.

11.1.3 Properties of $\vdash^{\tau}_{plan}$ with respect to $\models^{\tau}_{plan}$

We give here the statement of the main theorem which provides us with the soundness of $\vdash^{\tau}_{plan}$ with respect to the specification of $\models^{\tau}_{plan}$ in the case of a single goal (for multiple goals the theorems and proofs are similar). We can prove that, whenever $\vdash^{\tau}_{plan}$ returns an answer which is not $\bot, \bot$, this answer is correct with respect to the specification of planning (see Section 11.1.1).

Theorem 11.1 (Soundness of $\vdash^{\tau}_{plan}$). *Given a state $\langle KB, Goals, Plan\rangle$ and a mental goal $G \in Goals$, if*

$$\{G\} \vdash^{\tau}_{plan} \mathcal{G}s, \mathcal{A}s$$

(with $\mathcal{G}s \neq \bot$ and $\mathcal{A}s \neq \bot$), then

$$\langle KB, Goals, Plan\rangle, \{G\} \models^{\tau}_{plan} \{\langle G, \mathcal{A}s, \mathcal{G}s\rangle\}.$$

The proof of this theorem can be found in the Appendix (see Section C.1).

11.2 Identification of Precondition

The capability $\models_{pre}$ supports the reasoning capability of identifying (observable) *preconditions for the executability* of actions in $Plan$. This capability is used when an action has to be inserted in the plan of the computee, namely within the PI and RE transitions, when they insert into the state the outputs (goals and actions) of $\models^{\tau}_{plan}$ (for mental goals) and $\models^{\tau}_{react}$, within the PI transition, when it inserts into the state sensing actions for sensing goals, within SI, when it inserts into the state actions for sensing the preconditions of other actions.

As presented in D4, this capability does not require a separate part of the knowledge base KB. It is instead defined in terms of the knowledge base for planning, KB_{plan}.

Below, in section 11.2.1, we give the specification of the Identification of Preconditions capability with respect to (a part of) KB_{plan}. More details and motivations about the contents of this section can be found in D4. Then, in section 11.2.2, we define $\vdash_{pre}$ in terms of C-IFF. $\vdash_{pre}$ is illustrated in section 11.2.3. Finally, in section 11.2.4, we state (and prove) soundness and completeness results for $\vdash_{pre}$, wrt $\models_{pre}$ as specified in D4.

Note that the Identification of Preconditions capability is not specified abductively. We are however using C-IFF to provide a computational counterpart to it in order to use the same tool uniformely for as many capabilities as possible. However, note that we could even have used SLD resolution for it.

11.2.1 Specification of $\models_{pre}$

Given a state $\langle KB, Goals, Plan\rangle$ and a timed action operator $a[t]$,

$KB, a[t] \models_{pre} Cs$ iff

- either there exists c such that $P_{plan} \models_{LP} precondition(a, c)$ and $Cs = \bigwedge\{c[t] \mid P_{plan} \models_{LP} precondition(a, c)\}$
- or, otherwise, $Cs = true$. [19]

[19] We assume that *true* is a formula which is always entailed by KB.

11.2.2 $\vdash_{pre}$: Identification of Preconditions via C-IFF

This capability, as we have concretely defined it, involves hardly any reasoning. Indeed, the presence within the event calculus formulation of KB_{plan} of explicit *precondition* statements greatly facilitates the realisation of this capability. Any proof procedure, even SLD-resolution, would work as a computational counterpart of $\models_{pre}$. We use C-IFF instead, as we are already using it for other capabilities.

Let us first define as P_{pre} the set of all domain-dependent rules in P_{plan} defining the predicate *precondition*. Note that P_{pre} is a set of logic programming facts.

Then, the computational counterpart $\vdash_{pre}$ of $\models_{pre}$ can be defined as follows:

$$a[t] \vdash_{pre} Cs \text{ iff } Cs = \bigwedge\{c[t] \mid \langle P_{pre}, \{\}, \{\}\rangle, precondition(a, X) \vdash_{\text{ciff}} (\{\}, \{X = c\})\}$$

11.2.3 Example of $\vdash_{pre}$

Let us consider KB_{plan} with

$precondition(write(a, Y), have(connection))$
$precondition(write(a, Y), know\_address(Y))$

and the timed operator $write(a, b, t)$. Then, $\vdash_{pre}$ will return

$have(connection, t) \wedge know\_address(b, t)$

since

$\langle P_{pre}, \{\}, \{\}\rangle, precondition(write(a, b), X) \vdash_{\text{ciff}} (\{\}, \{X = have(connection)\})$
$\langle P_{pre}, \{\}, \{\}\rangle, precondition(write(a, b), X) \vdash_{\text{ciff}} (\{\}, \{X = know\_address(b)\})$

11.2.4 Correctness and completeness of $\vdash_{pre}$ wrt $\models_{pre}$

Trivially, $\vdash_{pre}$ is correct and complete wrt $\models_{pre}$ if C-IFF is correct and complete wrt $\models_{LP}$. In the specific case of $\langle P_{pre}, \{\}, \{\}\rangle$ C-IFF actually amounts at ordinary IFF, which in turns actually amounts to SLD. Thus, the soundness and completeness results trivially hold.

11.3 Reactivity

11.3.1 KB_{react} and specification of $\models^{\tau}_{react}$: recap

The capability $\models^{\tau}_{react}$ supports the reasoning capability of reacting to stimuli from the external environment as well as to decisions taken while planning. $\langle KB, Goals, Plan\rangle \models^{\tau}_{react} \mathcal{G}s, \mathcal{A}s$ stands for "the set of goals $\mathcal{G}s$ and set of actions $\mathcal{A}s$ are introduced in order to react to some observation recorded in (the KB_0 part of) the given KB or to some goals in $Goals$ and actions in $Plan$".

As the knowledge base KB_{react} used to represent the knowledge required for reactivity we adopt an extension of the knowledge base KB_{plan} as follows.

$KB_{react} = \langle P_{react}, A_{react}, I_{react}\rangle$
where

- $P_{react} = P_{plan}$
- $A_{react} = A_{plan} = \{assume\_happens\}$
- $I_{react} = I_{plan} \cup \mathcal{RR}$

where $\mathcal{RR}$ is a set of *reactive constraints*, of the form

$$Body \rightarrow Reaction \wedge Tc$$

where

- *Body* is a non-empty conjunction of items of the form
  - $observed(l[T])$,
  - $observed(c, a[T'], T'')$,
  - $executed(a[T'])$,
  - $holds\_at(l, T')$, where $l[T']$ is a timed fluent literal,
  - $happens(a, T')$, where $a[T']$ is a timed action operator, and
  - temporal constraints;
- *Reaction* is either $holds\_at(p, T)$, p being a fluent literal, or $assume\_happens(a, T)$, a being an action, and
- Tc are temporal constraints.

All variables in *Body* are implicitly universally quantified over the whole implication. All variables in $Reaction \wedge Tc$ not occurring in *Body* are implicitly existentially quantified on the righthand side of the implication.

Intuitively, a reactive constraint $Body \Rightarrow Reaction \wedge Tc$ is to be interpreted as follows: if (some instantiation of) all the observations in *Body* hold in KB_0 and (some corresponding instantiation of) all the remaining conditions in *Body* hold, then (the appropriate instantiation of) *Reaction*, with associated (the appropriate instantiation of) the temporal constraints Tc, should be added to *Goals* or *Plan* (depending on the nature of *Reaction*).

11.3.1.1 Specification of $\models^{\tau}_{react}$

Given a state $\langle KB, Goals, Plan \rangle$,

$$KB, Goals, Plan \models^{\tau}_{react} \mathcal{G}s, \mathcal{A}s$$

where

- either $\mathcal{G}s$ is a (possibly empty) set of pairs of the form $\langle l[t], Tc \rangle$, $l[t]$ being a timed fluent literal and Tc being a temporal constraint, and $\mathcal{A}s$ is a (possibly empty) set of pairs of the form $\langle a[t], Tc \rangle$, $a[t]$ being a timed action operator and Tc being a temporal constraint,
- or $\mathcal{G}s = \mathcal{A}s = \bot$.

If there exist sets $\mathcal{X}s$ and $\mathcal{Y}s$ satisfying the following conditions:

(i) $\mathcal{X}s$ is a (possibly empty) set of pairs of the form $\langle l[t], Tc\rangle$, $l[t]$ being a timed fluent literal and Tc being a temporal constraint, and $\mathcal{Y}s$ is a (possibly empty) set of pairs of the form $\langle a[t], Tc\rangle$, $a[t]$ being a timed action operator and Tc being a temporal constraint,

(ii) there exists a total Σ-valuation σ such that $\sigma \models_{\Re} TCS^{nr} \wedge \mathcal{T}$, where TCS^{nr} is the conjunction of all temporal constraints in $Goals^{nr}$ and $Plan^{nr}$, and $\mathcal{T}$ is the conjunction of all temporal constraints in $\mathcal{X}s$ and $\mathcal{Y}s$, namely

$TCS^{nr} = \bigcup_{\langle l[t],\_,Tc\rangle \in Goals^{nr}} Tc \cup \bigcup_{\langle a[t],\_,\_,Tc\rangle \in Plan} (Tc \wedge t > \tau)$

and

$\mathcal{T} = \bigcup_{\langle h[t],Tc\rangle \in \mathcal{X}s} Tc \cup \bigcup_{\langle a[t],Tc\rangle \in \mathcal{Y}s} (Tc \wedge t > \tau)$, and

(iii) $P_{react} \wedge \mathcal{EC}(Plan^{nr})\sigma \wedge \mathcal{EC}(Goals^{nr})\sigma \wedge \bigwedge_{\langle l[t],\_\rangle \in \mathcal{X}s} holds\_at(l,t)\sigma \wedge$
$\bigwedge_{\langle a[t],\_\rangle \in \mathcal{Y}s} assume\_happens(a,t)\sigma \models_{LP} I_{react}$

then $\mathcal{G}s = \mathcal{X}s$ and $\mathcal{A}s = \mathcal{Y}s$.
Otherwise, if there exist no such sets $\mathcal{X}s$ and $\mathcal{Y}s$, then $\mathcal{G}s = \mathcal{A}s = \bot$.

11.3.2 $\vdash^{\tau}_{react}$: Computational model for reactivity

Given KB_{react}, we first construct the abductive logic program KB^{+}_{react} as follows:

$$KB^{+}_{react} = \langle P^{+}_{react}, A^{+}_{react}, I^{+}_{react}\rangle$$

where:

- $A^{+}_{react} = A_{react} \cup \{assume\_holds\} \cup \{time\_now\}$
  $= \{assume\_happens, assume\_holds, time\_now\}$
- I^{+}_{react} is obtained from I_{react} by making three changes:

  (I1) replacing the constraint
  $assume\_happens(A,T) \wedge precondition(A,P) \rightarrow holds\_at(P,T)$
  with
  $assume\_happens(A,T) \wedge precondition(A,P) \rightarrow assume\_holds(P,T)$

  (I2) adding three additional integrity constraints:
  $holds\_at(G,T), assume\_holds(not\ G,T) \rightarrow \bot$
  $assume\_holds(G,T), holds\_at(not\ G,T) \rightarrow \bot$
  $assume\_happens(A,T), not\ executed(A,T), time\_now(T') \rightarrow T > T'$

  (I3) replacing every reactive constraint of the form
  $Body(terms1) \rightarrow Reaction(terms2) \wedge Tc(terms3)$
  where $terms1$, $terms2$ and $terms3$ are vectors of terms, and $terms2$ or $terms3$ possibly contain existentially quantified variables, with a new constraint
  $Body(terms1) \rightarrow P(vars')$,
  where $vars'$ are all the variables occurring in $terms1$ which also occur in $terms2$ or $terms3$, namely $vars' = (vars(terms2) \cup vars(terms3)) \cap vars(terms1)$ and P is a new predicate not occurring anywhere else in P_{react} or A_{react}. The new predicates P introduced in this way for such reactive constraints must all be distinct from one another.

- P^+_{react} is obtained from P_{react} by adding the following rules:

  (P1) $holds\_at(G,T) \leftarrow assume\_holds(G,T)$

  (P2) $holds\_at(not\ G,T) \leftarrow assume\_holds(notG,T)$

  (P3) $P(vars') \leftarrow Reaction(terms2) \wedge Tc(terms3)$
  for each reactive constraint of the form
  $Body(terms1) \rightarrow Reaction(terms2) \wedge Tc(terms3)$
  that has been replaced in I^+_{react} by
  $Body(terms1) \rightarrow P(vars')$.

In I^+_{react} the purpose for the change in (I1) is to avoid planning in the reactivity capability. The purpose of the rewriting in (I3) is that C-IFF (as well as the original IFF) does not deal with existentially quantified variables in the heads of integrity constraints, and it does not deal with conjunctions in the heads of integrity constraints. The simple rewriting of (I3) together with the addition (P3) sin P^+_{react} converts the reactive constraints to a form suitable for C-IFF. This is illustrated in the following example.
Consider the reactive rule
$happens(a,T), holds\_at(p,T) \rightarrow happens(b,T') \wedge T' > T$
This is rewritten to give a new constraint
$happens(a,T), holds\_at(p,T) \rightarrow P(T)$
and a new rule
$P(T) \leftarrow happens(b,T') \wedge T' > T$.

The computational model for reactivity amounts at running the C-IFF procedure starting from an initial query which depends on the current state $\langle KB, Goals, Plan\rangle$.

Initial query

Given a state $\langle KB, Goals, Plan\rangle$, and a time point τ, to compute $\mathcal{G}s$, $\mathcal{A}s$ such that

$$KB, Goals, Plan \models^{\tau}_{react} \mathcal{G}s, \mathcal{A}s$$

the initial query to the C-IFF proof procedure will be the conjunction of

1. $time\_now(\tau)$
2. $assume\_holds(l,t) \wedge Tc$
   for all non-reactive goals $\langle l[t], \_, Tc\rangle$ in $Goals$.
3. $assume\_happens(a,t) \wedge Tc$
   for all non-reactive actions $\langle a[t], \_, TC\rangle$ in $Plan$.
4. an equality $t' = \tau'$ for each assertion in KB_0 of the form:

   $executed(a[t'], \tau')$, or
   $observed(l'[t'], \tau')$.

The previous query will be referred to as $\mathcal{Q}_{react}(S)$.

Application of C-IFF

Let $S = \langle KB, Goals, Plan \rangle$ be a state.

(i) if $KB^{+}_{react}, \mathcal{Q}_{react}(S) \vdash_{\text{ciff}} fail$ or $KB^{+}_{react}, \mathcal{Q}_{react}(S) \vdash_{\text{ciff}} flounder$ then

$$\vdash^{\tau}_{react} \bot, \bot$$

(ii) if $KB^{+}_{react}, \mathcal{Q}_{react}(S) \vdash_{\text{ciff}} (\Delta, \Gamma)$,
then:

$$\vdash^{\tau}_{react} \mathcal{G}s, \mathcal{A}s$$

where

- $\mathcal{G}s = \{\langle p[t], Tc\rangle \mid assume\_holds(p,t) \in \Delta \ \wedge \ Tc = \Gamma \downarrow t\}$
  $\setminus \{assume\_holds(p,t) \in \Delta \mid \langle p[t], \_, \_\rangle \in Goals\}$
- $\mathcal{A}s = \{\langle a[t], Tc\rangle \mid assume\_happens(a,t) \in \Delta \ \wedge \ Tc = \Gamma \downarrow t\}\}$
  $\setminus \{assume\_happens(a,t) \in \Delta \mid \langle a[t], \_, \_, \_\rangle \in Plan\}$

where $\Gamma \downarrow t$ is defined as in Section 11.1.2.

11.3.3 Properties of $\vdash^{\tau}_{react}$ with respect to $\models^{\tau}_{react}$

We give here the statement of the theorem which provides us with the soundness of $\vdash^{\tau}_{react}$ with respect to the specification of $\models^{\tau}_{react}$. We can prove that, whenever $\vdash^{\tau}_{react}$ returns an answer which is not $\bot, \bot$, this answer is correct with respect to the specification of reactivity (see Section 11.3.1).

Theorem 11.2 (Soundness of $\vdash^{\tau}_{react}$). *Given a state $\langle KB, Goals, Plan\rangle$, if*

$$\vdash^{\tau}_{react} \mathcal{G}s, \mathcal{A}s$$

(with $\mathcal{G}s \neq \bot$ and $\mathcal{A}s \neq \bot$), then

$$\langle KB, Goals, Plan\rangle \models^{\tau}_{react} \mathcal{G}s, \mathcal{A}s.$$

The proof of this theorem can be found in the Appendix (see Section C.3).

11.4 Temporal Reasoning

This section studies the Temporal Reasoning capability $\models_{TR}$ of the KGP model for computee and proposes a computational model for it. This model is based on the "standard" computational model of abductive reasoning in ALP and can be implemented, for an appropriately restricted class of temporal reasoning knowledge bases, by any sound and complete proof procedure on this class of ALP theories. In particular, we will adopt the C-IFF proof procedure presented in Section 10.1. Formal properties of the computational model are stated and sketch proofs of these are provided in Appendix C.2.

The computational model for Temporal Reasoning is introduced in several steps of increasing complexity. First, the form of the temporal reasoning theories and the specification of this reasoning is recapped and some examples of use are provided. We then introduce a set of

assumptions on the temporal theories which we relax gradually as we develop further the computational model. Some of these extensions are used by other components of the computational model. Others are not used in the current simplified (with respect to the original KGP model in D4) computational model, but represent a valuable support for future extensions, as envisaged in Deliverable D4. For the ease of reading, the latter are detailed in Appendix A. Examples are provided, explaining the different aspects of TR supported by the developed computational model. Finally, the results obtained are summarised.

Our approach is based on the property, which we can call the *desert property*. This property was introduced in [65] to implement the language E. We follow this approach by exploiting the translation of the temporal reasoning theories of computees from the language E. A desert is a time interval without significant events, i.e., in the basic case, without action occurrences. According to the interpretation of the Abductive Event Calculus [107, 42, 108, 72] on which Temporal Reasoning is based, within a desert the theory and the facts that hold in any time point of the desert, are identical at all the time points in the desert. This allows us to restrict our attention to a finite set of time points called *oasis* points,[20] which typically are those points in which changes occur. Exploiting this property the ALP computation becomes ground and thus can be implemented easily through any ALP proof procedure.

11.4.1 KB_{TR} and specification of $\models_{TR}$: recap

Temporal reasoning theories KB_{TR} are ALP theories of an extended form of abductive event calculus translated from the language E. A KB_{TR} theory is an abductive logic program $\langle P_{TR}, A_{TR}, I_{TR}\rangle$ where:

- P_{TR} contains the following domain independent rules, which determine how properties (fluents) are caused by events and how they persist forwards in time (here we assume that 0 is the initial time point, as we have done in Section 11.1, $F, F_1, \ldots F_k$ are fluent literals, A is an action, and C a computee name. $T, T_1, ...$ are time variables.):

  $holds\_at(F, T_2) \leftarrow happens(A, T_1), T_1 < T_2, initiates(A, T_1, F), not\ clipped(T_1, F, T_2)$

  $holds\_at(\neg F, T_2) \leftarrow happens(A, T_1), T_1 < T_2, terminates(A, T, F), not\, declipped(T_1, F, T_2)$

  $holds\_at(F, T_2) \leftarrow observed(F, T_1), T_1 \leq T_2, not\ clipped(T_1, F, T_2)$

  $holds\_at(\neg F, T_2) \leftarrow observed(\neg F, T_1), T_1 \leq T_2, not\ declipped(T_1, F, T_2)$

  $holds\_at(F, T) \leftarrow assume\_holds(F, 0), not\, clipped(0, F, T)$

  $holds\_at(\neg F, T) \leftarrow assume\_holds(\neg F, 0), not\, declipped(0, F, T).$

  $clipped(T_1, F, T_2) \leftarrow happens(A,\ T), terminates(A, T, F), T_1 \leq T < T_2$

  $declipped(T_1, F, T_2) \leftarrow happens(A,\ T), initiates(A, T, F), T_1 \leq T < T_2$

  $happens(A, T) \leftarrow executed(A, T)\ (T \neq 0)$

  $happens(A(C), T) \leftarrow observed(C, A[T], \_)\ (T \neq 0)$

  We also have the rules:

[20] We will see below that this static separation of oasis points can be further refined depending on the query at hand.

$observed(F, T) \leftarrow happens(A, T), precondition(A, F)$
$observed(\neg F, T) \leftarrow happens(A, T), precondition(A, \neg F)$

We have assumed that no events occur at time 0. The predicates $executed/2$, $observed/2$ and $observed/3$ are evaluated from the theory KB_0. Note that the execution of all events is assumed to have been successful (though their initiation or termination of fluents may not be so).

- P_{TR} also contains domain dependent rules for $observed(F, 0)$, which replace $holds\_initially$, and for $initiates$, $terminates$ and $precondition$. Given an action A and a fluent F the user can include either a *strict* effect rule of the form:

  $initiates(A, T, F) \leftarrow holds\_at(L_1, T), \ldots, holds\_at(L_k, T)$

  or (but not both) a *default* effect rule of the form:

  $initiates(A, T, F) \leftarrow holds\_at(F_1, T), \ldots, holds\_at(F_k, T), not\ contrary(F, T)$

  Similarly, for $terminates$.

  The domain-independent axioms for $contrary$ are:

  $contrary(F, T) \leftarrow evidence(\neg F, T_1), T_1 > T, not\ clipped(T, F, T_1)$
  $contrary(\neg F, T) \leftarrow evidence(F, T_1), T_1 > T, not\ declipped(T, F, T_1)$

  with $evidence$ given by:

  $evidence(F, T) \leftarrow observed(F, T)$
  $evidence(\neg F, T) \leftarrow observed(\neg F, T)$

  The rules for $precondition$ take the simple form:

  $precondition(A, F)$

- A_{TR} consists of the all the ground instances of $assume\_holds(FluentLiteral, 0)$

- I_{TR} contains the following integrity constraints

  $holds\_at(F, T), assume\_holds(\neg F, T) \Rightarrow false$
  $holds\_at(\neg F, T), assume\_holds(F, T) \Rightarrow false$
  $holds\_at(F, T), holds\_at(\neg F, T) \Rightarrow false$

We refer to the temporal reasoning theory as KB_{TR}. It includes the domain dependent part of the theory, e.g. $initiates/3$ predicate, and the dynamic part of the knowledge ($observed/2, observed/3, executed/3$ predicates), which we specifically refer to as KB_0, and call a *narrative* or *scenario*. Note that we use for Temporal Reasoning a simplified representation with respect to the model in Deliverable D4. For instance, we represent the observation $observed(a[t], t')$, as $observed(a, t)$, since we do not reason about the "update" time, i.e. the time at which facts are perceived. Given an abductive logic program KB_{TR} as presented above, containing a specific database KB_0 of observations and event occurrences, and a (ground) timed fluent literal $fl[t]$, $\models_{TR}$ can be defined in two alternative ways.

credulous $\models_{TR}$ ($\models_{TR}^{cred}$): $KB_{TR} \models_{TR}^{cred} fl[t]$

iff there exists a set Δ of (ground) atoms in the predicates in A_{TR} such that

1. $P_{TR} \cup \Delta \models_{LP} holds\_at(fl, t)$,
2. $P_{TR} \cup \Delta \models_{LP} I_{TR}$.

Intuitively, condition 1 says that the timed fluent under consideration is entailed by appropriately "completing" the logic program in KB_{TR} by means of a set of assumptions Δ from A_{TR} and condition 2 says that the "completed" $P_{TR} \cup \Delta$ satisfies the integrity constraints in KB_{TR}.

skeptical $\models_{TR}$ ($\models_{TR}^{skep}$): $KB_{TR} \models_{TR}^{skep} fl[t]$ iff

- $KB_{TR} \models_{TR}^{cred} fl[t]$, and
- $KB_{TR} \not\models_{TR}^{cred} \overline{fl}[t]$,

where, if fl is a (positive) fluent literal f, then $\overline{fl} = \neg f$, and, if fl is a (negative) fluent literal $\neg f$, then $\overline{fl} = f$.

In the sequel we will assume that $\models_{TR}$, if not differently specified, amounts to $\models_{TR}^{skep}$.

This specification can be extended for the case where $KB_{TR} \cup KB_0$ is inconsistent as given in deliverable D4 [63]. This is also discussed briefly in Section A.3.

11.4.1.1 Examples of $\models_{TR}$

A simple first example is as follows. Consider the following domain specific knowledge of a computee:

$initiates(switchOn(Bulb), T, light) \leftarrow$
$\quad holds\_at(\neg broken(Bulb), T).$

together with the particular scenario knowledge in KB_0 of:

$executed(switchOn(bulb), 2).$
$observed(\neg light, 4).$

From this we can derive (skeptically) that *light* does not hold at any time and $broken(bulb)$ holds at all times.

A second example is as follows. A computee has the domain specific knowledge:

$initiates(email(Bob, Message), T, has(Bob, Message)) \leftarrow$
$\quad holds\_at(\neg at\_meeting(Bob), T),$
$\quad not\ contrary(has(Bob, Message), T).$

$initiates(email(secretary(Bob), Message), T, has(Bob, Message)) \leftarrow$
$\quad holds\_at(urgent(Message), T),$
$\quad holds\_at(at\_meeting(Bob), T),$
$\quad not\ contrary(has(Bob, Message), T).$

and the scenario knowledge in KB_0:

$executed(email(bob, message), 2).$
$executed(email(secretary(bob), message), 2).$ The email is cced to the secretary.
$observed(urgent(message), 0).$

Then we should conclude (skeptically) that $has(bob, message)$ holds true for any time after 2 despite the fact that we do not know if *bob* is at a meeting or not. Continuing this example suppose that the computee subsequently finds out that:

$observed(at\_meeting(bob), 1).$
$observed(\neg has(bob, message), 4).$

Then the computee will change its conclusion to $\neg has(bob, message)$ holds for any time after 2 recognizing that its email actions have failed to produced their (default) effect.

11.4.2 $\vdash_{TR}$: Computing temporal reasoning via ALP proof procedures

The computational model for $\models_{TR}$ relies on the idea of considering the time line according to its significant points, i.e. a minimal set of points which will guarantee a correct and complete interpretation of what happens and what holds as time passes by. Such points depend on the working hypotheses adopted. As the hypotheses become less restrictive, the computational model becomes less efficient. Intuitively speaking, the temporal reasoning proof procedure restricts its attention to a "projection" of the theory over significant points, which are finite in number (the more restrictive the assumptions the fewer the number of these points, and of the possible projections). The computation can then be restricted to this projected theory that can be explored by means of an abductive proof procedure, such as the C-IFFproof procedure presented in Section 10.1.

We first present the assumptions adopted and how these can be relaxed gradually so that we can develop a "hierarchy" of computational models alongside these.

The full set of initial assumptions on the KB_{TR} theories and queries are as follows.

1. All queries $KB_{TR} \models_{TR} fl[t]$ are fully ground.

2. All the observations (of fluents and actions) in KB_0 have associated a *ground* known time.

3. The theory $KB_{TR} \cup KB_0$ is consistent, i.e. there is at least one set of abducibles Δ such that $KB_{TR} \cup KB_0 \cup \Delta$ satisfies the constraint $holds(FL, T), holds(\neg FL, T) \Rightarrow false$ and the other domain specific integrity constraints if they exist.

4. The number of observed events, i.e. *observed* and *executed* facts in KB_0, is finite (and time flows in discrete steps from the initial time instant 0).

5. The theory $KB_{TR} \cup KB_0$ does not contain the concurrent execution of actions that initiate and terminate the same fluent.

As explained above the theory will be developed gradually in order to relax some of these assumptions. In particular, Section 11.4.3 shows how Assumption 1 can easily be relaxed to allow for more expressive queries. Assumption 2 will also be relaxed in Section A.1 in order to allow events whose occurrence cannot be assigned at a single known time point but at a time point within an interval. This in turn will help up relax Assumption 3 and also allow us to reason with the Plan that the computee is currently committed to execute in the future, as shown in Section A.3. The following hierarchy informally illustrates the above mentioned cases.

Ground Narrative and Ground Queries. Both the narrative part of the theory (i.e. KB_0) and the queries are ground.

This assumption allows us to know exactly the time points at which actions occur, and hence, to be able to "project" and check the universally quantified integrity constraints needed by the theory over a finite number of time points.

Under these initial (relatively strong) assumptions the temporal reasoning capability is suitable for its current use in Goal Decision and simplified forms of the Revision Transitions.

Example 1. *Let us consider a computee in charge of maintenace, having in its Goal Decision theory a rule for repairing broken bulbs. The rule, that will be subject to a given priority not shown here, requires that if a bulb is known to be broken, it has to be changed/repaired. According to the model, the following rule refers to the current time when Goal Decision is applied by means of the special term now.*

$$r_{repair\_bulb}(bulb): \\ repair(bulb)[RT] \leftarrow holds\_at(broken\_bulb, now), \\ repair\_time(now, RT).$$

Goal Decision capability relies on the Temporal Reasoning capability in order to check whether the predicate $holds\_at(broken\_bulb, now)$ of the rule, which refers to the ground time now, does or does not hold with respect to KB_{TR} and the current narrative in KB_0. Supposing that

$$initiates(switch\_on, T, light) \leftarrow holds\_at(neg(broken\_bulb), T).$$

is a domain dependent rule in KB_{TR}, then a ground narrative containing the observations

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$

will imply, via Temporal Reasoning, the ground fact $holds\_at(broken\_bulb, now)$, for any time now bigger or equal to 20.

Ground Narrative, Quantified queries. It is easy to relax from the previous case of assumptions the assumption that queries are ground, and deal with queries that refer to time variables which may be existentially or universally quantified over an interval. According to the same principle, such a quantified query can be proved by proving the query in one significant ground time point of the interval, in the case of existential quantification, and in all the significant ground points of the interval, in case of universal quantification.

This step of extension allows a more general form of the rules in Cycle and Goal Decision and stronger forms of the Revision Transitions.

Example 2. *Let us consider again the example about the maintenace computee, having a more elaborate rule. The rule requires that a broken bulb has to be changed/repaired if a room has been booked in the next five hours. Moreover, the rule fixes the deadline for changing the bulb to within a reasonable time (again, according to the model, when the rule is applied the special term now refers to the ground current time).*

$$\begin{aligned} &r_{repair\_bulb}(bulb): \\ &\quad repair(bulb)[RT] \leftarrow \\ &\qquad holds\_at(broken\_bulb, now), \\ &\qquad holds\_at(booked\_room, T), now < T,\ T < now + 300, \\ &\qquad repair\_time(now, RT). \end{aligned}$$

Goal Decision, in this case, relies on Temporal Reasoning in order to check whether the existential quantified predicate $holds\_at(booked\_room, T), now < T,\ T < now + 300$ does or does not hold. The predicate refers to a time point between now and the next five hours.

Consider at time $now = 30$ the computee has in its KB_0 the following ground narrative:

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$

Then the existential query $holds\_at(booked\_room, T), 30 < T,\ T < 30 + 300$, that will be generated in the computation of Goal Decision for Temporal Reasoning, will succeed (note that the query holds skeptically, since any explanation for $holds\_at(neg(booked\_room), T)$, with $30 < T < 330$, would clash with the observation $observed(booked\_room, 40)$.).

Note here that the last observation $observed(booked\_room, 40)$ means that the computee knows that the room is booked at time 40. It has observed this at some time before now (which is 30), say at 25. As said, the proper D4 syntax would require this to be recorded as $observed(booked\_room[40], 25)$, but we will use the above representation as we do not reason about the "update" times.

Narrative with existential actions, ground (and quantified) queries. This is the more complex case, where the theory allows for actions which may occur at existentially quantified time points. These times of the actions may be constrained but they are not known precisely. The problem is known to be difficult, because of all the consequences that all the possible interleaving of such actions can induce on what is entailed and what is not entailed by the theory. Our approach consists of suitably projecting the quantified variables of the theory, according to their possible temporal constraints, over significant points of the time line. A suitable combination of the techniques used in this and in the previous case, allows quantified queries to be used together with quantified actions in the theory.

We expect that the computational complexity, mainly due to the combinatorial nature of the problem, can be controlled by means of syntactical analysis of the (domain dependent) theory. For instance, we are currently investigating "query relevancy" functions used to prune away from the computation fluents and actions that are independent from the current query.

Example 3. *Let us consider the computee in charge of maintenace facing new information coming from its environment. The new information has been acquired by a Passive Observation (PO) step. Then, after that it has been decided in the previous step to repair the bulb, the cycle may prompt the computee to perform Goal Decision, and consider again the same rule (but now at* $now = 50$*):*

$$
\begin{array}{l}
r_{repair\_bulb}(bulb): \\
\quad repair(bulb)[RT] \leftarrow \\
\qquad holds\_at(broken\_bulb, now), \\
\qquad holds\_at(booked\_room, T), now < T,\ T < now + 300, \\
\qquad repair\_time(now, RT).
\end{array}
$$

Now, the Temporal Reasoning capability is required to prove (skeptically) the query $holds\_at(booked\_room, T), 50 < T,\ T < 50 + 300$*, being at the moment* $now = 50$*. The narrative currently is*

$$
\begin{array}{c}
executed(switch\_on, 10). \\
observed(neg(light), 20). \\
observed(booked\_room, 40). \\
observed(neg(booked\_room), 60).
\end{array}
$$

which is clearly inconsistent, implying that the fluent $booked\_room$ *and the fluent* $neg(booked\_room)$ *hold at the same time, for instance at the time* 60 *(the former by persistence, the latter by direct observation). We assume that this inconsistency has been detected (see Section A.3 how this could be done).*

The inconsistency of the theory arises from the lack of the knowledge of an action, $cancel\_reservation$ *say, which terminated* $booked\_room$*, and could have occurred in between* 40 *and* 60*. Let us suppose, for the moment, that the Planning capability, or Temporal Reasoning itself, can provide such an explanation, and therefore the (full) narrative*

of the computee now becomes:

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$
$$observed(neg(booked\_room), 60).$$
$$executed(cancel\_reservation, T), 40 \leq T,\ T < 60.$$

where T is an existentially quantified variable. The above query $holds\_at(booked\_room, T), 50 < T,\ T < 50 + 300$, is not anymore skeptically entailed by KB_{TR}. Indeed, the action cancel_reservation could have been executed (could be assumed to occur, equivalently) either before or after 50, and, consequently, there can be or cannot be instants between 50 and 350 in which the room is booked. Therefore, the computee does not have anymore (skeptically speaking) the goal of repairing the bulb.

Given this extension, Temporal Reasoning can be used for the full specification of the Revision Transitions where it is required to test fluents in the future. More importantly perhaps, this will allow the computee to reason with narratives that otherwise would make its temporal knowledge base inconsistent. The computee can resolve the inconsistency by assuming that some such existentailly quantified events have occurred. It would then have a consistent theory and will be able to reason with this to draw conclusions.

Resolving Persistence Inconsistency In this final phase we study how to generate, exploiting planning techniques, existentially quantified actions to resolve inconsistency in the temporal theory. We study how to detect *persistence inconsistency* of the theory and how it can be resolved via such minimal assumptions of events.

This aims to weakening the assumption of consistency of the initial theory to simply classical consistency, i.e. inconsistency of the KB_0 alone together with the integrity constraints I_{TR} (see [64] for a study of the different forms of inconsistency in a temporal reasoning theory).

11.4.2.1 A computationally viable interpretation of $\models_{TR}$

The formal definition of $\models_{TR}$ can be reformulated via an entailment-preserving transformation that makes it computationally viable. This entailment-preserving transformation is based on the idea that changes are only possible at specific time points, namely when an action is executed. Under the assumptions from 1 through 5 made in Section 11.4.2, there are no actions in the theory that occur at a non-ground time. This allows us to focus our analysis, in particular as far as (universally quantified) integrity constraints satisfaction is concerned, to those significant points only, which are finite in number. In particular, each derivation is preceded by a preprocessing phase, which instantiates the integrity constraints to a finite number of ground time values, thus making universal quantification over time variables manageable.
The following definition determines the time points of interest.

Definition 11.1. *Given a narrative, KB_0 (or, more generally, a theory KB_{TR}containing KB_0), with executed/2, observed/2 and observed/3 predicates, and a ground fluent literal $fl[t_{n+1}]$ (corresponding to the query $holds\_at(fl, t_{n+1})$), the relative* time line *TL is the (maximal) totally ordered sequence of ground instants*

$$TL = [0 = t_0, t_1, ..., t_n, \{t_{n+1}\}],$$

where $\forall\ i \in [1,\ldots,n]$ $(\exists\ observed(c, a[t_i], \tau_i) \in KB_0 \vee \exists\ executed(a, t_i) \in KB_0)$*, and* $\{t_{n+1}\}$ *means that the time instant of the query may need to be added to the sequence if* t_{n+1} *is a time instant beyond the last observation in* KB_0*.*

Example 4. *Considering the narrative in* KB_0 *presented in the Example 1:*

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$

and the query $happens(broken\_bulb), 30)$*, the relative time line is*

$$[0, 10, 30].$$

Informally speaking, the last condition about the time point of the query is due to the lack of backward persistence of observations, in general, and of the fluent of the query in particular. Even if the theory is consistent in the sense of Assumption 3, the query may be inconsistent with the theory. Considering, if necessary, the query point in the time line, guarantees that the compatibility of the query fluent with the other fluents, possibly holding at previous time instants, is checked at that specific point. (Note that forward persistence guarantees that all the fluents holding after the last observation in KB_0 still hold at the query time point). Otherwise, there would not be a significant point where the fluent of the query is checked against previous fluents, since it does not back persist.

Proposition 11.1. *Given the theory* $\langle P_{TR}, A_{TR}, I_{TR}\rangle$*, a fluent literal* $fl[t]$*, the relative time line* $TL = [t_0, ..., t_{n+1}]$*, and a set* $\Delta \subset A_{TR}$ *of abducibles then:*

$$\forall\ i = 0, ..., n\ \forall\ t \in [t_i{+}1, t_{i+1}]\ \ P_{TR} \cup \Delta \models_{LP} holds\_at(fl, t) \ \Leftrightarrow\ P_{TR} \cup \Delta \models_{LP} holds\_at(fl, t_i{+}1).$$

Consequently, each interval $[t_i + 1, t_{i+1}]$ *is called a* desert*, where nothing changes, and, hence, the point* $t_i + 1$ *is an* oasis*.*

Proposition 11.2. *Given the theory* $\langle P_{TR}, A_{TR}, I_{TR}\rangle$*, a fluent literal* $fl[t]$*, the relative time line* $TL = [t_0, ..., t_{n+1}]$*, and a set* $\Delta \subset A_{TR}$ *of abducibles then:*

$$\forall\ i = 0, ..., n \quad \forall\ t \in [t_i + 1, t_{i+1}] \quad P_{TR} \cup \Delta \models_{LP} I_{TR}(t) \ \Leftrightarrow\ P_{TR} \cup \Delta \models_{LP} I_{TR}(t_i + 1),$$

where $I_{TR}(t_i{+}1)$ *is obtained from* I_{TR} *by grounding all the universally quantified (time) variables of the integrity constraints to the value* $t_i + 1$*.*

Proposition 11.3. *Given the theory* $\langle P_{TR}, A_{TR}, I_{TR}\rangle$*, a fluent literal* $fl[t]$*, the relative time line* $TL = [t_0, ..., t_{n+1}]$*, and a set* $\Delta \subset A_{TR}$ *of abducibles then:*

$$P_{TR} \models_{TR}^{cred} holds\_at(fl, t) \ \Leftrightarrow\ \exists\ \Delta \quad P_{TR} \cup \Delta \models_{LP} holds\_at(fl, t)\ \wedge$$
$$\forall\ i = 0, ..., n\ P_{TR} \cup \Delta \models_{LP} I_{TR}(t_i + 1).$$

11.4.2.2 A computational model for credulous reasoning over deserts and oases

Proposition 11.3, which can be read as

$$\langle P_{TR}, A_{TR}, I_{TR}\rangle \models_{TR}^{cred} fl[t] \Leftrightarrow \langle P_{TR}, A_{TR}, I_{TR}(t_0 + 1) \wedge \cdots \wedge I_{TR}(t_n + 1)\rangle \models_{TR}^{cred} fl[t],$$

provides us with a viable computational model for $\models^{cred}_{TR}$, by permitting us to check the universally quantified integrity constraints, such as $holds(FL, T), holds(\neg FL, T) \Rightarrow false$ and possibly other user defined similar constraints, at oases points only.

Notice that the left hand side of the implication, where the query and the integrity constraints are ground, straightforward corresponds to a computational model once that $\models^{cred}_{TR}$ is replaced by its definition in term of abductive logic programming, and hence by the corresponding computational model, i.e.

$$\langle P_{TR}, A_{TR}, I_{TR}(t_0 + 1) \wedge \cdots \wedge I_{TR}(t_n + 1)\rangle \vdash_{ALP} fl[t],$$

The following specification provides an operational counterpart of this approach by embedding the underlying Abductive Logic Program computational model, and performing the transformation of the theory consisting in appropriately making ground the integrity constraints. This schema will be followed also for the extensions of this core computational model, presented in the rest of this document.

$query\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A) \leftarrow$
$extract\_oases(KB_0, O)$,
$instantiate\_constraints(I_{TR}, O, OI_{TR})$,
$prove\_credulously\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A)$.

$extract\_oases(KB_0, O) \leftarrow$
Works as expected, extracting the time line O from the narrative KB_0.

$instantiate\_constraints(I_{TR}, [t_0, \ldots, t_{n+1}], I_{TR}(t_0 + 1) \wedge \cdots \wedge I_{TR}(t_n + 1))$.

$prove\_credulously\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], [Delta, Theta]) \leftarrow$
$prove\_ALP(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, holds\_at(fl, t), Delta, Theta)$.

$prove\_ALP(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, holds\_at(fl, t), Delta, Theta) \leftarrow$
This predicate calls an abductive proof procedure[21], which works on
the program $\langle P_{TR} \cup KB_0, A_{TR}, OI_{TR}\rangle$ and the query $holds\_at(f, t)$.
It succeeds or fails according to the behaviour of the called procedure.
In particular, the procedure chosen in the overall computational model is the
C-IFF proof procedure, which, in case of success, returns a set of abducibles $Delta$,
and a set $Theta$ of constraints over the (time) variables of the computation.

Definition 11.2 ($\vdash^{cred}_{TR}$). *Given the abductive logic program KB_{TR}, which includes the dynamic knowledge KB_0 and implements the event calculus based temporal reasoning capability, as described in Section 11.4.1, the* computational model *for $\models^{cred}_{TR}$, indicated as $\vdash^{cred}_{TR}$, is defined as follows:*

$$KB_{TR} \vdash^{cred}_{TR} fl[t] \Leftrightarrow query\_credulous\_TR(KB_{TR}, KB_0, fl[t], A).$$

Theorem 11.3 ($\models^{cred}_{TR} \Leftrightarrow \vdash^{cred}_{TR}$). *Assuming that the abductive proof procedure used in the definition of the predicate $query\_credulous\_TR/4$ is correct and complete, the computational model $\vdash^{cred}_{TR}$ is correct and complete with respect to the formal model $\models^{cred}_{TR}$:*

$$KB_{TR} \models^{cred}_{TR} fl[t] \Leftrightarrow KB_{TR} \vdash^{cred}_{TR} fl[t].$$

[21] Note that a simple ALP proof procedure for ground abduction suffices here.

11.4.2.3 A computational model for skeptically reasoning over deserts and oases

The definition of a computational model for $\models_{TR}^{skep}$can be straightforwardly derived by its definition, as illustrated by the *query_skeptically_TR*/4 predicate definition.

Definition 11.3 ($\vdash_{TR}^{skep}$). *Given the abductive logic program KB_{TR}, which includes the dynamic knowledge KB_0 and implements the event calculus based temporal reasoning capability, as described in Section 11.4.1, the* computational model *for $\models_{TR}^{skep}$, indicated as $\vdash_{TR}^{skep}$, is defined as follows:*

$$KB_{TR} \vdash_{TR}^{skep} \ fl[t] \ \Leftrightarrow \ query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], A).$$

where the predicate query_skeptically_TR/4 is given by

$$\begin{array}{l} query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], A) \ \leftarrow \\ \quad query\_credulously\_TR(KB_{TR}, KB_0, fl[t], A), \\ \quad negate\_fluent(fl[t], Nfl[t]), \\ \quad not\ query\_credulously\_TR(KB_{TR}, KB_0, Nfl[t], \_), \end{array}$$

where, if $fl[t]$ is a positive fluent $f[t]$, then $Nfl[t] = \neg f[t]$, and, if $fl[t]$ is a negative fluent literal $\neg f[t]$, then $Nfl[t] = f[t]$.

Theorem 11.4 ($\models_{TR}^{skep} \Leftrightarrow \vdash_{TR}^{skep}$). *Assuming that the abductive proof procedure used in the definition of the predicate query_credulous_TR/4 is correct and complete, the computational model $\vdash_{TR}^{skep}$is correct and complete with respect to the formal model $\models_{TR}^{skep}$:*

$$KB_{TR} \models_{TR}^{skep} \ fl[t] \ \Leftrightarrow \ KB_{TR} \vdash_{TR}^{skep} \ fl[t].$$

11.4.3 Reasoning with non-ground queries

In this section we show how Assumption 1 of Section 11.4.2 can be relaxed. Given KB_{TR} as defined in Section 11.4.1, and a ground time interval $[a, b]$, with $a \leq b$, we define the following entailment relations:

$$KB_{TR} \ \models_{TR}^{cred^{ngq}} \ \exists\, T \in [a, b] \ fl[T], \tag{1}$$

$$KB_{TR} \ \models_{TR}^{cred^{ngq}} \ \forall\, T \in [a, b] \ fl[T], \tag{2}$$

$$KB_{TR} \ \models_{TR}^{cred^{ngq}} \ fl[t], \tag{3}$$

where (1) is read as

$$\exists\, t \ \text{ ground } t \in [a, b] \ \wedge \ KBtr \ \models_{TR}^{cred} \ fl[t],$$

(2) is read as

$$\forall\, t \ \text{ ground } t \in [a, b] \ KBtr \ \models_{TR}^{cred} \ fl[t],$$

and (3) is read as

$$KBtr \ \models_{TR}^{cred} \ fl[t].$$

11.4.3.1 Credulously reasoning with non-ground queries

A simple entailment-preserving transformation, based on the same idea of deserts and oases, provides a computational model to deal with such extended queries with respect to the core ground theory previously discussed. In order to prove that exists a time point in an interval in which a given fluent literal holds, it is sufficient to prove that the fluent literal holds in some desert which intersects the interval, and hence in some of the oases of such deserts, as stated by the following proposition.

Proposition 11.4. *Given the theory KB_{TR}, a fluent literal $fl[t]$, the relative time line $TL = [t_0, ..., t_{n+1}]$, and a ground time interval $[a, b]$, it holds*

$$\exists\ t\ ground\ t \in [a,b]\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t]$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1].$$

Similarly, in order to prove that for all the time points in an interval a given fluent literal holds, it is sufficient to prove that the fluent literal holds in all the deserts which intersect the interval, and hence in all the oases of such deserts, as stated by the following proposition.

Proposition 11.5. *Given the theory KB_{TR}, a fluent literal $fl[t]$, the relative time line $TL = [t_0, ..., t_{n+1}]$, and a ground time interval $[a, b]$, it holds*

$$\forall\ t\ ground\ t \in [a,b]\ KBtr\ \models_{TR}^{cred}\ fl[t]$$
$$\Leftrightarrow$$
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1].$$

Propositions 11.4 and 11.5 directly provide a computational model for the extended theory to non-ground queries, as formalised by the following program.

$query\_ngq\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, exists([a,b], fl[t]), A)\ \leftarrow$
$\quad extract\_oases(KB_0, O),$
$\quad instantiate\_constraints(I_{TR}, O, OI_{TR}),$
$\quad get\_intersecting\_oases([a,b], O, List\_of\_oases),$
$\quad member(Ti, List\_of\_oases),$
$\quad prove\_credulously\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[Ti+1], A).$

$query\_ngq\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, forall([a,b], fl[t]), A)\ \leftarrow$
$\quad extract\_oases(KB_0, O),$
$\quad instantiate\_constraints(I_{TR}, O, OI_{TR}),$
$\quad get\_intersecting\_oases([a,b], O, List\_of\_oases),$
$\quad foreach\_member\_prove\_credulously\_TR(List\_of\_oases, \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A).$

$query\_ngq\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A)\ \leftarrow$
$\quad query\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A).$

$foreach\_member\_prove\_credulously\_TR([], \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A).$

$foreach\_member\_prove\_credulously\_TR([Ti|Rest\_Tis], \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], [A1, A2])\ \leftarrow$

$prove\_credulously\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[Ti+1], A1)$,
$foreach\_member\_prove\_credulously\_TR(Rest\_Tis, \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A2)$.

$get\_intersecting\_oases([a,b], O, List\_of\_oases) \leftarrow$
Works as expected. Given the interval $[a,b]$ and the time line O,
it returns the list of oases $[t_1, .., t_n]$, such that $\forall\ i\ [t_i+1, z_i] \cap [a,b] \neq \emptyset$,
with z_i the end of the desert started at $t_i + 1$.

Definition 11.4 ($\vdash_{TR}^{cred^{ngq}}$). *Given the abductive logic program KB_{TR}, which includes the dynamic knowledge KB_0 and implements the event calculus based temporal reasoning capability, as described in Section 11.4.1, the* computational model *for $\models_{TR}^{cred^{ngq}}$, indicated as $\vdash_{TR}^{cred^{ngq}}$, is defined as follows:*

$$\begin{aligned}
KB_{TR} \vdash_{TR}^{cred^{ngq}} \exists\ T \in [a,b]\ fl[T] &\Leftrightarrow query_ng_credulous_TR(KB_{TR}, KB_0, \exists\ T \in [a,b]\ fl[T], A)\\
KB_{TR} \vdash_{TR}^{cred^{ngq}} \forall\ T \in [a,b]\ fl[T] &\Leftrightarrow query_ng_credulous_TR(KB_{TR}, KB_0, \forall\ T \in [a,b]\ fl[T], A)\\
KB_{TR} \vdash_{TR}^{cred^{ngq}} fl[t] &\Leftrightarrow query_ng_credulous_TR(KB_{TR}, KB_0, fl[t], A)
\end{aligned}$$

Theorem 11.5 ($\models_{TR}^{cred^{ngq}} \Leftrightarrow \vdash_{TR}^{cred^{ngq}}$). *Assuming that the abductive proof procedure used in the definition of the predicate $query\_credulous\_TR/4$ is correct and complete, the computational model $\vdash_{TR}^{cred^{ngq}}$ is correct and complete with respect to the formal model $\models_{TR}^{cred^{ngq}}$:*

$$\begin{aligned}
KB_{TR} \models_{TR}^{cred^{ngq}} \exists\ T \in [a,b]\ fl[T] &\Leftrightarrow KB_{TR} \vdash_{TR}^{cred^{ngq}} \exists\ T \in [a,b]\ fl[T]\\
KB_{TR} \models_{TR}^{cred^{ngq}} \forall\ T \in [a,b]\ fl[T] &\Leftrightarrow KB_{TR} \vdash_{TR}^{cred^{ngq}} \forall\ T \in [a,b]\ fl[T]\\
KB_{TR} \models_{TR}^{cred^{ngq}} fl[t] &\Leftrightarrow KB_{TR} \vdash_{TR}^{cred^{ngq}} fl[t]
\end{aligned}$$

Note that at this stage it is easy to produce an *answer substitution* to existential queries. For the credulous query $\exists\ T \in [a,b]\ fl[T]$ for every oasis point A computed by $query\_ngq\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, exists([a,b], fl[t]), A)$ we return as the answer substitution the intersection of the desert interval that contains the oasis point A with the interval $[a,b]$ of the query. For each such computed oasis point A these answer substitution intervals can be given separately or they can be joined together to give one answer.

11.4.3.2 Skeptically reasoning with non-ground queries

The definition of skeptically reasoning with non-ground queries follows the intuition that a fluent literal skeptically proved must hold irrespectively of any set of assumption that can be used to prove it, or its negation. The following definition formalises this intuition. A computational model will then be provided, exploiting, as usual, the deserts and oases approach.

Given KB_{TR} as defined in Section 11.4.1, and a ground time interval $[a,b]$, with $a \leq b$, the skeptical entailment relations, defined over ground and non-ground queries as follows,

$$KB_{TR} \models_{TR}^{skep^{ngq}} \exists\ T \in [a,b]\ fl[T], \tag{4}$$
$$KB_{TR} \models_{TR}^{skep^{ngq}} \forall\ T \in [a,b]\ fl[T], \tag{5}$$
$$KB_{TR} \models_{TR}^{skep^{ngq}} fl[t], \tag{6}$$

have, respectively, the following semantics

$$\exists\ t\ \ \text{ground}\ t \in [a,b]\ \wedge\ KBtr\ \models_{TR}^{skep}\ \ fl[t],$$
$$\forall\ t\ \ \text{ground}\ t \in [a,b]\ KBtr\ \models_{TR}^{skep}\ \ fl[t],$$
$$KB_{TR}\ \models_{TR}^{skep}\ \ fl[t].$$

In order to prove that exists a time point in an interval in which a given fluent literal skeptically holds, it is sufficient to prove that the fluent literal skeptically holds in some desert which intersects the interval, and hence in some of the oases of such deserts, as stated by the following proposition.

Proposition 11.6. *Given the theory KB_{TR}, a fluent literal $fl[t]$, the relative time line $TL = [t_0, ..., t_{n+1}]$, and a ground time interval $[a, b]$, it holds*

$$\exists\ t\ \textit{ground}\ t \in [a,b]\ \wedge\ KBtr\ \models_{TR}^{skep}\ \ fl[t]$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i + 1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{skep}\ \ fl[t_i + 1].$$

Similarly, in order to skeptically prove that for all the time points in an interval a given fluent literal holds, it is sufficient to prove that the fluent literal skeptically holds in all the deserts which intersect the interval, and hence in all the oases of such deserts, as stated by the following proposition.

Proposition 11.7. *Given the theory KB_{TR}, a fluent literal $fl[t]$, the relative time line $TL = [t_0, ..., t_{n+1}]$, and a ground time interval $[a, b]$, it holds*

$$\forall\ t\ \textit{ground}\ t \in [a,b]\ KBtr\ \models_{TR}^{skep}\ \ fl[t]$$
$$\Leftrightarrow$$
$$\forall\ i\ [t_i + 1, t_{i+1}] \cap [a,b] \neq \emptyset\ \ KBtr\ \models_{TR}^{skep}\ \ fl[t_i + 1].$$

Propositions 11.6 and 11.7 directly provide a computational model for the extended theory, as formalised by the following programs.

$query\_ngq\_skeptically\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, exists([a,b], fl[t]), A)\ \leftarrow$
$\quad extract\_oases(KB_0, O),$
$\quad instantiate\_constraints(I_{TR}, O, OI_{TR}),$
$\quad get\_intersecting\_oases([a,b], O, List\_of\_oases),$
$\quad member(Ti, List\_of\_oases),$
$\quad query\_skeptically\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[Ti+1], A).$

$query\_ngq\_skeptically\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, forall([a,b], fl[t]), A)\ \leftarrow$
$\quad extract\_oases(KB_0, O),$
$\quad instantiate\_constraints(I_{TR}, O, OI_{TR}),$
$\quad get\_intersecting\_oases([a,b], O, List\_of\_oases),$
$\quad foreach\_member\_prove\_skeptically\_TR(List\_of\_oases, \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A).$

$query\_ngq\_skeptically\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A)\ \leftarrow$
$\quad query\_skeptically\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A).$

$foreach\_member\_prove\_skeptically\_TR([], \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A).$

$foreach\_member\_prove\_skeptically\_TR([Ti|Rest\_Tis], \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], [A1, A2]) \leftarrow$
$query\_skeptically\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[Ti+1], A1),$
$foreach\_member\_prove\_skeptically\_TR(Rest\_Tis, \langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[t], A2).$

$get\_intersecting\_oases([a, b], O, List\_of\_oases) \leftarrow$
Works as expected (as above for the case of $\vdash_{TR}^{cred^{ngq}}$).

Definition 11.5 ($\vdash_{TR}^{skep^{ngq}}$). *Given the abductive logic program KB_{TR}, which includes the dynamic knowledge KB_0 and implements the event calculus based temporal reasoning capability, as described in Section 11.4.1, the* computational model *for $\models_{TR}^{skep^{ngq}}$, indicated as $\vdash_{TR}^{skep^{ngq}}$, is defined as follows:*

$$
\begin{array}{rrcl}
KB_{TR} \vdash_{TR}^{skep^{ngq}} & \exists\ T \in [a,b]\ fl[T] & \Leftrightarrow & query\_ng\_skeptically\_TR(KB_{TR}, KB_0, \exists\ T \in [a,b]\ fl[T], A) \\
KB_{TR} \vdash_{TR}^{skep^{ngq}} & \forall\ T \in [a,b]\ fl[T] & \Leftrightarrow & query\_ng\_skeptically\_TR(KB_{TR}, KB_0, \forall\ T \in [a,b]\ fl[T], A) \\
KB_{TR} \vdash_{TR}^{skep^{ngq}} & fl[t] & \Leftrightarrow & query\_ng\_skeptically\_TR(KB_{TR}, KB_0, fl[t], A)
\end{array}
$$

Theorem 11.6 ($\models_{TR}^{skep^{ngq}} \Leftrightarrow \vdash_{TR}^{skep^{ngq}}$). *Assuming that the abductive proof procedure used in the definition of the predicate query_credulous_TR/4 is correct and complete, the computational model $\vdash_{TR}^{cred^{ngq}}$ is correct and complete with respect to the formal model $\models_{TR}^{skep^{ngq}}$:*

$$
\begin{array}{rrcl}
KB_{TR} \models_{TR}^{skep^{ngq}} & \exists\ T \in [a,b]\ fl[T] & \Leftrightarrow & KB_{TR} \vdash_{TR}^{skep^{ngq}} \exists\ T \in [a,b]\ fl[T] \\
KB_{TR} \models_{TR}^{skep^{ngq}} & \forall\ T \in [a,b]\ fl[T] & \Leftrightarrow & KB_{TR} \vdash_{TR}^{skep^{ngq}} \forall\ T \in [a,b]\ fl[T] \\
KB_{TR} \models_{TR}^{skep^{ngq}} & fl[t] & \Leftrightarrow & KB_{TR} \vdash_{TR}^{skep^{ngq}} fl[t]
\end{array}
$$

11.4.4 Use of TR in the overall computational model for the computee

The theory presented up to now supports the use of TR within the overall computational model for the computee developed in this deliverable. The use of TR consists hence of a mix of the presented proof procedures, according to the needs of the other components of the model that use it. Here we define the overall $\vdash_{TR}$, and we link its definition to the use of C-IFF as underlying proof procedure. TR is used

- by the *Goal Decision* capability, for which TR is part of the background logic. This component of the overall computational model calls TR with a ground query, exploiting the $\vdash_{TR}^{skep}$ computational instance of TR, defined in Definition 11.3.
- by the *Cycle theory*, in the enabling conditions (which contain the core selections functions) and the behaviour conditions (which contain the heuristic selections functions), and by the *Selection Functions* and the *Goal Revision* transition, which call TR with an existentially quantified query, and a set of temporal constraints. The query is dealt with by the $\vdash_{TR}^{skep^{ngq}}$ computational instance of TR, defined in Definition 11.5,[22] exploiting the C-IFF proof procedure for dealing with the temporal constraints.

[22] Actually, the definition of $\vdash_{TR}^{skep^{ngq}}$ includes that of $\vdash_{TR}^{skep}$, but for sake of clarity, we have preferred to separate the two cases.

Note that both the uses of TR refers to the skeptical case.

The call to the generic underlying proof procedure has been defined in Section 11.4.2 as

$$\langle P_{TR}, A_{TR}, I_{TR}(t_0+1) \wedge \cdots \wedge I_{TR}(t_n+1)\rangle \vdash_{ALP} fl[t].$$

Using the C-IFF proof procedure (Section 10.2), it becomes

$$\langle P_{TR}, A_{TR}, I_{TR}(t_0+1) \wedge \cdots \wedge I_{TR}(t_n+1)\rangle, fl[t] \vdash_{\text{ciff}} Answer,$$

with $Answer$ equal to (Δ, Γ), $failure$ or $flounder$ in case of a successful, failure or floundering derivation, respectively. Note that, moreover, the C-IFF may also not terminate.

Correspondingly, the predicate $query\_credulously\_TR(KB_{TR}, KB_0, fl[t], Answer)$, which defines $\vdash_{TR}^{cred}$ (Definition 11.2), returns the answer $Answer$ computed by C-IFF, as above described. In this section we refer with $\vdash_{TR}^{cred}$ to the instance of computational model based on C-IFF as now described.

The predicate $query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], A)$, given in Definition 11.3 as

$query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], A) \leftarrow$
$\quad query\_credulously\_TR(KB_{TR}, KB_0, fl[t], A),$
$\quad negate\_fluent(fl[t], Nfl[t]),$
$\quad$ not $query\_credulously\_TR(KB_{TR}, KB_0, Nfl[t], \_),$

was used to define $\vdash_{TR}^{skep}$ under the assumption that the underlying proof procedure was correct and complete. It must hence be reconsidered in the C-IFF settings, where only soundness holds in general. Interpreting *not* $query\_credulously\_TR(KB_{TR}, KB_0, Nfl[t], \_)$ as finite failure, the previous definition can be recast in

$query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], (\Delta, \Gamma)) \leftarrow$
$\quad query\_credulously\_TR(KB_{TR}, KB_0, fl[t], (\Delta, \Gamma)),$
$\quad negate\_fluent(fl[t], Nfl[t]),$
$\quad query\_credulously\_TR(KB_{TR}, KB_0, Nfl[t], fail).$

$query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], fail) \leftarrow$
$\quad query\_credulously\_TR(KB_{TR}, KB_0, fl[t], fail).$

$query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], fail) \leftarrow$
$\quad query\_credulously\_TR(KB_{TR}, KB_0, fl[t], (\Delta, \Gamma)),$
$\quad negate\_fluent(fl[t], Nfl[t]),$
$\quad query\_credulously\_TR(KB_{TR}, KB_0, Nfl[t], (\Delta', \Gamma')).$

Under this new definition, the query for $fl[t]$ skeptically succeeds returning an answer (Δ, Γ), when the query for $fl[t]$ credulously succeeds and the query for $Nfl[t]$ credulously finitely fails. On the other hand, it skeptically fails when either the query for $fl[t]$ credulously finitely fails, or both the queries for $fl[t]$ and $Nfl[t]$ credulously succeed. In this section we refer with $\vdash_{TR}^{skep}$ to the instance of the computational model based on C-IFF as now described.

Now, $\vdash_{TR}$ can be specified with respect to the use of C-IFF as the underlying proof procedure.

Definition 11.6 ($\vdash_{TR}$). *Let KB_{TR} be the abductive logic program that implements the event calculus based temporal reasoning capability (including the dynamic knowledge KB_0), as described in Section 11.4.1. The* computational model *for $\models_{TR}$, indicated as $\vdash_{TR}$, is defined as follows:*

1. *Let $fl[t]$ be a ground fluent and Answer the answer returned by the above redefined predicate*

$$query\_skeptically\_TR(KB_{TR}, KB_0, fl[t], Answer),$$

which now defines $\vdash_{TR}^{skep}$. Then

$$KB_{TR} \vdash_{TR} fl[t] \quad \textit{iff} \quad KB_{TR} \vdash_{TR}^{skep} fl[t], \textit{ with } Answer = (\Delta, \Gamma)$$

$$KB_{TR} \vdash_{TR} \textit{(finitely) fails to prove } fl[t] \quad \textit{iff} \quad KB_{TR} \vdash_{TR}^{skep} fl[t] \textit{ with } Answer = fail$$

2. *Let $fl[T]$ be a fluent containing the existentially quantified variable T, and $TCS(T)$ a conjunction of temporal constraints over T (which is, at least, constrained in an interval $[a, b]$) and possibly over other existentially quantified variables. Let Answer be the answer returned by the predicate, defined in Definition 11.5,*

$$query\_ngq\_skeptically\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, exists(TCS(T)^{23}, fl[t]), Answer),$$

which now defines $\vdash_{TR}^{skep^{ngq}}$. Then

$$KB_{TR} \vdash_{TR} fl[T] \wedge TCS(T) \quad \textit{iff} \quad KB_{TR} \vdash_{TR}^{skep^{ngq}} exists(TCS(T)^{24}, fl[t]) \textit{ with } Answer = (\Delta, \Gamma)$$

$$KB_{TR} \vdash_{TR} \textit{(finitely) fails to prove } fl[T] \wedge TCS(T) \quad \textit{iff} \quad KB_{TR} \vdash_{TR}^{skep^{ngq}} exists(TCS(T), fl[t]) \textit{ with } Answer = fail$$

According to this definition, $\vdash_{TR}$ returns an answer only when the query is successfully proved by means of C-IFF, or in case of the finite failure of C-IFF, given the query. The success and failure soundness results for C-IFF given in Section 10.1 allow the soundness of $\vdash_{TR}$ to be proved for the relative success and finite failure.

Theorem 11.7 ($\vdash_{TR}$ soundness). *Let KB_{TR} be the abductive logic program that implements the event calculus based temporal reasoning capability, as described in Section 11.4.1 (including the dynamic knowledge KB_0). Let $fl[t], fl[T]$ and $TCS(T)$ be a ground fluent literal, a fluent literal and a set of temporal constraints, respectively. Then*

[24] Slightly abusing the notation, the constraint $T \in [a, b]$ has been replaced by a set of constraints $TCS(T)$, which is dealt with by means of the C-IFF proof procedure. This is obtained by adding the other constraints C, possibly appearing in $TCS(T)$, as a conjunct to the query $fl[T_i + 1]$ in the predicate $query\_skeptically\_TR(\langle P_{TR}, A_{TR}, OI_{TR}\rangle, KB_0, fl[T_i + 1] \wedge C, Answer)$. This predicate, occurring in the definition of $query\_ngq\_skeptically\_TR/4$ for the existential case (Definition 11.5), will ultimately pass the so-extended query to the C-IFF, and the constraints C will be taken into consideration when resolving the query. The same strategy has been implemented in Prototype Demonstrator developed within WP4 (see Deliverable D9, [5]).

$$
\begin{array}{rcl}
KB_{TR} \vdash_{TR} \ fl[t] & \Rightarrow & KB_{TR} \models_{TR} \ fl[t] \\
KB_{TR} \vdash_{TR} \ \textit{(finitely) fails to prove } fl[t] & \Rightarrow & KB_{TR} \not\models_{TR} \ fl[t] \\
KB_{TR} \vdash_{TR} \ fl[T] \wedge TCS(T) & \Rightarrow & \exists\, \sigma.\ KB_{TR} \models_{TR} fl[T]\sigma \ \wedge \ \sigma \models_{\Re} TCS(T) \\
KB_{TR} \vdash_{TR} \ \textit{(finitely) fails to prove } fl[T] \wedge TCS(T) & \Rightarrow & \nexists\, \sigma.\ KB_{TR} \models_{TR} fl[T]\sigma \ \wedge \ \sigma \models_{\Re} TCS(T)
\end{array}
$$

(where $\models_{TR}$, according to its definition, is based on the semantical entailment of the underling ALP proof procedure, i.e. $\models^3_{\Re}$ for the case of C-IFF).

Remark I (Correctness): We are currently working at the conjecture that the TR theory can guarantee (or could guarantee under weak syntactic restrictions) the completeness of the underlying ALP proof procedure in general. In this respect, we are exploring the definition of a suitable level mapping induced by time on the TR theory, combined with the approach that makes integrity constraints ground, with the aim of avoiding floundering and non-termination. In this case, we expect to extend the proved completeness results to the case in which C-IFF is used as underlying proof procedure. Similar conditions have been studied in [122]. This point is currently under further investigation.

Remark II (Extensions): After having set the computational counterpart of TR that supports the definition of the overall computee computational model, other open issues are worth being investigated. As explained above we have started our study with a set of simplifying assumptions that allowed us to define a simple computational model for ground queries and ground narratives. Then we have relaxed the hypothesis of groundness for the query and obtained the current computational model. We have further extended the model by allowing *actions happening within intervals* and by means of them, *theories with inconsistent narratives* as required by the *KGP* model of computees proposed in [63]. For simplicity of presentation, since these further extensions are not exploited in the current version of the computational model of the computee, they are presented in Appendix A, and intended for future extensions.

11.4.5 Examples of $\vdash_{TR}$

In this section we outline how the Temporal Reasoning capability computes some of the examples shown above. Other examples, illustrating the extensions of Temporal Reasoning presented in Appendix A, can be found in the same Appendix.

Example 5 (Ground narrative, ground queries). *In Example 1, we had the following domain dependent knowledge, partially in the narrative in KB_0, partially in the domain dependent part of KB_{TR}:*

$$
\begin{array}{c}
executed(switch\_on, 10). \\
observed(neg(light), 20).
\end{array}
$$

$$
initiates(switch\_on, T, light) \ \leftarrow \ holds\_at(neg(broken\_bulb)), T).
$$

and the query $holds\_at(broken\_bulb, 30)$*. According to the basic computational model introduced in Section 11.4.2, integrity constraints are grounded in the set*

$$\begin{array}{ll}\{ & holds\_at(F,1), holds\_at(neg(F),1) \Rightarrow false. \\ & holds\_at(F,11), holds\_at(neg(F),11) \Rightarrow false. \\ & holds\_at(F,31), holds\_at(neg(F),31) \Rightarrow false. \quad \}\end{array}$$

since $[0, 10, 30]$ *is the relative time line. It easy to verify that the constraint*

$$holds\_at(F,31), holds\_at(neg(F),31) \Rightarrow false$$

is violated when we abduce $assume\_holds(neg(broken\_bulb), 0)$*, since it would cause* $holds\_at(light, 31)$ *against the fact that* $holds\_at(neg(light), 31)$ *which can be proved because of the observation* $observed(neg(light), 20)$*. None of the constraints is instead violated by abducing* $assume\_holds(broken\_bulb, 0)$*. Hence, the credulous query*

$$\begin{array}{l}query\_credulous\_TR(\, KB_{TR}, \\ \qquad\qquad [executed(switch\_on, 10).observed(neg(light), 20).], \\ \qquad\qquad holds\_at(broken\_bulb, 30), \\ \qquad\qquad Answer).\end{array}$$

with KB_{TR} *equal to the union of the domain dependent and independent theory, succeeds by successfully calling, after the necessary transformation of the theory, the underlying ALP proof procedure.*

Similarly, there is no abducible set which can be used in credulously proving $holds\_at(neg(broken\_bulb, 30))$ *(because of the same constraint), hence the skeptical query for* $holds\_at(broken\_bulb, 30)$ *also holds.*

Example 6 (Ground narrative, existentially quantified queries). *In Example 2, with the slightly more complex narrative*

$$\begin{array}{c}executed(switch\_on, 10). \\ observed(neg(light), 20). \\ observed(booked\_room, 40).\end{array}$$

Temporal Reasoning was required to check whether $holds\_at(broken\_bulb), T)$ *holds with* $30 < T$, $T < 330$ *. Therefore we are interested in the query* $exists([30, 330],\ holds\_at(broken\_bulb, T))$*. According to the computational model defined in Section 11.4.3 the relative time line is* $[0, 10, 30, 330]$*, and one of the possible ways to make the query ground, is to assign to* T *the time value* 11*:*

$$holds\_at(broken\_bulb, 11),$$

i.e. an oasis which intersects the interval of the query (i.e. $[30, 330]$*). It is straightforward to check, after having appropriately grounded the integrity constraints, that the now ground, query can be proved skeptically. Since the query is existentially quantified, the success in a single grounding point implies that the original query holds (there are also other possibilities, for instance to ground the query in the point* 31*).*

Example 7 (Ground narrative, universally quantified query). *Consider again the narrative and the domain dependent theory of Example 1:*

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$

and the universally quantified query $forall([0, 50],\ holds\_at(broken\_bulb, T))$. *The time line is* $[0, 10, 30, 50]$, *and the intersecting point of interest with the interval of the query are* $[1, 11, 31, 51]$. *Analogously with what done in the previous example, it is easy to verify that all the ground queries*

$$holds\_at(broken\_bulb), 1),$$
$$holds\_at(broken\_bulb), 11),$$
$$holds\_at(broken\_bulb), 31),$$
$$holds\_at(broken\_bulb), 51),$$

are skeptically entailed by the theory (where the integrity constraints have been grounded in the usual way). Indeed, again, the assumption $assume\_holds(broken\_bulb, 0)$, *that is a valid explanation for* $holds\_at(broken\_bulb, T))$, *where* T *is any oasis point, is compatible with the subsequent observation* $observed(neg(light), 20)$, *while the assumption* $assume\_holds(neg(broken\_bulb), 0)$, *which would explain* $holds\_at(neg(broken\_bulb), T))$, *is not, for instance, because it violates the constraints*

$$holds\_at(broken\_bulb, 31), holds\_at(neg(broken\_bulb), 31) \Rightarrow false.$$

It follows that the the original query $forall([0, 50],\ holds\_at(broken\_bulb, T))$ *skeptically holds.*

11.4.6 Properties of $\vdash_{TR}$ with respect to $\models_{TR}$ (Summary)

We summarize here the various properties of the computational model for Temporal Reasoning that we have shown above and in Appendix A in the gradual development of this model.

We have shown how the computational model for Temporal Reasoning can be defined over a (Constraint) ALP proof procedure by means of simple program transformations of the given temporal reasoning theory that operate only on the user-defined part of the theory. For both the credulous and the skeptical case, we have defined a computational model that can be based on any ALP proof procedure, taking in our case the C-IFF procedure, $\vdash_{TR}^{cred}$ and $\vdash_{TR}^{cred}$ respectively. We sketched how the relative soundness and completeness with respect to the formal model can be proved building on the soundness and completeness of the ALP proof procedure adopted (Theorem 11.3 and Theorem 11.4).

We have extended this core computational model to queries which may have existentially or universally quantified variables, with derivability relations $\vdash_{TR}^{cred^{ngq}}$ and $\vdash_{TR}^{skep^{ngq}}$ respectively, and sketched again the relative soundness and completeness proofs (Theorem 11.5 and Theorem 11.6).

A third extension deals with the possibility of having non-ground narratives, encompassing actions with existentially quantified time variables, and this is presented in Section A.1. The extended computational model both for credulous $\vdash_{TRint}^{cred}$ and skeptical $\vdash_{TRint}^{skep}$ reasoning is shown to be sound in Theorem A.1 and complete in Theorem A.2, in the same sense as above.

We have also studied in Section A.2 how the two previous extensions can be combined in a simple way to to deal with quantified queries in domains with quantified actions.

Finally, in Section A.3 we have analyzed the problem of how to deal with theories that are (frame) inconsistent using the ability to reason with quantified actions to generate explanations with unknown events that can resolve the inconsistency.

The various cases of the computational model developed support the functioning of the computee for its Temporal Reasoning, and a subset of them have been implemented and integrated in the Prototype Demonstrator (see Deliverable D9 [5]).

11.5 Goal Decision

The computee Goal Decision capability selects, at a given instant, the top level goals to be pursued. These goals are preferred by the computee at the time of their selection, but this may change over time. Some of these goals may be direct reactions to new information that arrives at the computee and as such these goals may simply be realizable by a single action.

This section proposes a simple computational model for $\models^{\tau}_{GD}$ that is based directly on the general computational model of the preference relation $\models_{pr}$ of $LPwNF$ as described in 10.3. Formal properties of the computational model are stated (under some assumptions of simplification) and sketch proofs are provided.

11.5.1 KB_{GD} and specification of $\models_{GD}$: recap

The knowledge from which Goal Decision draws its conclusions is a $LPwNF$ theory, called KB_{GD}, which contains the goal preference policy of the computee. With KB_{GD} it uses also in its background logic as auxiliary information the knowledge of $KB_{TR} \cup KB_0$.

The knowledge base KB_{GD} contains three main parts: the *auxiliary part* with rules defining auxiliary predicates, the *lower-level part* with rules to generate goals and the *higher-level part* with rules that specify priorities between other rules of the theory. We assume that a subset of the fluents in the language of the knowledge base of the computee is separated out as the set of *goal fluents* of the computee.

The lower-level part of KB_{GD} consists of rules in $LPwNF$ of the form

$$g[t], Tg \leftarrow L_1, \ldots, L_n, T_C \quad (n \geq 0)$$

where

- $L_1, \ldots, L_n$ are either time-dependent conditions of the form $holds\_at(l, t)$, where $l[t]$ is a timed fluent literal, or time-independent conditions, formulated in terms of auxiliary predicates defined within KB_{GD}. T_C are temporal constraints on the time variables in the body of the rule.

- g is a goal fluent, Tg is a (possibly empty) set of temporal constraints and the time variable t is existentially quantified with scope the conclusion of the rule.

All variables except t in the rules are implicitly universally quantified from the outside and a rule then represents all its ground instances in the Herbrand universe of the program and under any total valuation of the time variables in the body that satisfies T_C. The time variable t in $g[t], Tg$ not occurring in the body of the rule (if it exits) is implicitly existentially quantified with scope the conclusion of the implication.

The auxiliary part, KB^{aux}_{GD}, of KB_{GD} is simply a set of $LPwNF$ rules defining, via the background logic $\models_H$, any auxiliary predicates occurring in the remaining part of KB_{GD}. In particular, it contains statements of incompatibility such as:

$$incompatible(l_1, l_2)$$

expressing that the (possibly timed) goal fluent literals l_1 and l_2 are incompatible. In some cases this incompatibility could hold only under some conditions B (a conjunction of auxiliary literals), expressed by the rule

$$incompatible(l_1, l_2) \leftarrow B.$$

We will assume that this incompatibility relation is defined only for fluent literals at the same time.

The conditions of the rules are evaluated in KB_{GD} together with $KB_0 \cup KB_{TR}$ by combining the background derivability $\models_H$ of the $LPwNF$ framework with the Temporal Reasoning capability $\models_{TR}$. With abuse of notation we will denote in this subsection this combined derivability simply by $\models_H$ assuming that $KB_0 \cup KB_{TR}$ is contained in the auxiliary part of KB_{GD}.

The higher-level part of KB_{GD} consists of rules in $LPwNF$ of the form:

$$h\_p(rule1, rule2) \leftarrow L_1, \ldots, L_n, T_C$$

where $rule1, rule2$ are (parameterised) names of other rules in the knowledge base KB_{GD} (but not in the auxiliary part of KB_{GD}) and $L_1, \ldots, L_n, T_C$ are as above for the rules in the lower-level part. These rules in the higher-level part of KB_{GD} represent the (local) priorities amongst rules in the lower-level part or other priority rules in the higher-level part.

Rules in KB_{GD} may have in their body a special time variable, denoted by T_{now} that refers to the (current) time τ at which the capability of goal decision is applied by the computee. We will denote by $KB_{GD}(\tau)$ the knowledge base obtained from KB_{GD} by instantiating in this the time variable T_{now} with τ.

Specification of $\models^{\tau}_{GD}$

Formally, the capability of $\models^{\tau}_{GD}$ is defined, using the admissibility semantics for $LPwNF$ and its preference entailment relation, as given in [63], in the following way. Given a state $\langle KB, Goals, Plan\rangle$, and a time point τ,

$$KB \models^{\tau}_{GD} \mathcal{G}s$$

iff $\mathcal{G}s$ is a maximal set, $\mathcal{G}s = \{\langle g_1[t_1], Tg_1\rangle, \ldots, \langle g_n[t_n], Tg_n\rangle\}$, $n \geq 0$, where g_i are goal fluent literals and Tg_i are temporal constraints on t_i, such that:

- $KB_{GD}(\tau) \models_{pr} (g_1[t_1], Tg_1), ..., g_n[t_n], Tg_n)$ for $i = 1, \ldots, n$.

This means that a new (possibly empty) set of goals $\mathcal{G}s$ is generated that is currently (sceptically) preferred under the policy in KB_{GD} and the current information in KB_0 as used by the Temporal Reasoning capability to evaluate conditions in these policy rules.

More explicitly, following the definition of the sceptical preference entailment $\models_{pr}$ of $LPwNF$, the above definition of $KB \models^{\tau}_{GD} \mathcal{G}s$ holds iff $\mathcal{G}s$ is a maximal set of goals such that there exists a subset Δ of rules in $KB_{GD}(\tau)$ such that for each $G = \langle g, T_g \rangle \in \mathcal{G}s$

1. Δ contains an instance $R\sigma$ of a rule R in $KB_{GD}(\tau)$ of the form $L, T_L \leftarrow B, T_C$ such that σ is a valuation of the temporal constraints T_C, $L\sigma = g$, $T_L\sigma = T_g$ and $KB \models_H B\sigma$,[25]

2. Δ is admissible in $KB_{GD}(\tau)$

3. no such Δ satisfying the first two conditions can be constructed for any goal $\overline{G}$ that is incompatible with G.

Note that any two goals in $\mathcal{G}s$ are necessarily compatible with each other. Note also that $\models^{\tau}_{GD}$ may return an empty set of goals when there are no sceptically preferred goals at the time τ of application of this capability.

11.5.2 $\vdash^{\tau}_{GD}$: Computing the Goal Decision capability

As seen from the specification of Goal Decision this relies directly on the underlying preference reasoning within the $LPwNF$ framework. It simply uses this form of reasoning with its specific theory of KB_{GD}. The theories $KB_{TR} \cup KB_0$ are used as auxiliary background theories for the computation of conditions in the rules of $KB_{GD}(\tau)$.

Hence the derivability relation $\vdash^{\tau}_{GD}$ and the computational model for Goal Decision can be drawn directly from the general derivability relations $\vdash^{cred}_{pref}$ and $\vdash_{pref}$ of the $LPwNF$ framework as presented in 10.3. A simple but relatively inefficient way to compute $\vdash^{\tau}_{GD}$ would then be to generate one by one sceptical goals, via $\vdash_{pref}$, adding the most recently generated goal to the previous goals and re-checking, again via $\vdash_{pref}$, that the whole set remains a sceptical conclusion.

A more efficient algorithm for computing $\vdash^{\tau}_{GD}$ that exploits some of the special features of the goal decision knowledge base KB_{GD} is as follows.

Let $\mathcal{T}$ denote the theory $KB_{GD}(\tau) \cup KB_{TR} \cup KB_0$ at the given current time τ. Then $\mathcal{T} \vdash^{\tau}_{GD} Gs$ iff Gs is given by:

- generate the set Gs' of all atomic goals G such that $\mathcal{T} \vdash^{cred}_{pref} G$

- Gs is a maximal set such that $Gs = \{G' \in Gs' \mid \nexists G'' \in Gs' : incompatible(G', G'')\}$

Hence this algorithm relies only on the credulous derivability relation of $LPwNF$ to *generate* in the first step a set of candidate goals and then, using checks of incompatibility, it *filters* from this the required goals. The soundness (and completeness) of this algorithm will rest on the assumption that the set Gs' is finite and will follow from the special form of the goal decision knowledge base KB_{GD}.

[25] We remind the reader that here in $KB \models_H B\sigma$ the auxiliary part KB^{aux}_{GD} of KB_{GD} is used alongside $KB_{TR} \cup KB_0$ and $\models_{TR}$ to evaluate the conditions $B\sigma$ of the rules.

11.5.3 Example of $\vdash^{\tau}_{GD}$

Let us illustrate the goal decision capability and the main steps of computation of $\vdash^{\tau}_{GD}$ by considering the example of a KB_{GD} theory as given earlier in section 10.3. We reproduce part of the example here to help its readability.

Consider the following part of the goal decision knowledge base KB_{GD} of a computee describing its simple policy for deciding how to respond to requests. This has two object-level (or lower-level) generation rules one for responding "yes" and the other for responding "no":

$r_y(Asker, RequestedNeed, yes)$:
$respond(Myself, Asker, need(RequestedNeed), yes, ResponseTime), C(ResponseTime) \leftarrow$
$holds\_at(request(Asker, Myself, need(RequestedNeed), RequestTime), T_{now})$
$currently\_satisfiable(Myself, RequestedNeed)$,
$response\_time(RequestTime, T_{now}, ResponseTime)$.

$r_n(Asker, RequestedNeed, no)$:
$respond(Myself, Asker, need(RequestedNeed), no, ResponseTime), C(ResponseTime) \leftarrow$
$holds\_at(request(Asker, Myself, need(RequestedNeed), RequestTime), T_{now})$
$incompatible\_current\_needs(Myself, RequestedNeed)$,
$response\_time(RequestTime, T_{now}, ResponseTime)$.

where *respond* is the only goal fluent. All other predicates are auxiliary and their definition is not important for the purposes of the example. Here the $ResponseTime$ in both rules can be either ground or an existentially quantified time with some constraints, $C(ResponseTime)$, to respond within a certain time from the request time. Note that we also have the incompatibility statement between a yes and a no reply for the same request:

$incompatible(respond(M, A, need(RN), yes, RT), respond(M, A, need(RN), no, RT))$.

The knowledge base also contains a higher-level part with priorities as follows:

$p_1(A, RN) : h\_p(r_n(A, RN, no), r_y(A, RN, yes)) \leftarrow urgent(Myself, RN)$.
$p_2(A, RN) : h\_p(r_y(A, RN, yes), r_n(A, RN, no)) \leftarrow urgent(A, RN)$.
$p_3(A, RN) : h\_p(r_y(A, RN, yes), r_n(A, RN, no)) \leftarrow \neg urgent(Myself, RN)$.

expressing the policy to "prefer to refuse requests for needs that you urgently need and prefer to accept requests for needs that you do not need urgently or that are urgently needed by the asker". When the requested need is urgent for both the asker and yourself then the preference to refuse is stronger unless the asker is your manager. This is captured with the higher-order priorities:

$c_1(A, RN) : h\_p(p_1(A, RN), p_2(A, RN))$.
$c_2(A, RN) : h\_p(p_2(A, RN), p_1(A, RN)) \leftarrow manager(Asker, Myself)$
$d_1(A, RN) : h\_p(c_2(A, RN), c_1(A, RN))$.

Now suppose that a computee receives a request for some need, *req_need*, for which both $currently\_satisfiable$ and $incompatible\_current\_needs$ hold true. Given this information alone

$\vdash^{\tau}_{GD}$ will return the empty set as the preferred current goals. This is computed as follows:

- $\vdash^{cred}_{pref}$ will first produce as the current credulous set of goals:

  - $\langle respond(myself, asker, need(req\_need), yes, t_1), C(t_1)\rangle$,
  - $\langle respond(myself, asker, need(req\_need), no, t_2), C(t_2))\rangle$

  where $C(t_1)$ and $C(t_2)$ are (the same) temporal constraints on the response time set by the *response_time* condition in the above rules.

- as these goals are incompatible with each other they will both be deleted by the second step of filtering of the credulous goals produced in the first step. Hence, $\vdash^{\tau}_{GD} \{\}$.

Suppose instead that the computee also knows that $urgent(myself, req\_need)$ holds then the first step of generation of $\vdash^{\tau}_{GD}$ will produce only the second goal of refusal shown above as the first goal of acceptance is not generated now by $\vdash^{cred}_{pref}$. Hence now we have that:

- $\vdash^{\tau}_{GD} \{< respond(myself, asker, need(req\_need), no, t_2), C(t_2)) >\}$

Finally, let us consider the case where the requested need is known by the computee to be urgent for both itself and the asker. In this case the first step of generation of $\vdash^{\tau}_{GD}$ will generate both goals as with the starting scenario of this example. In this case though these goals are supported by higher priority rules, namely the first one by the rule $p_1(asker, new\_need)$ and the second one by the rule $p_2(asker, new\_need)$. The second step of $\vdash^{\tau}_{GD}$ will again filter out both these goals and so again $\vdash^{\tau}_{GD}$ will return the empty set of goals. The computee is again in a dilemma.

Note that the difference of this case with the starting scenario is the fact that the generated goals are necessarily supported by the priority rules, mentioned just above, which in fact are incompatible with each other (they have negative conclusions). Hence if the two generated goals themselves were compatible with each other then the algorithm for $\vdash^{\tau}_{GD}$ will return them both giving an unsound answer. We will see below in the next subsection where we study the properties of this algorithm how it can be extended to overcome this. We will see that the filtering second step can be extended to look for incompatibility not only of the top level goals generated but also for incompatibility between the admissible sets (of priority rules) that support these goals.

11.5.4 Properties of $\vdash^{\tau}_{GD}$ with respect to $\models_{GD}$

Let us assume that the knowledge base, $KB_{GD}(\tau)$, for any given current time, τ, together with its auxiliary use of $KB_{TR} \cup KB_0$ is such that only a finite number of goals G can be entailed from $\mathcal{T} = KB_{GD}(\tau) \cup KB_{TR} \cup KB_0$ via the background logic $\models_H$ of $LPwNF$. This is guaranteed for example if the rules in $KB_{GD}(\tau)$ when grounded produce a finite number of ground rules. We will call this assumption the *goal finiteness* assumption.

We can then show the following result.

Theorem 11.8. *Let $\mathcal{T}$ denote the theory $KB_{GD}(\tau) \cup KB_{TR} \cup KB_0$ and suppose that a sound and complete derivation $\vdash_H$ of $\models_H$ is used within $\vdash^{cred}_{pref}$. If $\mathcal{T} \vdash^{\tau}_{GD} Gs$ then, Gs is a maximal set such that for each $G_i \in Gs$, $\mathcal{T} \models^{scept}_{pref} G_i$.*

To satisfy the specification of $\models_{GD}$ fully we need to show that the conjunction of the goals in Gs, generated by $\mathcal{T} \vdash^{\tau}_{GD} Gs$, is also a skeptical conclusion of the theory $\mathcal{T}$. To do this we exploit the property of the theory $\mathcal{T}$ that incompatibility can only arise at the level of the goal fluents together with the assumptions that:

- the auxiliary part of $\mathcal{T}$, namely $KB^{aux}_{GD}(\tau) \cup KB_{TR} \cup KB_0$, is consistent.
- there is only one maximally admissible subset of the higher-level part of $KB_{GD}(\tau)$, i.e. for any pair of (ground) rules R_1 and R_2 in the higher-level part of $KB_{GD}(\tau)$ that are inconsistent with each other and whose conditions currently hold there exists a priority rule C_1 (whose conditions also currently hold) that assigns to one of R_1 and R_2 higher priority [26].

Given these additional assumptions which we will call the *deterministic* assumptions we can easily show as a corollary of the above theorem the soundness and completeness of $\vdash^{\tau}_{GD}$.

Corollary 11.1. *Let $\mathcal{T}$ denote the theory $KB_{GD}(\tau) \cup KB_{TR} \cup KB_0$ and suppose that $\mathcal{T}$ has the goal finiteness property and satisfies the above deterministic assumptions. Then $\mathcal{T} \vdash^{\tau}_{GD} Gs$ iff $\mathcal{T} \models_{GD} Gs$.*

While the first condition of the above deterministic assumption is reasonable the second condition can be restrictive in some cases especially within an open environment and so we may want to relax it or delete it completely.

In such a case we have two possibilities. The first one is to use directly a sceptical derivation, denoted by $\vdash^{scep-all}_{pref} G$, that extends $\vdash^{scep}_{pref} G$, by checking not only that it is not possible to derive credulously (via $\vdash^{cred}_{pref}$) any goal incompatible to the top-level goal G but also checking the same is true for any other subsidiary (non-auxiliary) goal that is part of the admissible set Δ computed for G by $\vdash^{scep-all}_{pref}$. We can then collect together in Gs all the goals G computed by $\vdash^{scep-all}_{pref}$ in this way. This set will satisfy the specification of $\models_{GD}$.

Although such an extension would be necessary for general theories in $LPwNF$ for the special case of KB_{GD} we have an alternative way which would be more effective as this can exploit the special features of such theories. We can extend the above algorithm for computing goal decision, which first computes all the goals that credulously follow from the theory, in the following way.

Let $\mathcal{T}$ denote the theory $KB_{GD}(\tau) \cup KB_{TR} \cup KB_0$ at the given current time τ. Then $\mathcal{T} \vdash^{\tau}_{GD} Gs$ iff Gs is given by:

- generate the set Gs' of all atomic goals G and admissible sets Δ computed for G via $\mathcal{T} \vdash^{cred}_{pref} G$. Each member of Gs' is a tuple $\langle G, S_G \rangle$, where S_G is the set of all the admissible Δ computed for G.
- Gs is a maximal set such that $Gs = \{G' \mid \langle G', S_{G'} \rangle \in Gs'$ and $\nexists \langle G'', S_{G''} \rangle \in Gs'$ s.t. $incompatible(G', G'')$ and there exists $\Delta' \in S_{G'}$ s.t. for any $\langle G'', S_{G''} \rangle \in Gs'$ $\nexists \Delta'' \in S_{G''}$ s.t. $incompatible(\Delta', \Delta'')$ i.e. for any $\delta' \in \Delta'$ and $\delta'' \in \Delta''$ $incompatible(\delta', \delta'')$ does not hold $\}$

[26] Note if there exists also a rule C_2 (whose conditions currently hold) that assigns higher priority the other way around then this condition implies that there will be a rule D_1 that will assign priority to one of C_1 and C_2 and so on until we reach at a finite level where only one such rule exists.

We can then show that this computes correctly the goal decision capability.

Theorem 11.9. *Let $\mathcal{T}$ denote the theory $KB_{GD}(\tau) \cup KB_{TR} \cup KB_0$ which has the goal finiteness property and $KB^{aux}_{GD}(\tau) \cup KB_{TR} \cup KB_0$ is consistent. Suppose also that a sound and complete derivation $\vdash_H$ of $\models_H$ is used within $\vdash^{cred}_{pref}$. If $\mathcal{T} \vdash^{\tau}_{GD} Gs$ then $\mathcal{T} \models_{GD} Gs$.*

Proof (Sketch): There exists a union of the admissible sets of the separate goals in Gs that is consistent (by construction of Gs) and therefore this is an admissible set that satisfies all the requirements of the specification of $\models_{GD}$. □

Another condition under which the first (non-extended) algorithm for $\vdash^{\tau}_{GD}$ will be sound and complete is to assume that in the higher-level part of $KB_{GD}(\tau)$ any priority rule amongst two rules of the lower part of $KB_{GD}(\tau)$ is such that these two lower-level rules have goal fluent conclusions that are incompatible with each other. This condition although relatively simple is not always appropriate as in several cases we may want part of the preference policy to apply between compatible goals. In this case we will need the extended algorithm for $\vdash^{\tau}_{GD}$.

12 Correctness of the computational model of selection functions and transitions

In section 9, we have provided computational models for selection functions and transitions, which are trivially correct given a correct constraint solver $\vdash_{\Re}$ for $\models_{\Re}$, a correct computational model for $\models_{LP}$ (for fluent selection), and correct computational models for $\models_{TR}$ and $\models^{\tau}_{plan}$. Having provided, in the earlier section 11, such computational counterparts for $\models_{TR}$ and $\models^{\tau}_{plan}$, and proven correctness results for them, we can now give the following trivial correctness result for the computational selection functions and transitions.

Theorem 12.1. *(Correctness of computational selection functions and transitions)*
Let the computational counterparts for $\models_{TR}$ and $\models^{\tau}_{plan}$ be the ones given in section 11. Moreover, let $\vdash_{\Re}$ be a constraint solver correct and complete wrt $\models_{\Re}$, Finally, let $\vdash_{LP}$ be a correct computational counterpart for $\models_{LP}$. Then

- *the computational selection functions $c^c_{AS}, c^c_{GS}, c^c_{FS}, c^c_{PS}$ given in section 9 are correct with respect to the corresponding selection functions given in D4;*
- *the computational models $\{GI^c, PI^c, RE^c, SI^c, POI^c, AOI^c, AE^c, GR^c, PR^c\}$ given in section 9 are correct with respect to the corresponding transitions given in D4.*

13 Correctness of the cycle computational model

In section 8, we have provided a notion of computational operational trace of a computee, responsible for its behaviour. In that section we have also provided a conditional correctness result for the computational operational trace (Theorem 8.4), parametric on correct computational models for $\models_{pr}$, selection functions and transitions. Having proven, in the the earlier section 10.3 a correctness result of $\vdash_{pr}$ (given by definition 10.14) wrt $\models_{pr}$, and, in the earlier section 12, correctness results for the computational counterparts of the transitions and selection functions given in section 9, we can now give the following corretness result for the computational operational trace, which trivially follows from the earlier results.

Theorem 13.1. *(Correctness of the computational operational trace)*
Let

- $\vdash_{pr}^{cred}$ *be the one given in definition 10.14 in section 10.3 for the general framework of* $LPwNF$*;*
- $\{\vdash_{GI}, \vdash_{PI}, \vdash_{RE}, \vdash_{SI}, \vdash_{POI}, \vdash_{AOI}, \vdash_{AE}, \vdash_{GR}, \vdash_{PR}\}$ *be the computational counterparts of the transitions* $\{GI, PI, RE, SI, POI, AOI, AE, GR, PR\}$*, given in section 9, and*
- $c^c_{AS}, c^c_{GS}, c^c_{FS}, c^c_{PS}$ *be the computational counterparts of the (core and heuristic) selection functions (for action, goal, fluent and precondition selection) given in section 9.*

Then, any computational operational trace, as in section 8.3, wrt some given cycle theory $\mathcal{T}_{cycle}$ *and initial state* S_0*, is an operational trace, as in section 8.2.2, wrt the same cycle theory and initial state.*

14 An example

In this section we give a simple example of the computational operational trace of a computee, which exemplifies how all the components of the computational model that we have provided work together, from operational trace, to transitions, to capabilities and proof procedures. In particular, we will show how the cycle theory decides the next transition and then how the state changes as the transition is executed. The example is adapted from [4].

For the purposes of this example we assume that the computee, here called Francisco (see [111]), which has inspired this example), has the normal cycle theory as described earlier in this section. Also the only other part of its static knowledge base that plays a role is its KB_{GD} and KB_{TR}. For the latter we assume that the computee has the standard theory KB_{TR} given in D4, that we will revise in section 11.4. Its KB_{GD} is given below (for syntax and semantics of goal decision see section 11.5).

Intuitively, in this example we examine how Francisco's computee can assist Francisco to leave San Vincenzo. To illustrate this we assume that Francisco's computee contains in its goal decision theory, KB_{GD}, a personality theory on needs and motivations of the kind discussed in [74] and re-interpreted in [121]. Following the re-interpretation proposed in [121] the computee has its possible goal decisions labelled (or separated) into five major categories based on the needs each goal decision contributes to fulfill: *operational*, self-benefit, peer-interaction, community-interaction, and non-utilitarian (see [121] for an explanation).

In our example, Francisco's computee, has to take into consideration, in addition to the goal (or need) of Francisco to leave San Vincenzo, the needs of Francisco as part of his peer-interaction motivations, for example, Francisco's everyday needs. These include context-independent and recurring needs, such as reading daily news, and context-dependent needs, such as leaving San Vincenzo at 14:00h. The computee also has to take into consideration its own needs, for example, its operational needs dictate to take appropriate action when Francisco's PDA is running out of battery power. Possible goal decisions contributing to the fulfillment of all these needs are encoded in Francisco's computee's knowledge base for goal decision, KB_{GD}.

In KB_{GD}, a low-battery alert from the PDA has higher priority over all other needs, such as aggregation of news feeds and context-dependent needs, like catching a train. Similarly, context-dependent needs are ranked higher than context-independent and recurring needs like reading of news. In other words, the context-dependent need of Francisco to catch the 14:00h

train to leave San Vincenzo has higher priority than aggregating news feeds on the day of the departure. However, a low-battery alert from the PDA is always preferred in order to avoid loosing all the PDA data.

- KB_{GD}^{low}: There are three possible goals: (a) Leaving San Vincenzo (lsv), (b) aggregation of news feeds (nfa), and (c) low-battery alert (lba). The knowledge base has the following low-level generation rules:

$$\begin{aligned}
&gd(lsv) : lsv(T), T < T' \leftarrow \\
&\qquad holds_at(finished_work, T_{now}), \\
&\qquad time_now(T_{now}), T' = T_{now} + 6. \\
&gd(nfa) : nfa(T), T < T' \leftarrow \\
&\qquad holds_at(away, T_{now}), \\
&\qquad time_now(T_{now}), T' = T_{now} + 3. \\
&gd(lba) : lba(T), T < T' \leftarrow \\
&\qquad holds_at(low_battery, T_{now}), \\
&\qquad time_now(T_{now}), T' = T_{now} + 2.
\end{aligned}$$

- KB_{GD}^{aux}: Each goal is assigned a category, respectively, *required*, *optional*, and *operational*, and priority relationships amongst these categories are specified, i.e.:

$$\begin{aligned}
&typeof(lsv, required) \\
&typeof(nfa, optional) \\
&typeof(lba, operational) \\
&more_urgent_wrt_type(required, optional) \\
&more_urgent_wrt_type(operational, required) \\
&more_urgent_wrt_type(operational, optional)
\end{aligned}$$

  Note also that the three goals are pairwise incompatible, i.e. that the computee can only do one of these goals at a time.

- KB_{GD}^{high}: The following preferences apply:

$$\begin{aligned}
gd_pref(X, Y) : gd(X) < gd(Y) \quad \leftarrow \quad & typeof(X, XT), \\
& typeof(Y, YT), \\
& more_urgent_wrt_type(YT, XT).
\end{aligned}$$

Starting initially, at time $T_{now} = 0$, with a state S_0 where $Goals$ and $Plan$ are empty and KB_0 contains only the observation that $finished\_work$ holds at time 0, we have the following computational operational trace.

- If $time\_now(1)$, a GI transition is preferred by the normal cycle theory as both $Goals$ and $Plan$ are empty. No other transition is entailed credulously (via $\vdash_{pr}^{cred}$) by the cycle theory. The first element in the trace is therefore:

$$\vdash_{GI} (S_0, S_1, 1)$$

where the execution of $\vdash_{GI}$ calls the (computational counterpart of the) goal decision capability at $\tau = 1$, giving $KB \vdash^{\tau}_{GD} \langle lsv(T), \{T < 7\}\rangle$ and hence the new state S_1 of the computee will change so that now $Goals$ is given by:

$$Goals = \{g_1 = \langle (lsv, T), \bot^{nr}, \{T < 7\}\rangle\}$$

- Next, we suppose that a Passive Observation Introduction (POI) occurs when $time\_now(2)$ as again the normal cycle theory would choose this when indeed new obsevations have arrived in the meantime. We assume that this is the case when the new observation is that the battery is low. At the same time Francisco's computee observes that Francisco has finished work. The second element in the trace is therefore:

$$\vdash_{POI} (S_1, S_2, 2)$$

where the execution of the $\vdash_{POI}$ transition simply changes the state into a new state with:

$$KB_0 = \{observed(finished\_work(T), 0), observed(low\_battery(T'), 2)\}$$

- Then the computee's normal cycle theory will choose at $time\_now(3)$ again a GI transition as the next transition. Now, because of the extra information acquired, the goal decision capability returns a low-battery alert (lba) as the only preferred goal and hence executing the GI transition will replace the previous goal with this. The third element in the trace is therefore:

$$\vdash_{GI} (S_2, S_3, 3)$$

where the new state S_3 has as $Goals$:

$$Goals = \{g_2 = \langle lba(T''), \bot^{nr}, \{T'' < 5\}\rangle\}$$

.

- The cycle theory will then choose as the next transition a Plan Introduction (PI) transition for the (only) goal of lba. No other transition is credulously derived via $\vdash^{cred}_{pr}$ by the cycle theory (note here we have assumed that no new observation has arrived at the computee). The fourth element in the trace is therefore:

$$\vdash_{PI} (S_3, S_4, 4)$$

where the new state S_4 differs from S_3 in $Goals$ and/or $Plan$ where a valid plan consisting of subgoals and/or actions for the goal g_2 is added.

- Depending on this plan the computee continues with further PI transitions in order to end up to executable actions, or if such actions already exist with AE transitions.

15 Discussion and future work

In this part of the document, we have described the computational model for the KGP formal model, obtained by integrating, as dictated by the KGP model, the computational counterparts of the various components of the model, namely its capabilities, transitions, selection functions, and cycle theories. We have defined the overall computational model top-down, from the operational traces, using cycle theories, to the transitions, invoked within the operational trace, and the selection functions, invoked within cycle theories to compute appropriate inputs for the transitions, to the capabilities, invoked by the transitions and selection functions. The capabilities, the fluent selection function and the cycle steps within the operational trace have been defined by relying upon two abductive proof procedures, one for abductive logic programming, to deal with the abductive tasks/capabilities underlying the model, the other for logic programming with priorities, to deal with the preferential reasoning tasks within the model. The proof procedures are C-IFF and Gorgias for $LPwNF$, respectively.

Here, we point out some limitations of the computational models we have defined and of the formal correctness results we have provided for them, as well as some possible future extensions of the computational model.

Limitations

- The computational counterparts for the planning and reactivity capability are proven correct in the case in which globally consistent sets of actions and goals are returned by C-IFF.
- The computational counterpart of the temporal reasoning capability is defined under some simplifying assumptions on the specification of temporal reasoning, namely that the computee makes no inconsistent observations and that no concurrent actions are recorded affecting the same fluents in a contradictory way.
- The computational counterparts of the temporal reasoning and goal decision capabilities are proven correct conditionally on the correctness and completeness of the underlying abductive proof procedure C-IFF. However, we have not proven completeness of C-IFF, and actually believe that C-IFF is not complete in general. However, we believe that C-IFF might be complete in the special case it is used within goal decision. Indeed, in that case C-IFF is used with a ground abductive logic program which seems to be acyclic, in the sense of [122], and the completeness result for IFF proven for acyclic abuctive logic programs in [122] might carry through to C-IFF.

Extensions

The computational model could be extended to remove some of the limitations above, in particular those related to temnporal reasoning. Moreover, it could be extending to incorporate the following extensions to the KGP model.

- Both the formal KGP model and its computational counterpart only deal with temporal variables, and assume that no other variables occur in actions and goals in the state of a computee. So, for example, an action of the form

$$\langle move(a, L, T), G, free(L, T), \{T < T'\}\rangle$$

indicating to move an object a to some unspecified but free location L at some time T before another time T', cannot be represented within the model. Similarly, a goal of the form

$$\langle at(a, L, T'), \bot, \{T' < 10, L \neq c\}\rangle$$

indicating the intention to have a at some location L different from location c, cannot be represented within the model.

We have restricted the models in this way to simplify them and concentrate with the issues involved in reasoning with temporal variables and constraints. However, the extension to the more general case is possible and seemingly not very difficult. This extension would amount to associating non-temporal constraints to goals and actions, alongside temporal ones, and non-temporal variables to fluent literals and action operators, alongside temporal ones. Moreover, this extension would require extending Σ to deal with instantiations of non-temporal variables and $\models_{\Re}$ and $\vdash_{\Re}$ to establish and check their satisfiability. Finally, AE, AOI and POI would have to be modified in order to possibly instantiate non-temporal variables when interacting with the environment via the sensing capability.

- We have assumed that temporal constraints are kept locally with goals and actions in the state. However, when using these constraints, we always consider all the other constraints in the state (indicated with TCS). Thus, we could have used the global constraint store TCS directly. This is the solution we have adopted within D9 [5], in the PROSOCS platform.

- We have assumed that observations are just recorded within KB_0, and no (credulous) explanation for their occurrence can be stored and used within the models. In some cases, it might be useful to generate and use such explanations, to cope with the incompleteness of information available to the computee. We have started investigating this extension within temporal reasoning, with the possibility to reason with actions whose time is existentially quantified within an interval (see appendix A).

16 Related Work

During the past few years we have witnessed an explosion of proposed models and architectures for individual agents. In deliverable D4, we have identified a number of such models and architectures mostly relevant to the KGP computee model proposed in D4. Here, we compare the computational counterpart for the KGP model that we have proposed earlier on in this deliverable with the computational counterpart for the models and architectures for agents proposed in the literature, where applicable. We identify similarities and differences between the computational counterpart of the KGP model and the computational counterparts of the relevant proposals. We only briefly report on the features of the approaches we compare ours with, and do not compare the approaches with the KGP model (as we have already done this in deliverable D4).

We will also stress the "distance" between abstract models and computational counterparts and, if applicable, any system built as a directly usable counterpart of the abstract and computational models. We will throughout refer to the prototype system that we have built

as a practical counterpart to the KGP (abstract and computational) model, as given in the companion deliverable D9.

As in D4, we start with a number of existing proposals that are popular in modelling agents and multi-agents systems, most notably, the classical BDI model [102], the modelling features of the agent-programming languages: Agent0 [109], AgentSpeak [99] and its variants, 3APL [57], and the agent-modelling framework DESIRE [17]. Then we compare KGP with existing computational logic-based approaches that use, as we do here, non-monotonic logic programming frameworks and techniques to model, specify and implement software agents. These include the work developed by the IMPACT project [8], the logic-based system $\mathcal{MINERVA}$ [87], the agent specification language GOLOG [88] and its variants.

We believe the existence of a clearly identified and provably correct computational counterpart for the KGP model, on which the implemented system is soundly grounded, to be one of the strenghts of the KGP model with respect to other agent models in the literature. Thus, through the comparison of computational counterparts of existing systems, we also implicitly stress the significance of the KGP model.

Throughout this section we advocate a number of advantages of the (computational counterpart of the) KGP model, which we summarise here as follows:

- The KGP model is equipped with a clearly identified, and provably correct computational counterpart, which synthesises within a single setting an abductive proof procedure in abductive logic programming (ALP) and a proof procedure for reasoning with dynamic priorities in logic programming (LP), both appropriately extended by means of a constraint solver (as in Constraint Logic Programming - CLP).

- In order to provide the features required by the KGP model, which are necessary to cope with the Global Computing challenges, we have defined novel extensions of existing computational logic techniques and proof procedures, thus pushing forward the state of the art within computational logic, rather than simply transferring know-how from computational logic to the Multi-Agent-Systems and Global Computing communities.

- The computational counterpart of the KGP model is defined modularly, by appropriately integrating computational counterparts of the different components of the KGP model (capabilities, transitions, selection functions, cycle theory). Different components might use the same underlying proof procedure, but adjusting it to their needs. The overall correctness result is obtained as a by-product of the correctness results for the individual components, modularly.

- The computational model we have proposed is flexible in that the concrete proof procedures that are at the heart of our model could in principle be replaced by other proof procedures.

- The prototype for the single computee that we have developed in the companion deliverable D9 is strongly coupled with the computational model for the KGP model proposed earlier on in this deliverable, and the gap between the computational model and the prototype is rather narrow. As a consequence of this and of the correctness results we have proved earlier on, the gap between the KGP model and the existing prototype is rather narrow.

16.1 The BDI model

Following [15], Cohen and Levesque [27] have formalised some philosophical aspects of Bratman's *Beliefs, Desires, and Intentions* theory [14]. In their formalism, intentions are defined in terms of temporal sequences of an agent's beliefs and goals, i.e. intentions depend on what is believed as time progresses, and what is desired to be achieved at specific times. In related work, Rao and Georgeff have developed a modal logic framework for agent theory based on the three primitive modalities of beliefs, desires, and intentions [101, 102]. Their formalism (referred to as BDI) is based on a branching model of time in which belief-, desire-, and intention-accessible worlds are themselves branching time structures.

To establish the link between the BDI theory and practice, Rao and Georgeff have also presented an architecture [98, 100] that focuses on practical/computational concerns (unlike that of Bratman et al), also illustrating how a BDI system can be designed to have data structures that correspond to beliefs, desires, and intentions, together with update and query operations on these structures. This design is advantageous, Rao and Georgeff argue, when an agent has to communicate with humans and other agents, and can be expected to simplify the building, maintenance, and verification of application systems.

However, the architecture does not rely on the use of modal-logic theorem provers, as one might have expected. The reason for this is that by using such computational tools the time taken to reason, and thus the time taken to act, is potentially unbounded, thereby destroying reactivity that is essential in the agent's survival. Instead, the update operations on the beliefs, desires, and intentions structures are controlled by an interpreter as shown below:

BDI-interpreter
initialise-state();
repeat
    options:= option-generator(event-queue);
    selected-options := deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-successful-attitudes();
    drop-impossible-attitudes();
end repeat

At the beginning of every cycle, the option-generator() reads an event-queue structure and returns a list of options. The deliberate() selects a subset of selected-options to be adopted and adds these to the intentions structure. If there is an intention to perform an atomic action at this point in time, the agent then executes it by calling execute(). Any external events that have occurred during the interpreter cycle are then added in the event-queue by calling get-new-external events(). Internal events are added as the occur. Next, the agent modifies the intention and desire structures by calling drop-successful-attitudes() and drop-impossible-attitudes() to deal with successful as well as unrealisable (or impossible) intentions and desires.

The ideas behind this new, computing-centric, abstract architecture is to bridge the gap, between BDI theory - presented in terms of the BDI architecture and the modal logics on the one hand, and a number of existing BDI implementations - most notably the work on the systems PRS [51, 49, 50, 60] and dMARS [52] on the other.

One major difference between BDI and KGP is that KGP is not based on a modal-logic approach to represent an agent's beliefs but instead it is based on a non-monotonic computational logic language which is certainly less expressive. This language choice allows us to have a provably correct computational model on which the prototype system of D9 is strictly based. Thus, the correspondence between agent specification and executable implementations is closer for our KGP model than for classical BDI.

Rao [99] argues that the complexity of the code written for classic BDI implementations such as PRS and the simplifying assumptions made by them have meant that these implementations have lacked a strong theoretical underpinning, and that the specification logics for BDI have shed very little light on the practical problems and, as a result, the two streams of work on theory and practice seem to have been diverging:

"... due to its abstraction this work was unable to show a one-to-one correspondence between the model theory, proof theory, and the abstract interpreter"

"... the holy grail of BDI agent research is to show a one-to-one correspondence with a reasonably useful and expressive language".

The development of our KGP model has been motivated by similar observations. We believe that our provision of computational counterpart in D8 and prototype based on it in D9 do provide a strong theoretical underpinning for the prototype and a practical counterpart for the model.

16.2 AGENT0

AGENT0 [109] is as an agent-oriented programming language that extends the AI language Lisp. It is probably one of the first attempts to promote a social view of computation based on the interaction of different co-operating agents. The approach is grounded on a multi-modal logic with an explicit representation of time, with modalities such as *beliefs* and *commitments*, and communication primitives, such as REQUEST and INFORM.

Using the AGENT0 architecture an agent has four component data structures: a set of *capabilities* – specifying what an agent can or is able to do; a set of *beliefs* – stating what an agent believes at certain times and about certain times; a set of *commitments* – representing the actions that the agent ought to do at specific times; and a set of *commitment rules* – describing how new commitments can be introduced or old commitments can be dropped.

An agent cycle interprets commitment rules in AGENT0 roughly as follows: a new incoming message may update the beliefs and will be matched against the agent's message conditions of the commitment rules set. These conditions are then matched against the beliefs of the agent. If the commitment rule fires (i.e. both the message and the conditions of the commitment rule are satisfied), then the agent becomes committed to the action. The execution of an action then may update the commitments and the beliefs of the agent.

AGENT0 was only intended as a prototype, to illustrate the principles of agent-oriented programming, and was refined by Thomas in the Planning Communicating Agents (PLACA) language [114]. Despite the improvements of PLACA over AGENT0, the language still inherits the gap that there is in AGENT0 between the high-level concepts such as beliefs and commitments and the implementable architecture. In contrast, the computational model that we have provided in this deliverable serves to bridge the gap between the formal KGP model and its implementation (see deliverable D9). Moreover, the formal way in which the KGP model is defined facilitates the concrete computational counterpart we have provided here. For example, the notion of how beliefs persist in AGENT0 is described in terms of informal guidelines, which

need to be followed at the agentification (implementation) stage. Instead, in KGP, the way the knowledge persists and changes is described by means of concrete event calculus axioms that are precisely given and whose semantics is precisely specified and so can be directly executed by adopting a suitable abductive proof procedure.

16.3 AgentSpeak

The first version of AgentSpeak [120] attempted to provide an agent-oriented programming language with BDI–like modelling capabilities such as PRS [60] and appropriate language constructs, influenced by work in object-based programming languages.

Like with PRS, however, there was a large gap between AgentSpeak programs and the theory of the BDI model. To bridge this gap, Rao in [99] proposes AgentSpeak(L), a programming language that can be viewed as an abstraction of the BDI implemented systems (such as PRS - described in [49] and dMARS - in the way formalised in [35]) and allows agent programs to be written in a restricted first-order language with events and actions. In this context, Rao argues that the shift in perspective of taking a simple specification language as the execution model of an agent and then ascribing the mental attitudes of beliefs, desires and intentions, from an external viewpoint is likely to have a better chance of unifying theory and practice. It is worth saying here that KGP has been constructed very much in this spirit.

In AgentSpeak(L), an agent contains, besides beliefs, plans, and intentions, also events, actions, selection functions. The selection of plans, their adoption as intentions, and the execution of these intentions are described via an operational semantics in terms of an interpreter that runs the agent programs specified in AgentSpeak(L). The beliefs, desires and intentions are not defined as modal formulas, but instead as a set of base beliefs (or, as Rao puts it: *facts in the logic programming sense*).

The proof theory of AgentSpeak(L) language is given by a labelled transition system. The notion of *configuration* is a labelled description of the events, beliefs, intentions, and actions of an agent. Proof rules define how the agent moves from one configuration to the next. It is argued that these transitions have a direct relationship to the operational semantics of the language and hence help to establish the strong correspondence between the interpreter and the proof theory, although no formal proofs are provided. AgentSpeak(L) also suffers from proof–rules being embedded in the cycle algorithm, that is, they are not defined separately and modularly, as in the KGP model and in the computational counterpart we provide in this document. In any case, undoubtedly the AgentSpeak(L) work has opened up an alternative, restricted, first–order characterisation of BDI agents, which somewhat bridges the gap between theory and practice.

Up until recently, there was no implementation of the AgentSpeak(L) interpreter. Bordini et al [11, 12] have implemented an extended AgentSpeak(L) interpreter which they call AgentSpeak(XL). This extended interpreter allows the programmer to handle extensions of the original AgentSpeak(L) (e.g. plan failures), by means of a quantitative model of goals, plans, and the environment. In a broad sense, the approach supporting these extensions is similar to the KGP cycle approach, in its preference-based orchestration of the various capabilities, and in the representation of goals and plans. However, the lack of a semantics of the quantitative model integrated within the overall semantics, makes difficult to prove the correspondence of the model with its computational counterpart (which is only informally sketched), and hence, any relative (correctness) property.

16.4 3APL

Another programming language for agent programming relevant to KGP is 3APL, presented in a number of articles by Hindriks et al [56, 57, 59, 58]. Unlike KGP which is based purely on declarative logic programming, the 3APL language is a combination of imperative and logic programming. From the imperative programming viewpoint, 3APL inherits the full range of regular programming constructs, including recursive procedures and state-based computation. States of agents in 3APL, however, are belief (or knowledge) bases, which are not characterised by the usual variable assignments of imperative programming. From the computational logic perspective, also taken by our KGP model, answers to queries in the beliefs of a 3APL agent are proofs in the logic programming sense.

At run–time an agent program in 3APL is viewed as consisting of a set of Beliefs, a set of Basic Actions, a set of (Achievement and Test) Goals, and a set of Practical Reasoning Rules. Although the beliefs in [57] are exemplified in a logic programming like (first-order) language with integrity constraints, any logical language, even a modal language could in principle be used. In KGP, however, we have already seen that the representation language is fixed to be a combination of ALP, CLP, and LP with dynamic priorities. Achievement goals act like procedures in imperative programming and have a procedural meaning for things that agent has to do. Test goals allow the agent to query its beliefs, and are evaluated relative to the current beliefs of the agent. Together, the basic actions, achievement goals and test goals are the *basic goals* of the 3APL language. *Complex goals* are then composed from basic goals by using imperative programming constructs for sequential composition and non-deterministic choice. There are many similarities between Practical Reasoning Rules in 3APL and some parts of the KGP model, as we discuss in D4.

The operational semantics of 3APL is specified in terms of transition rules, which – similarly to AgentSpeak(L) – are relations on so-called configurations. Two distinct classes of transition rules are defined. The first type defines what it means to execute a single goal given the current belief base of an agent and the current computational state (which include the variable bindings). At this level, transition rules provide a formal specification for 3APL basic actions, achievement goals, test goals, complex goals, and practical reasoning rules. The second type of transitions is defined in terms of the first type and defines what it means to execute an agent in terms of executing multiple goals.

To deal explicitly with selection mechanisms for goals and actions, 3APL introduces a separate formal specification for the control structures of the agent language. A second transition system is introduced, called the *meta-transition* system, which includes features for referring to the object level language, as well as operators for programming control structures for the object (agent) level. The meta-transition system also supports a set of basic actions that allow the agent to select, apply rules, and execute goals. For this purpose four actions are introduced: (1) an action for selecting an applicable rule, (2) an action for the application of a number of rules, (3) an action for selecting an enabled goal, and (4) an action for the execution of a set of goals. Finally, constructs for expressing the preference order over goals and rules (such as those in the practical reasoning rules base) are also provided.

The control structure that is proposed by 3APL [30] is a specialisation of a one-size-fits-all update-act interpreter [109, 99, 79] as follows:

Update-Act Cycle
1 Select a rule R to fire

2 Update the goal base by firing R
3 Select a goal G
4 Execute (part of) G
5 Goto 1

In [30] an extension of the meta-language is discussed, whose aim is to make the cycle of 3APL programmable. Within the KGP model, we share the 3APL aim to make the behaviour of agents/computees programmable and the selection mechanisms explicit [30]. The programmable meta-language, which is based on control constructs that are in the style of the imperative programming languages, differs from the programmable "control layer" of the KGP model, which instead is based on computational logic. Moreover, while the programmable cycle of 3APL is based on tests (entailment relations of the underlying logic), set-like selection operations and control structures, the KGP one orchestrates a versatile set of modular components that exhibit more complex reasoning capabilities.

These differences are reflected also at the computational level. In the case of 3APL, the semantics prescribe how to integrate computational logic and imperative control structures, in the case of KGP the semantics more directly corresponds to the computational model, as shown in this document, and the computational model defines, almost straightforwardly, the implementation. According to the general aims of the project, we expect that our use of declarative control theories with a clear and provably correct computational counterpart will give us a solid base on which to formally reason about the properties and the behaviour of a computee.

16.5 DESIRE

DESIRE (DEsign and Specification of Interacting REasoning components) is a high–level modelling framework that explicitly models the knowledge, interaction, and coordination of complex tasks and reasoning capabilities in agent systems [18, 17, 16]. The framework views both individual agents and the overall system in terms of a compositional architecture – where functionalities are given by interacting, task-based, hierarchically structured components.

Tasks are characterised in terms of their inputs, their outputs and their relationship to other tasks. Interaction and co–ordination between components, between components and the external world, and between components and users is specified in terms of informational exchange, sequencing information and control dependencies. The components themselves can be of any complexity, from simple functions and procedures up to whole knowledge-based systems, and can perform any domain function (e.g. numerical calculations, information retrieval, optimisations, etc).

In [17], DESIRE has been extended to define a generic BDI model to incorporate beliefs, desires and intentions. The result is a more specific BDI agent where an agent's task control is capable of six tasks: the own process control deals with how the agent determines its own beliefs, desires and intentions, the agent specific tasks deals with the agent performing its own tasks, the world interaction management deals with managing interaction with the environment, the agent interaction management deals with communication with other agents, the maintenance of world information deals with modelling the world, and the maintenance of agent information deals with modelling other agents.

The formal specification of DESIRE is based on a many–sorted predicate logic [37], which distinguishes between object-level and meta-level descriptions of components. The dynamics

of the overall compositional system in DESIRE is modelled through temporal models based on temporal logic [19].

In this perspective, the approach of DESIRE is mainly oriented to the verification of the properties of the overall system, given the specification of the agents which compose it and their interactions. The computational behaviour of each agent, responsible for a (sub-)task, is based on a *if-then* conditional rule model (which may be further specified). The temporal logic analysis is carried out to verify properties of interest of the compound system, as resulting from the interaction of the agents. In this sense, DESIRE is more relevant to the prosecution of our project, concerned with the definition and verification of properties of societies of computees, than to the computational model of the KGP model, which relies on slightly different assumptions.

16.6 Computational logic-based approaches

16.6.1 IMPACT

The principal goal of the IMPACT (Interactive Maryland Platform for Agents Collaborating Together) project [8, 39, 38, 40, 41, 113], has been to develop both a logic-based theory as well as a software implementation that facilitates the creation, deployment, interaction, and collaborative aspects of software agents in a heterogeneous, distributed environment, representing hence a complete framework which covers many aspects of agent development and execution support.

The IMPACT project proposes a unifying approach for many different features of agent behaviour based on the adoption of computational logic as the underlying methodology for system development and analysis. It provides a set of servers (yellow pages, thesaurus, registration, type and interface) that facilitate agent inter-operability in an application independent manner. It also provides an Agent Development Environment for creating, testing, and deploying agents.

IMPACT is centered on the integration, based on agentification, of heterogeneous, possibly legacy, information sources, and in their cooperation in order to successfully accomplish a coordinated task. Agents encapsulates information sources via a logical representation, mixing classical execution of code with logical-based reasoning: their behaviour consists of the execution of actions (code), as prescribed by the agent program. Such a program is defined upon the logical representation in a computational logic declarative style (e.g. preconditions, integrity constraints, postconditions...), featuring also modal operators, like deontic constructs (e.g. possible and necessary actions, ...). The underlying architecture supports the interaction with other agents, for instance by allowing communications. Semantics prescribes which actions are feasible, permitted, required, forbidden to be (concurrently) executed in the current state, and it can be sometimes mapped onto classical non-monotonic reasoning semantics.

The different motivations of KGP and IMPACT make apparent the different ways agents are modelled in the two systems, most notably differences in the treatment of beliefs and actions (for instance, KGP has not modal operators), and also in the treatment of temporal reasoning, planning, goals, communication, and control (see Deliverable D4 for a more precise comparison of the two models, [63]). These differences are reflected on the computational level.

The computational model of IMPACT is based on specialised, sound, iterative, fix-point computational procedures that compute the behaviour of the agent, as fix-point sets of actions to be executed, according to the agent program and the current state. This determines a state-transformation cycle that depends on the current state and the received messages. Com-

putational procedures are also complete, and polynomial–time under certain conditions, even if the more expressive ones are generally NP.

The KGP model does not deal with modal operators and concurrency in action execution, and with mechanisms to guarantee a correct course of actions in this sense (this is subject for future extensions). The computational counterpart of the KGP model hence consists of an orchestration of a set of modular proof procedures, which might in principle be selected according to the problem at hand, and are subject to a "programmable" control cycle. Moreover, while the notion of agentification makes the distinction of the implementation of an agent from its abstract model not always so clear, our work keeps the separation of the computational model from the formal model (and the implementation) as a main concern.

Overall, the KGP model, and its computational counterpart, focus on building autonomous agents by integrating existing logic programming techniques and their extensions in order to cope with highly dynamic nature of open and global computing environments.

16.6.2 $\mathcal{MINERVA}$

$\mathcal{MINERVA}$ is an agent framework which exploits computational logic as a means for integrating diverse non-monotonic formalisms within a unique (dynamic) model, [87, 86]. The basic architecture consists of a structured knowledge base encompassing both the knowledge of the agent, i.e. its representation of the environment and other agents, and (BDI-like) features such as capabilities, intentions, goals, and plans. The knowledge base is controlled by a varying set of modules, each of which is devoted to a specific task, like, for instance, a communicator, a sensing and reacting module as interface with the environment, a planner, a learner, and a scheduler module. All of these components update the agent's knowledge base. The architecture is modular in that it is composed of functional sub-agents, which may add and remove functionality to the single agent.

$\mathcal{MINERVA}$ relies on the Multidimensional Dynamic Logic Programming (MDLP) model [85] and Knowledge and Behaviour Update Language (KABUL) [86], in order to represent the dynamic evolution of an agent's knowledge. MDLP is a non–monotonic LP–based model which allow representation of (modules in) the knowledge base of agents by means of (generalised) logic programs, and it is equipped with both an answer-set-based declarative semantics and an operational semantics. KABUL is a declarative representation of state transitions, namely behaviours.

The $\mathcal{MINERVA}$ project is more recent than the longer running projects previously cited. Some of its components are provided with a clear semantics, and implementations of the updating languages are also provided as meta-interpreters (and pre-processors) over Prolog and DLV [1] systems. However, the definition of a model for the overall agent computational part, which emerges from the interaction of the separate sub-agents, seems not determinant in the passage from the semantics to the implementation. Indeed, the definition of a computational model, (formally) proved to correspond to the semantics, is less emphasised in this approach than it has in ours.

Anyway, given the similarities between the two projects, e.g. the declarative and operational use of computational logic for representing agent knowledge, the modularity of agent architecture, and the interest for open and dynamic environments that evolve non-monotonically, we argue that the comparison of the forthcoming advances of the KGP and $\mathcal{MINERVA}$ models, and also of their computational counterparts, will be interesting for both projects.

16.6.3 GOLOG

GOLOG, after alGOl LOGic, [88] is another approach to a logic-based modelling of multi-agent systems. As the name suggests, GOLOG is a language which tries to import the programming paradigm of a procedural language like Algol into the realm of logic. In particular, it is based on the situation calculus [93], which represents a sophisticated logic of actions.

GOLOG is provided with procedural constructs like sequencing, choice and iteration of situation transforming actions, and it has a computational implementation based on Logic Programming. An explicit representation of the dynamic world being modelled evolves according to actions, which are characterised by (user supplied) axioms about their preconditions and effects. This allows programs to reason about the state of the world and consider the effects of various possible courses of action before committing to a particular behaviour.

A concurrency-based extension of the original language [53] provides a high-level agent control based on facilities for prioritising the concurrent execution, interrupting the execution when certain conditions become true, and dealing with exogenous actions. This sort of high-level agent control constitutes an alternative to planning, being the course of actions of a plan determined by the concurrent synchronisations that actions and conditions induce on each other. A distinguishing feature with respect to other procedural formalisms for concurrency, which is valuable when modelling open systems, is the possibility of dealing with incompletely specified states, which represent a partially accessible environment. However, the original research line of GOLOG is based on off-line planning the course of actions to be executed.

This logic-based approach to modelling the evolutions of a dynamic world shares with our work here many motivations, and in particular the interest in a representation of the agent's state which allows for reasoning about changes occurring over time.

To evaluate programs in an open world setting, an extension of Golog and ConGolog is specified in [54], and is known as IndiGolog (Incremental deterministic Golog). This system allows a programmer to specify guarded action theories, that can determine, by dynamically checking conditions, the on-line (and off–line) execution of programs. In the on-line execution case, a sensing capability affects the current state of the computation, which is obtained by incrementally executing programs represented as guarded theories. Such implementation is provably correct under certain conditions, and is reminiscent of the KGP combination of sensing capability combined with the knowledge revision obtained by event calculus theories.

However, the GOLOG family, being based on the quite sophisticated situation calculus, requires computational counterparts that are closer to full-fledged theorem-provers than to classical computational logic, as exploited by the KGP model, and are typically less computationally effective.

Part III
Societies

Abstract. In this part we present the operational counterpart, as a proof system based on rewriting, for the society infrastructure, and in particular for the part devoted to checking goal achievement and compliance to the protocols specified at society level of (computees') interactions, expressed, in their turn, as socially relevant events. We start by recalling the abductive interpretation of the society, and by restating its declarative semantics (updated with respect to Deliverable D5 [94]). We then present the social level proof procedure, that is inspired by, and extends the C-IFF proof procedure. The extension allows accepting new events, producing a set of expectations, and detecting violation. It deals with constraints, involving both existentially and universally quantified variables. Finally, we state the desirable properties of the Society Constrained IFF proof Procedure ($\mathcal{S}$CIFF in the following). In particular, we state soundness and completeness properties with reference to the (declarative) semantics given in Deliverable D5 [94] for the society model.

The society model envisaged in the first year of the project and presented in Deliverable D5 [94] is capable of modelling interactions among computees in an open environment, and also supports goal-directed behaviour of societies of computees.

The main features achieved by the society model are:

(i) the use of Computational Logic (CL, for short) to model and give semantics to interactions; and

(ii) the use of a uniform formalism (based upon what we named Social Integrity Constraints) for expressing both interaction protocols and "social" semantics of communication languages.

The society infrastructure is devoted to checking the compliance of a society member's behaviour, with respect to the expectations of the society, as by the integrity constraints and required to achieve a goal of the society. Compliance to the specified protocols of interactions, communication language semantics and required behaviour to achieve certain goals can be checked by a suitable CL-based proof procedure.

In the second year of the project, in Workpackage 3, the Consortium aims at providing a computational counterpart for both the single computee model and the society model. In the following, we focus on this latter.

To this purpose, this part is based on the abductive interpretation of the society model (introduced in D5), since this interpretation smoothly allows for re-using existing proof procedures for Abductive Logic Programming (ALP). These proof procedures need to be extended, of course, due to the richer syntax of the society knowledge bases and social integrity constraints, the need for assimilating new events coming interactively and the need for dynamically checking fulfillment and violations.

In Section 17 we recap the formal society model defined in the Deliverable D5, [94], together with some updates done in the context of Workpackage 3.

In Section 18 we present the society model as ALP and its revised declarative semantics.

In Section 19 we show the modifications we considered necessary to the IFF proof procedure, to take into account the society model defined in Section 18.

In Section 20, we state desirable properties (in terms of soundness and completeness) of the society proof procedure under definition. In particular, for stating these properties we refer to the (declarative) semantics given for the society model. Proofs for soundness properties are given in Section 20.

17 Society formal model: Recap and Update

17.1 The Syntax of the Society

The society knowledge consists of the following 4-tuple [94]:

$$\langle SOKB, SEKB, \mathcal{IC}_S, \mathcal{G} \rangle$$

where:

- $SOKB$ is the *Social Organization Knowledge Base*,
- $SEKB$ is the *Social Environment Knowledge Base*,
- $\mathcal{IC}_S$ is the set of *Social Integrity Constraints* (IC_S), and
- $\mathcal{G}$ is the set of *Goals* of the society.

Social Environment Knowledge Base. The $SEKB$ dynamically evolves and is composed of:

- *Happened events*: atoms indicated with functor **H**
- *Expectations* on the future: events that should (but might not) happen in the future (atoms indicated with functor **E**), and events that should not (but might indeed) happen in the future (atoms indicated with functor **NE**).

The happened events are not all the events that happen, but only those observable and relevant for the society, as discussed in deliverable D5 [94]. These events make up the history **HAP** of the society, and are represented as ground atoms

$$\mathbf{H}(Event[, Time]).$$

The expectations can be

$$\mathbf{E}(Event[, Time])$$
$$\mathbf{NE}(Event[, Time])$$

and can contain variables, with the following scope rules and quantifications:

- variables in **E** atoms are always existentially quantified with scope the entire set of expectations
- the other variables, that occur only in **NE** atoms are universally quantified (the scope of universally quantified variables is not important, as $\forall X.p(X) \wedge q(X)$ is equivalent to $\forall X.p(X) \wedge \forall Y.q(Y)$).

Social Organization Knowledge Base. The SOKB is a logic program, consisting of clauses

$$\begin{array}{rcl} Clause & ::= & Atom \leftarrow Body \\ Body & ::= & ExtLiteral\ [\ \wedge ExtLiteral\ ]^{\star} \\ ExtLiteral & ::= & Literal \mid Expectation \mid Constraint \\ Expectation & ::= & [\neg]\mathbf{E}(Event\ [,T]) \mid [\neg]\mathbf{NE}(Event\ [,T]) \end{array} \tag{7}$$

In a clause, the variables are quantified as follows:

- Universally, if they occur only in literals with functor **NE** (and possibly constraints), with scope the body;
- Otherwise universally, with scope the entire *Clause*.

We call *definite* the predicates for which there exists a definition; i.e., a predicate that occurs in at least the head of a clause.

Goal. The goal $\mathcal{G}$ of the society has the same syntax as the *Body* of a clause in the SOKB. Notice that the variables occurring in G are considered *free* by the IFF proof procedure. In the devised proof procedure for the society infrastructure (named $\mathcal{S}$CIFF in the following), for ease of presentation and without loss of generality, they will be considered as existentially (or where appropriate as universally) quantified variables. Coherently with the SOKB, the variables occurring in $\mathcal{G}$ are quantified

- *existentially* if they occur in a definite literal, or literals with functor **E**;
- *universally* if they occur only in literals with functor **NE**.

Any variable in $\mathcal{G}$ must occur in at least a literal **E** or **NE**.

Social Integrity Constraints are in the form of implications. We report here, for better readability, the characterizing part of their syntax (the full syntax is given in document D5):

$$\begin{array}{rcl} ic & ::= & \chi \rightarrow \phi \\ \chi & ::= & (HEvent|Expectation)\ [\wedge BodyLiteral]^{\star} \\ BodyLiteral & ::= & HEvent|Expectation|Literal|Constraint \\ \phi & ::= & HeadDisjunct\ [\ \vee HeadDisjunct\ ]^{\star}|\bot \\ HeadDisjunct & ::= & Expectation\ [\ \wedge (Expectation|Constraint)]^{\star} \\ Expectation & ::= & [\neg]\mathbf{E}(Event\ [,T]) \mid [\neg]\mathbf{NE}(Event\ [,T]) \\ HEvent & ::= & [\neg]\mathbf{H}(Event\ [,T]) \end{array} \tag{8}$$

Given a IC_S $\chi \rightarrow \phi$, χ is called the *body* (or the *condition*) and ϕ is called the *head* (or the *conclusion*).

The rules of scope and quantification are as follows:

1. Any variable in an IC_S must occur in at least an *Event* or in an *Expectation*.
2. The variables that occur both in the body and in the head are quantified universally with scope the entire IC_S.
3. The variables that occur only in the head must occur in at least one *Expectation*, and

(a) if they occur in literals **E** or ¬**E** are quantified existentially and have as scope the disjunct they belong to;

(b) otherwise they are quantified universally.

4. The variables that occur only in the body are quantified inside the body as follows:

   (a) if they occur only in conjunctions of ¬**H**, **NE**, ¬**NE** or *Constraint*s are quantified universally;

   (b) otherwise are quantified existentially.

5. Of course, the order of the quantifiers is, in general, significant. In our syntax, the quantifier ∀ cannot be followed by ∃.

17.2 Syntax Update

The syntax proposed in document D5 [94] has been restricted, concerning Workpackage 3. We list the introduced restrictions together with the motivation.

Quantification of negative literals The IFF proof procedure imposes syntactic restrictions on the integrity constraints, on the clauses and on the goal. These restrictions are due to the treatment of negation and quantifiers (that are implicit, rather than explicit). In the IFF proof procedure, all atoms are given the standard Herbrand interpretation of logic programming.

Our language gives a specific meaning to some of the atoms; as in CLP, we give interpretation to some atoms called *constraints*. Moreover in atoms **H**, **E**, and **NE** the quantifiers are treated explicitly. It is reasonable that, for variables that do not occur in the interpreted atoms, we inherit the same syntactic restrictions of the IFF proof procedure; while for the other variables we have a specific treatment (see Section 19). We extend the IFF proof procedure allowedness condition, as follows.

Definition 17.1. *A clause* $Head \leftarrow Body$ *is* allowed *if every variable that occurs in a negative literal of a definite predicate*

- *occurs in at least a positive literal or in the head (as in the IFF)*
- *or it occurs in atoms* **E** *or* ¬**E**.

A Goal is Allowed *if every variable that occurs in a negative literal of a definite predicate*

- *occurs in at least a positive literal (as in the IFF)*
- *or it occurs in atoms* **E** *or* ¬**E**

The aim of these definitions is to ensure that the quantification of variables in negative literals in the resolvent cannot be universal.

The IFF also imposes a condition on the integrity constraints:

> An integrity constraint is allowed if (and only if) every variable in the conclusion occurs in the condition.

We do not need this condition, because we can always convert a non allowed integrity constraint by adding a new predicate. E.g., the integrity constraint:

$$\mathbf{E}(p(X)) \rightarrow \mathbf{E}(q(Y))$$

can be converted into

$$\mathbf{E}(p(X)) \rightarrow r.$$

$$r \leftarrow \mathbf{E}(q(Y)).$$

Notice that, due to the syntactic restrictions already in document D5, a variable cannot occur in a Social Integrity Constraint only in negative, definite literals, but it must always appear in literals with predicates **H**, **E**, **NE**.

Quantification of E atoms The rules of quantification given in deliverable D5, explain that

> variables in **E** atoms are always existentially quantified, and their scope is the entire set of expectations.

On the other hand, universally quantified variables may occur in **NE** atoms. Consider the example:

$$\mathbf{NE}(p(X)) \rightarrow \mathbf{E}(q(X)). \tag{9}$$

Since variable X is universally quantified with scope the whole social integrity constraint, if the current set of expectations contains $(\forall Y)\ \mathbf{NE}(p(Y))$, it should also contain (for the satisfaction of IC_S 9):

$$\forall Y\ \mathbf{E}(q(Y))$$

which is not in the syntax of the SEKB. A similar consideration can be done for $\neg\mathbf{H}$ literals; the IC_S:

$$\neg\mathbf{H}(p(X)) \rightarrow \mathbf{E}(q(X))$$

entails that

$$(\forall X)\ \mathbf{E}(q(X))$$

if the history is empty.

Also, as explained in the definition of the society's proof procedure (Section 19), the negative literals of defined predicates are handled as by the IFF proof procedure. Thus, when occurring in the body of an implication, they are moved to the head. We must ensure that no defined atom is invoked with universally quantified variables.

For these reasons, we define the following allowedness condition:

Definition 17.2. *A Social Integrity Constraint is* Quantifier Allowed *if any variable occurring in the head in literals of type* **E**, $\neg$**E**, *or (in the body) in a negative, defined literal*

- *either does not occur in the body*
- *or it occurs in the body in a literal of type* **H**, **E**, $\neg$**E**.

The society knowledge is quantifier allowed if all the Social Integrity Constraints are quantifier allowed.

Constraints and quantifier restrictions In document D5, CLP constraints can be applied to universally quantified variables. During Workpackage 3 we refined the concept, and introduced quantifier restrictions [24], distinguished from constraints.

For this reason, in the equations defining the BNF of SOKB and Social Integrity Constraints (Equations 7 and 8 in Section 17.1), one should read "quantifier restrictions" instead of "constraints". The two concepts coincide for existentially quantified variables, but are different for universally quantified ones.

For example, the sentence

$$(\forall_X)\ \mathbf{NE}(p(X)), X > 0$$

should be interpreted as

$$(\forall_{X>0})\ \mathbf{NE}(p(X))$$

which means

$$(\forall_X)\ X > 0 \rightarrow \mathbf{NE}(p(X))$$

or *"for all X greater than zero, $p(X)$ is expected not to happen"*. Quantifier restrictions will be discussed in greater detail in the operational semantics of the society (Section 19.1.1).

Quantifier restrictions occurring in the body of Social Integrity Constraints From the syntax in document D5, we have that variables occurring only in **NE**, ¬**NE** and ¬**H** literals in the body of a Social Integrity Constraint are universally quantified with scope the body of the IC_S.

Propagation of Quantifier Restrictions (QR in the following) for these variables would be very difficult to manage, and computationally expensive. For example, checking that $\forall_{X<Y}\mathbf{NE}(p(X,Y))$ is not trivial, in general.

Moreover, implementing a constraint solver that efficiently deals with general quantifier restrictions would be very complex. We restrict ourselves to quantifier restrictions that are *unary*, i.e., in which only one variable is involved (except for variables occurring in **H**, which will become ground when they are unified with a corresponding atom). We leave as a possible extension for future work the treatment of general quantifier restrictions.

For these reasons, we give the following definition:

Definition 17.3. *A Social Integrity Constraint is* Constraint Allowed *if*

- *all the variables that are universally quantified with scope the body do not occur in quantifier restrictions;*
- *the other variables (that occur only in the head, or both in the head and in the body) can occur in quantifier restrictions. For each quantifier restriction c occurring in the Social Integrity Constraint,*
  - *either c only involves variables that also occur in* **E**, ¬**E**, **H** *atoms*
  - *or it involves* one *variable that also occurs in at least one* **NE** *atom and possibly other variables each of which occurs in* **H** *atoms.*

Notice that the C-IFF and $\mathcal{S}$CIFF proof procedures have different syntactic restrictions, concerning CLP constraints occurring in Integrity Constraints. In $\mathcal{S}$CIFF, variables in a CLP constraint must occur also in expectations or happened events, thus, all the variables in a CLP constraint will eventually occur in an abduced atom or become ground.

Variable quantification problems can also happen due to unfolding; e.g.,

$$\mathbf{E}(a(X)), p(X) \rightarrow \mathbf{E}(b(X))$$

$$p(X) \leftarrow \mathbf{NE}(d(X,Y)), Y > 1.$$

In this case, Unfolding is applicable (see Section 19.2.1), and would give the following situation (we skip the copy phase for the sake of simplicity):

$$\mathbf{E}(a(X)), \mathbf{NE}(d(X,Y)), Y > 1 \rightarrow \mathbf{E}(b(X)).$$

so the partially-solved integrity constraint would not be Constraint Allowed. For this reason, we give a similar definition also for clauses:

Definition 17.4. *A* Clause is *Constraint Allowed if the variables that are universally quantified with scope the body do not occur in quantifier restrictions, and each variable that occurs in a quantifier restriction also occurs in at least one atom* **E** *in the body.*

Definition 17.5. *A* Society Knowledge is Constraint Allowed *if all its social integrity constraints and all its clauses in the SOKB are Constraint Allowed.*

18 ALP Interpretation of the Society model and declarative semantics

Abduction has been widely recognized as a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge [29, 43, 70]. Incomplete knowledge is handled by labeling some pieces of information as abducibles, i.e., possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. More formally, given a theory T and a formula G, the goal of abduction is to find a (possibly minimal) set of atoms Δ which together with T "entails" G, with respect to some notion of "entailment" that the language of T is equipped with.

Operationally, the idea is to exploit abduction for generating expected behaviour on the part of computees inhabiting the society. For our purposes, abduction smoothly allows for modeling hypotheses about expected/forbidden events, and a suitably extended abductive proof procedure can be used for (social) integrity constraint checking. In the sequel, we often refer to the latter function together with the function of detecting fulfillment and violation of expectations, as compliance check, for the sake of brevity.

18.1 The society and society instance as an Abductive Logic Program

In the following, we recall the society model as Abductive Logic Program, presented in D5 [94], and refine it. In particular, we introduce the notion of *instance* of a society as an Abductive Logic Program in order to capture the dynamic aspects of a society.

The (static) model of a society, $\mathcal{S}$, is represented as the following triple:

$$\langle SOKB, \mathcal{E}, \mathcal{IC}_S \rangle$$

where:

- $SOKB$ is the *Social Organization Knowledge Base*,
- $\mathcal{IC}_S$ is the set of *Social Integrity Constraints*, and
- $\mathcal{E}$ is the set of abductive predicates.

$\mathcal{E}$ consists of expectations. Expectations are events that should (but might not) happen (atoms indicated with functor **E**, which we called positive expectations), and events that should not (but might indeed) happen (atoms indicated with functor **NE**, which we called negative expectations). Expectations can also be negated (by negation $\neg$).

The *Social Organization Knowledge Base* ($SOKB$) provides specifications for society goals. In $SOKB$, clauses may contain in their body positive and negative expectations (possibly negated) about the behaviour of computees, and auxiliary literals (positive and negative) too, while their heads are atoms which possibly correspond to society's goals.

Finally, *Social Integrity Constraints* ($\mathcal{IC}_S$) are forward rules, of kind *body* $\rightarrow$ *head*, which have in their *body* literals, conditions about happened events (possibly negated) and (positive and negative) expectations (possibly negated), and in their *head* (disjunctions of) conjunctions of positive and negative (possibly negated) expectations. *Happened events* are atoms indicated with functor **H**.

Furthermore, CLP-like constraints (and quantifier restrictions) can occur in the body of $SOKB$ clauses, and in the body and head of $\mathcal{IC}_S$ constraints.

For details about the formal syntax of the society tuple components, and variable quantification in particular, the reader can refer to Section 17 and to D5 [94].

In the following, we introduce the notion of instance of a society as an Abductive Logic Program (ALP), where abducible predicates correspond to **E** and **NE** (and their negation $\neg$).

In our framework, abducibles are positive and negative expectation literals. In fact, we represent the literals **E**, **NE** and their negation $\neg$**E** and $\neg$**NE** as positive abducible atoms, in accordance with the usual way abduction can be used to deal with negation [43, 72]. In this case, the additional constraint $\forall X\ A(X), \neg A(X) \rightarrow \bot$ is implicitly considered.

In the proof procedure, we will use three types of negation, depending on the type of negative literal.

- For expectations (i.e., abducibles), we adopt the same viewpoint as in ACLP [72]: for each abducible predicate A, we have also the abducible predicate $notA$ for the negation of A together with the integrity constraint $(\forall X) notA(X), A(X) \rightarrow \bot$. Following [72], we assume that Negation As Failure (NAF) will not be used on abducibles, as it would be not appropriate to apply this type of negation on expectations, which have no definition. The usual constraint $\forall X\ A(X) \vee \neg A(X)$ adopted in NAF has in fact no meaning in an open world, where we can have no expectation at all.
- For definite predicates, we will use the same rules as the IFF proof procedure.
- For happened events (indicated by the functor **H**), we will use a sort of Constructive Negation [112]. In particular, we allow for variables in $\neg$**H**() literals.

In the following, we formally define the notion of instance of a society, and extension and closure of an instance of a society.

Definition 18.1. *An* instance $\mathcal{S}_{\mathbf{HAP}}$ *of a society* $\mathcal{S}$ *is represented as an ALP, i.e., a triple* $\langle P, \mathcal{E}, \mathcal{IC}_S \rangle$ *where:*

- *P is the $SOKB$ together with the history of happened events* **HAP***;*
- *$\mathcal{E}$ is the set of* abducible predicates *of $\mathcal{S}$;*
- *$\mathcal{IC}_S$ are the social integrity constraints of $\mathcal{S}$.*

The set **HAP** characterizes the instance of a society, and represents the set of *observable* and *relevant* events for the society which have already happened. Note that we assume that such events are always ground.

In this way, our social framework (and its dynamic counterpart, as instance of a society) has been smoothly given an abductive interpretation. This is interesting in its own right, and plays a role in Workpackage 3, in order to exploit well-known proof-theoretic techniques to generate expectations about social behaviour of members, and to check members' compliance to these expectations.

If the society is goal driven, then there exists a goal G at the society level (which is simply *true* if the society is not goal driven).

Definition 18.2. *Given two instances, $\mathcal{S}_{\mathbf{HAP}}$ and $\mathcal{S}_{\mathbf{HAP}'}$, of a society $\mathcal{S}$, $\mathcal{S}_{\mathbf{HAP}'}$ is a* proper extension *of $\mathcal{S}_{\mathbf{HAP}}$ if and only if $\mathbf{HAP} \subset \mathbf{HAP}'$.*

Definition 18.3. *Given an instance, $\mathcal{S}_{\mathbf{HAP}}$, of a society $\mathcal{S}$, the instance is* closed *iff it has no proper extensions. We denote a closed instance as $\mathcal{S}_{\overline{\mathbf{HAP}}}$.*

In the following, we indicate a closed history by means of an overline: $\overline{\mathbf{HAP}}$. Notice that in a closed instance, we assume that no further event might occur (i.e., the instance has no further extensions and the history is closed under CWA).

18.2 Declarative semantics: update

In the following we give semantics to a society instance by identifying sets of expectations which, together with the society's knowledge base and the happened events, imply an instance of the goal - if any - and *satisfy* the integrity constraints.

For notion of integrity constraint satisfaction we rely, in the following, upon a notion of entailment in a three-valued logic, since more general and capable of dealing with both open and closed society instances. Therefore, in the following, the symbol $\models$ has to be interpreted as the notion of entailment in a three-valued setting.

Furthermore, in this section, we consider negative literals of the kind $\neg\mathbf{H}()$ as new positive literals that have no definition in each open society instance. For closed society instances, we use Clark's completion of the history, $Comp(\mathbf{HAP})$, and negation is interpreted in the Closed World Assumption (CWA).

Throughout this section, for the sake of simplicity, we always consider a ground version of society's knowledge base and integrity constraints, and do not consider CLP-like constraints.

We first introduce the concept of *$\mathcal{IC}_S$-consistent set of social expectations* (previously called "admissible" in deliverable D5 [94]). Intuitively, given a society instance, a $\mathcal{IC}_S$-consistent set of social expectations consists of a set of expectations about social events that are compatible with P (i.e., the $SOKB$ and the set **HAP**), and with $\mathcal{IC}_S$.

Definition 18.4. ($\mathcal{IC}_S$-consistency) *Given a (closed/open) society instance $\mathcal{S}_{\mathbf{HAP}}$, an* $\mathcal{IC}_S$-consistent *set of social expectations Δ is a set of expectations such that:*

$$SOKB \cup \mathbf{HAP} \cup \Delta \models \mathcal{IC}_S \tag{10}$$

In definition 18.4 (and in the following definitions 18.7, 18.8, 18.9 and 18.10), for open instances we refer to a three-valued completion where only the history of events has not been completed. Therefore, for open instances,

$$SOKB \cup \mathbf{HAP} \cup \Delta \models \mathcal{IC}_S$$

is a shorthand for:

$$Comp(SOKB \cup \Delta) \cup \mathbf{HAP} \cup CET \models \mathcal{IC}_S$$

where $Comp()$ is three-valued completion [84] and CET Clark's equational theory.

For closed instances, instead,

$$SOKB \cup \overline{\mathbf{HAP}} \cup \Delta \models \mathcal{IC}_S$$

is a shorthand for:

$$Comp(SOKB \cup \Delta \cup \overline{\mathbf{HAP}}) \cup CET \models \mathcal{IC}_S$$

since also the history of events (closed) needs to be completed.

$\mathcal{IC}_S$-consistent sets of expectations can be self-contradictory (e.g., both $\mathbf{E}(p)$ and $\neg\mathbf{E}(p)$ may belong to a $\mathcal{IC}_S$-consistent set). In particular, among $\mathcal{IC}_S$-consistent sets of expectations, we are interested in those which are also consistent with respect to E-consistency (previously called "coherence" in D5) and ¬-consistency (previously called "consistency" in D5).

Definition 18.5. (E-consistency) *A set of social expectations* Δ *is* E-consistent *if and only if for each (ground) term* p*:*

$$\{\mathbf{E}(p), \mathbf{NE}(p)\} \not\subseteq \Delta$$

Definition 18.6. (¬-consistency) *A set of social expectations* Δ *is* ¬-consistent *if and only if for each (ground) term* p*:*

$$\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \Delta$$

and

$$\{\mathbf{NE}(p), \neg\mathbf{NE}(p)\} \not\subseteq \Delta$$

Given a closed (respectively, open) society instance, a set of expectations is called *closed* (resp. *open*) *admissible* if it satisfies Definitions 18.4, 18.5 and 18.6, i.e. if it is $\mathcal{IC}_S$-, E- and ¬-consistent.

Definition 18.7. (Fulfillment) *Given a (closed/open) society instance* $\mathcal{S}_{\mathbf{HAP}}$*, a set of social expectations* Δ *is* fulfilled *if and only if for each (ground) term* p*:*

$$\mathbf{HAP} \cup \Delta \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \nvdash \bot \tag{11}$$

Notice that Definition 18.7 above requires, for a closed instance of a society, that each positive expectation in Δ has a corresponding happened event in **HAP**, and each negative expectation in Δ has no corresponding happened event. This requirement is weaker for open instances, where a set Δ is not fulfilled only when a negative expectation occurs in the set, but the corresponding event happened (i.e., the implication $\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)$ is false).

Symmetrically, we define a violation:

Definition 18.8. (Violation) *Given a (closed/open) society instance $\mathcal{S}_{\mathbf{HAP}}$, a set of social expectations* **EXP** *is* violated *if and only if there exists a (ground) term p such that:*

$$\mathbf{HAP} \cup \Delta \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \models \bot \tag{12}$$

Finally, we give, in the following, the notion of goal achievability and achievement.

Definition 18.9. Goal achievability *Given an open instance of a society, $\mathcal{S}_{\mathbf{HAP}}$, and a ground goal G, we say that G is* achievable *(and we write $\mathcal{S}_{\mathbf{HAP}} \mid\!\approx_{\Delta} G$) iff there exists an (open) admissible and fulfilled set of social expectations Δ, such that:*

$$SOKB \cup \mathbf{HAP} \cup \Delta \models G \tag{13}$$

(which, as explained earlier, is a shorthand for $Comp(SOKB \cup \Delta) \cup \mathbf{HAP} \cup CET \models G$).

Definition 18.10. Goal achievement *Given a closed instance of a society, $\mathcal{S}_{\overline{\mathbf{HAP}}}$, and a ground goal G, we say that G is* achieved *(and we write $\mathcal{S}_{\overline{\mathbf{HAP}}} \vdash_{\Delta} G$) iff there exists a (closed) admissible and fulfilled set of social expectations Δ, such that:*

$$SOKB \cup \overline{\mathbf{HAP}} \cup \Delta \models G \tag{14}$$

(i.e., $Comp(SOKB \cup \overline{\mathbf{HAP}} \cup \Delta) \cup CET \models G$).

19 The society proof procedure

In the previous section, we have defined a society $\mathcal{S}$ as a triple $\langle SOKB, \mathcal{E}, \mathcal{IC}_S \rangle$ where

- $SOKB$ is the social organization knowledge base
- $\mathcal{E}$ is the set of abducible predicates (**E**, **NE** and their negation $\neg$**E**, $\neg$**NE**)
- $\mathcal{IC}_S$ is the set of social integrity constraints

Given a history **HAP** of events, we have defined an *instance* $\mathcal{S}_{\mathbf{HAP}}$ of a society $\mathcal{S}$ as the triple

$$\mathcal{S}_{\mathbf{HAP}} \equiv \langle SOKB \cup \mathbf{HAP}, \mathcal{E}, \mathcal{IC}_S \rangle.$$

The syntax of IC_S of the society model is strictly related to that of integrity constraints in the IFF proof procedure [47] enriched with constraints (C-IFF). This leads to the idea of using an extension of the C-IFF proof procedure for generating expectations on social behavior of members, and checking for their fulfillment or violation.

A first, obvious difference stands in the fact that our framework requires more dynamics. New facts (**H** events) continuously occur in the knowledge of the society, and must be taken into account by the proof procedure.

The proof procedure of the society, called $\mathcal{S}$CIFF (Society C-IFF) should have the following features [94]:

- it should accept new events as they happen
- it should produce sets of expectations

- it should detect fulfillment of expectations
- it should detect violations as soon as possible.

In this section we describe the proof procedure for the society as a transition system. In doing this, we draw inspiration from the IFF proof procedure [47], which we have briefly described in Section 6.1.3, and in particular the C-IFF extension, presented in Section 10.1, that also deals with constraints.

19.1 Data Structures

The $\mathcal{S}$CIFF proof procedure is based on a rewriting system transforming one node to another (or to others). A node can be either the special node *false*, or defined by the following tuple

$$T \equiv \langle R, CS, PSIC, \mathbf{EXP}, \mathbf{HAP}, \mathbf{FULF}, \mathbf{VIOL} \rangle$$

where

- R is a conjunction (initially set to the goal G), the conjuncts can be atoms or disjunctions (of conjunctions of atoms)
- CS is the constraint store
- $PSIC$ is the set of partially solved integrity constraints
- **EXP** is the set of (pending) expectations
- **HAP** is the history of happened events
- **FULF** is a set of fulfilled expectations
- **VIOL** is a set of violated expectations

If one of the elements of the tuple is *false*, then the whole tuple is the special node *false*, which cannot have successors.

We have seen that a society instance can be open or closed, depending on whether more events can happen or not, i.e., whether **HAP** is an open set or a closed set. We assume that we can rely on a predicate *closed*/1, which holds true if its argument represents a closed set.

19.1.1 Variable quantification

In the IFF proof procedure, the variable quantification is easily determined syntactically. There are three possible quantifications: a variable can be *free*, *existentially quantified* or *universally quantified*. The variables occurring in the initial goal are *free*; in our proof procedure we drop this distinction without loss of generality. In particular, in the IFF, a variable occurring in an abduced atom or in an atomic conjunct is existentially quantified (or free). Our proof procedure has to deal with universally quantified variables in the abducibles and in R (see also Table 1). Conversely, in the IFF proof procedure, variables in an implication are existentially quantified (or free) only if they also appear in an abducible (or, in general, in an atomic conjunct). In the society knowledge [94], we can have existentially quantified variables in the integrity constraints even if they do not occur elsewhere.

	IFF	$\mathcal{S}$CIFF
Variable in abduced atom	$\exists$ or *free*	$\exists$ or $\forall$
Variable occurring only in an implication	$\forall$	$\exists$ or $\forall$

Table 1: Variable quantification in the IFF and in $\mathcal{S}$CIFF

For all these reasons, we need to distinguish, among the variables, those that appear in abduced literals (or occur in R) and those that occur only integrity constraints. The variables in abduced literals and in the conjunction R have the whole tuple T as a scope. The others have as scope the implication in which they appear. We call the variables that appear in the conjunction R or in abduced literals *flagged*.

Definition 19.1. *All the variables that appear in R,* **EXP**, **FULF**, *and* **VIOL** *are* flagged. *Variables that are flagged in a node N are flagged in all the nodes descendant of N. Flagged variables have the entire node as scope.*

With respect to our terminology, the IFF proof procedure only has existentially, flagged variables (those occurring at least in an atomic conjunct) and universally, non flagged variables (appearing only in implications).

In the following, when we want to make explicit the fact that a variable X is flagged (when it is not clear from the context), it will be indicated with $\hat{X}$, while if we want to highlight that it is not flagged, it will be indicated with $\check{X}$.

Variables can be associated with *quantifier restrictions* [24]. Quantifier restrictions limit the applicability of a quantifier; in the case of universally quantified variables, they represent an implication, while in the case of an existentially quantified variable they are a conjunction. We restrict ourselves to quantifier restrictions that are *unary*, meaning that they involve only one variable.

Definition 19.2. *A* quantifier restriction *for a universally quantified variable X is a (unary) constraint $c(X)$ indicating the values that the variable X represents. If $QR(X)$ is the set of quantifier restrictions of X, then the following formula:*

$$\forall X_{QR(X)} F$$

holds iff the following holds:

$$\forall X. QR(X) \rightarrow F.$$

Note the difference between quantifier restrictions and CLP constraints. A constraint would mean that

$$\forall X. c(X) \wedge F$$

which is false if $c(X)$ is not satisfied for every possible X.

In the case of existentially quantified variables, quantifier restrictions coincide with the concept of CLP constraint.

Definition 19.3. *A* quantifier restriction *for an existentially quantified variable X is a (unary) constraint $c(X)$ indicating the values that the variable X represents. If $QR(X)$ is the set of quantifier restrictions of X, then the following formula:*

$$\exists X_{QR(X)} F$$

holds iff the following holds:

$$\exists X.QR(X) \wedge F.$$

In the tuple, the quantifier restrictions $QR(X)$ are recorded in the constraint store CS, and will be handled by the constraint solver.

19.1.2 Initial Node and Success

A derivation D is a sequence of nodes

$$T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_{n-1} \rightarrow T_n.$$

Given a goal G and a set of integrity constraints $\mathcal{IC}_S$, we build the first node in the following way:

$$T_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

i.e., the conjunction R is initially the query ($R_0 = \{G\}$) and the partially solved integrity constraints $PSIC$ is the set of integrity constraints ($PSIC_0 = \mathcal{IC}_S$).

The other nodes $T_j, j > 0$, are obtained by applying the transitions that we will define in the next section, until no further transition can be applied (we call this last condition *quiescence*).

Every arc in a derivation is labelled with the name of a transition.

Let us now give the definition of successful derivation, both in the case of an open society instance (where new events may be added to the history) and of a closed society instance.

Definition 19.4. *Starting with an open society instance $\mathcal{S}_{\mathbf{HAP}^i}$ there exists an* open successful derivation *for a goal G iff the proof tree with root node $\langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ has at least one leaf node*

$$\langle \emptyset, CS, PSIC, \mathbf{EXP}, \mathbf{HAP}^f, \mathbf{FULF}, \emptyset \rangle$$

where $\mathbf{HAP}^f \supseteq \mathbf{HAP}^i$ and CS is consistent (i.e., there exists a ground variable assignment such that all the constraints are satisfied). In that case, we write:

$$\mathcal{S}_{\mathbf{HAP}^i} \mid\!\sim^{\mathbf{HAP}^f}_{\mathbf{EXP} \cup \mathbf{FULF}} G$$

Definition 19.5. *Starting with a society instance $\mathcal{S}_{\mathbf{HAP}^i}$ there exists a* closed successful derivation *for a goal G iff the proof tree with root node $\langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ has at least one leaf node*

$$\langle \emptyset, CS, PSIC, \mathbf{EXP}, \overline{\mathbf{HAP}^f}, \mathbf{FULF}, \emptyset \rangle$$

where $\overline{\mathbf{HAP}^f} \supseteq \mathbf{HAP}^i$, CS is consistent, and $\mathbf{EXP}$ contains only negative literals $\neg\mathbf{E}$ and $\neg\mathbf{NE}$. In such a case, we write:

$$\mathcal{S}_{\mathbf{HAP}^i} \vdash^{\overline{\mathbf{HAP}^f}}_{\mathbf{EXP} \cup \mathbf{FULF}} G.$$

From a non-failure leaf node N, answers can be extracted in a very similar way to the IFF proof procedure. Answers of the $\mathcal{S}$CIFF proof procedure are called *expectation answers*. To compute an expectation answer, first, a substitution σ' is computed such that

- σ' replaces all variables in N that are not universally quantified by a ground term
- σ' satisfies all the constraints in the store CS_N.

If the constraint solver is (theory) complete [62] (i.e., for each set of constraints c, the solver always returns *true* or *false*, and never *unknown*), then there will always exist a substitution σ' for each non-failure leaf node N. Otherwise, if the solver is incomplete, σ' may not exist. The non-existence of σ' is discovered during the answer extraction phase. In such a case, the node N will be marked as a failure node, and another leaf node can be selected (if it exists).

Definition 19.6. *Let $\sigma = \sigma'|_{vars(G)}$ be the restriction of σ' to the variables occurring in the initial goal G. Let $\Delta = (\mathbf{FULF}_N \cup \mathbf{EXP}_N)\sigma'$. The pair (Δ, σ) is the* expectation answer *obtained from the node N.*

The society's proof procedure performs some inferences based on the semantics of time. We make the following hypothesis (that was already in document D5):

Definition 19.7. Hypothesis of full temporal knowledge. *We assume that all the socially significant events that have happened are known to the society at all times after occurrence.*

The proof procedure is based on the following transitions. They are repeated until quiescence.

19.2 Transitions

The transitions are based on those of the IFF proof procedure, augmented with those of CLP [61], and with specific transitions accommodating the concepts of fulfillment, dynamically growing history and consistency of the set of expectations with respect to the given definitions (Definitions 18.4, 18.5, and 18.6).

We first give the definition of *copy* of a formula.

Definition 19.8. *Given a formula F, we call* copy *of F a formula*

$$F' = copy(F)$$

where the universally quantified variables and the non flagged variables are renamed.

Notice that, by Definition 19.8, if F contains only flagged existentially quantified variables, then $copy(F) \equiv F$.

Notation In this section we adopt the following conventions. The letter k will indicate the level of a node, with 0 indicating the level of the initial node. Each transition will generate one or more nodes from level k to $k+1$. We will not explicitly report the new state for items that do not change; e.g., if a transition generates a new node from the node

$$T_k \equiv \langle R_k, CS_k, PSIC_k, \mathbf{EXP}_k, \mathbf{HAP}_k, \mathbf{FULF}_k, \mathbf{VIOL}_k \rangle$$

and we do not explicitly state the value of R_{k+1}, it means that $R_{k+1} = R_k$.

Let $vars(t)$ be the set of variables in the term t. Let $vars_\forall(t)$ (respectively, $vars_\exists(t)$) be the set of universally (resp. existentially) quantified variables in t and $vars_{flag}(t)$ be the set of flagged variables in t.

19.2.1 IFF-like transitions

Unfolding Since the variables in the head of a clause in the SOKB are all universally quantified with scope the entire clause, the unfolding step is basically the same as in many abductive proof procedures. It is defined as follows.

Let L_i be the selected literal in $R_k = L_1, \ldots, L_r$. Let L_i be a defined predicate. *Unfolding* replaces it with (one of) its definition. If $H_1 \leftarrow B_1, \ldots, H_n \leftarrow B_n$ are the clauses in the SOKB such that $H_1, \ldots, H_n$ unify with L_i, unfolding generates n nodes. In the j-th node:

- firstly, a renaming of the clause is obtained $H'_j \leftarrow B'_j$;
- then, all the variables in B'_j (that do not occur in the head) are flagged;
- the constraints of unification are added to the constraint store $CS_{k+1} = CS_k \cup \{H'_j = L_i\}$ (where $H'_j = L_i$ is a shorthand for the conjunction of equations between corresponding arguments of H'_j and L_i);
- B'_j is substituted for L_i in the new conjunction R, i.e., $R_{k+1} = L_1, \ldots, L_{i-1}, B'_j, L_{i+1}, \ldots, L_r$.

Moreover, as in the IFF proof procedure, unfolding can also be applied to atoms in the body of an implication. In that case, if

$$PSIC_k = \{Atom, BodyIC \rightarrow HeadIC\} \cup PSIC',$$

unfolding replaces (copies of) *Atom* with all its definitions; i.e., if the clauses $H_1 \leftarrow B_1, \ldots, H_n \leftarrow B_n$ belong to the SOKB and $H_1, \ldots, H_n$ unify with *Atom*,

$$\begin{aligned} PSIC_{k+1} = \quad & \{Atom^1 = H_1, B_1, BodyIC^1 \rightarrow HeadIC^1, \\ & \ldots, \\ & Atom^n = H_n, B_n, BodyIC^n \rightarrow HeadIC^n\} \cup PSIC' \end{aligned}$$

where, for all i, $Atom^i, BodyIC^i \rightarrow HeadIC^i$ is a copy of $Atom, BodyIC \rightarrow HeadIC$. The equalities in the body of implications will be dealt with by *Case Analysis*.

Abduction In the IFF the abduced atoms are kept with all other conjuncts in each node. In our approach, we have kept the set of expectations distinct from the other conjuncts in R, thus we need to introduce a transition of abduction. The *Abduction* step is defined as follows.

If $R_k = L_1, \ldots, L_r$, the selected literal L_i is of type **E**, **NE**, $\neg$**E**, $\neg$**NE**, then $R_{k+1} = L_1$, $\ldots$, L_{i-1}, L_{i+1}, $\ldots$, L_r and $\mathbf{EXP}_{k+1} \equiv \mathbf{EXP}_k \cup \{L_i\}$.

Propagation Let $L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j$ be an integrity constraint, belonging to the set $PSIC_k$, and let A be an atom,

- either belonging to $\mathbf{HAP}_k$ (in which case A is an **H** event),
- or an expectation belonging to $\mathbf{EXP}_k$, $\mathbf{FULF}_k$ or $\mathbf{VIOL}_k$,

such that A unifies with L_i. Then, by *Propagation*, we perform the following steps:

- we make a copy of A, $copy(A)$; this new atom is inserted in the same element of the tuple identifying the atom (e.g., if A is an **H** event, i.e. $A \in \mathbf{HAP}_k$, then $copy(A) \in \mathbf{HAP}_{k+1}$, and if $A \in \mathbf{EXP}_k$, then $copy(A) \in \mathbf{EXP}_{k+1}$). Notice that if A is an **H** event, it is ground, thus $copy(A) \equiv A$. Moreover, since **HAP**, **EXP**, **FULF** and **VIOL** are sets, duplicating one of their elements does not change them.
- $PSIC_{k+1} = PSIC_k \cup \{A = L'_i, L'_1, \ldots, L'_{i-1}, L'_{i+1} \ldots, L'_n \rightarrow H'_1 \vee \cdots \vee H'_j\}$, where $L'_1, \ldots, L'_i, \ldots, L'_n \rightarrow H'_1 \vee \cdots \vee H'_j = copy(L_1, \ldots, L_i, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j)$

Again, $A = L'_i$ is a shorthand for a conjunction of equalities between corresponding arguments.

This transition does not have any effect on the constraint store, since – as we will see soon – case analysis will take care of the equality in the body of the implication.

Splitting Given a node with

- $R_k = L_1, \ldots, L_{i-1}, (L^1_i \vee L^2_i), L_{i+1}, \ldots, L_r$

splitting produces two nodes, N^1 and N^2 such that in node N^1

- $R^1_{k+1} = L_1, \ldots, L_{i-1}, L^1_i, L_{i+1}, \ldots, L_r$

and in node N^2

- $R^2_{k+1} = L_1, \ldots, L_{i-1}, L^2_i, L_{i+1}, \ldots, L_r$

In the $\mathcal{S}$CIFF proof procedure, disjunctions may appear also in the constraint store (see rule 1 in Section 19.2.5). Disjunctions can be dealt with by the constraint solver itself (e.g., by means of constructive disjunction [116] or cardinality operator [115]), or by splitting. In the latter case, given a node with

- $CS_k = L_1, \ldots, L_{i-1}, (L^1_i \vee L^2_i), L_{i+1}, \ldots, L_r$

splitting produces two nodes, N^1 and N^2 such that in node N^1

- $CS^1_{k+1} = L_1, \ldots, L_{i-1}, L^1_i, L_{i+1}, \ldots, L_r$

and in node N^2

- $CS^2_{k+1} = L_1, \ldots, L_{i-1}, L^2_i, L_{i+1}, \ldots, L_r$

Case Analysis Given a node with an implication

$$PSIC_k = PSIC' \cup \{A = B, L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$$

the node is replaced by two nodes, as follows:
Node 1:

- $PSIC^1_{k+1} = PSIC' \cup \{L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$
- $CS^1_{k+1} = CS_k \cup \{A = B\}$

Node 2:

- $PSIC^2_{k+1} = PSIC'$

- $CS^2_{k+1} = CS_k \cup \{A \neq B\}$

where $\neq$ stands for the constraint of non-unification.

Since our proof procedure also needs to deal with (CLP) constraints in the body of implications, we also extend case analysis to the following situation (as in the C-IFF):
Given a node with an implication

$$PSIC_k = PSIC' \cup \{c, L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$$

where c is a constraint, if all the variables in $vars(c)$ are flagged, then case analysis can be applied. We distinguish three cases.

1. If all the variables in $vars(c)$ are existentially quantified, then case analysis generates two nodes.

   Node 1:

   - $PSIC^1_{k+1} = PSIC' \cup \{L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$
   - $CS^1_{k+1} = CS_k \cup \{c\}$

   Node 2:

   - $PSIC^2_{k+1} = PSIC'$
   - $CS^2_{k+1} = CS_k \cup \{\neg c\}$

2. If all the variables in $vars(c)$ are universally quantified, then only one node is generated, in which

   - $CS_{k+1} = CS_k \cup \{\forall_{X \in vars(c):c}\}$
   - $PSIC_{k+1} = PSIC' \cup \{L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$

   Example 8. *For example, if we have:*[27]

   $$\begin{array}{rl} \forall_{\hat{X}}, & \mathbf{NE}(p(\check{Y})), \check{Y} > 0 \rightarrow \mathbf{NE}(q(\check{Y})) \\ & \mathbf{EXP}_k = \{\mathbf{NE}(p(\hat{X}))\} \end{array}$$

   we can apply Propagation *and obtain (after applying* Case Analysis *to the resulting equality in the body)*

   $$\begin{array}{rl} \forall_{\hat{X}}, \forall_{\hat{X}'} & \mathbf{NE}(p(\check{Y})), \check{Y} > 0 \rightarrow \mathbf{NE}(q(\check{Y})) \\ & \mathbf{EXP}_{k+2} = \{\mathbf{NE}(p(\hat{X})), \mathbf{NE}(p(\hat{X}'))\} \\ & \hat{X}' > 0 \rightarrow \mathbf{NE}(q(\hat{X}')) \end{array}$$

   Now we can apply Case Analysis *to handle the constraint $\hat{X}' > 0$ in the body of an implication, and obtain:*

   $$\begin{array}{rl} \forall_{\hat{X}}, \forall_{\hat{X}'>0} & \mathbf{NE}(p(\check{Y})), \check{Y} > 0 \rightarrow \mathbf{NE}(q(\check{Y})) \\ & \mathbf{EXP}_{k+3} = \{\mathbf{NE}(p(\hat{X})), \mathbf{NE}(p(\hat{X}'))\} \\ & \mathit{true} \rightarrow \mathbf{NE}(q(\hat{X}')) \end{array}$$

   from which we will reach (through futher transitions, see Logical Equivalence*) a node with the intuitive meaning: "p(X) is expected not to happen for all X, and q(X′) is expected not to happen for all X′ > 0".*

[27] Remember that we use $\hat{A}$ to denote flagged variables, and $\check{A}$ to denote non-flagged ones. We show explicitly that X is universally quantified and flagged, even if it should be clear from the context.

3. If $vars(c)$ contain both universally and existentially quantified variables, case analysis will not be applied. However, we have assumed that the society knowledge is Constraint Allowed (Definition 17.5), so we do not need to deal with this case.

Factoring In the IFF proof procedure, transition factoring distinguishes answers in which abducible atoms are merged from answers in which they are distinct. That is, it generates two nodes: in one node two hypotheses unify, in the other one a constraint is imposed in order to avoid the unification of the hypotheses.

In the $\mathcal{S}$CIFF proof procedure, abducibles can contain universally quantified variables; it is not reasonable to unify atoms with universally quantified variables, because we would lose some of the information given by the abduced atoms. Another possibility would be to copy the atoms before trying to join them, but we would only obtain redundant information.

Example 9. *Suppose that the set of expectations, in a node N_k, is the following:*

$$\forall \hat{X}, \hat{Y}\ \mathbf{EXP}_k = \{\mathbf{NE}(p(1, \hat{X})), \mathbf{NE}(p(\hat{Y}, 2))\}.$$

By unifying the two hypotheses, we would obtain

$$\mathbf{EXP}_{k+1} = \{\mathbf{NE}(p(1, 2))\}$$

which has a different meaning from the union of the two previous hypotheses: in $\mathbf{EXP}_{k+1}$, $p(1, 7)$, for example, is no longer expected not to happen.

The second possibility would be to perform a copy of the two abduced atoms before unifying them:

$$\forall \hat{X}, \hat{Y}\ \mathbf{EXP}_{k+1} = \{\mathbf{NE}(p(1, \hat{X})), \mathbf{NE}(p(\hat{Y}, 2)), \mathbf{NE}(p(1, 2))\}$$

but this is not very informative.

For this reason, we apply *factoring* only if all the variables in the two atoms are existentially quantified. Notice that this coincides with the factoring transition of the IFF proof procedure.

Formally, *factoring* can be applied in a node N_k, in which:

- $\mathbf{EXP}_k \cup \mathbf{FULF}_k \cup \mathbf{VIOL}_k \supseteq \{A_1, A_2\}$

where A_1 and A_2 are (unifiable abducible) atoms in which all the variables are existentially quantified (and, of course, flagged). Factoring generates two children nodes; N^1 is the following:

- $CS^1_{k+1} = CS_k \cup \{A_1 = A_2\}$

and N^2 is:

- $CS^1_{k+1} = CS_k \cup \{A_1 \neq A_2\}$

Equivalence Rewriting The equivalence rewriting operations are delegated to the constraint solver. Note that a constraint solver works on a constraint domain which has an associated interpretation. In addition, the constraint solver should handle the constraints among terms derived from the unification. Therefore, beside the specific constraint propagation on the constraint domain, we need further inference rules for coping with the unification. In other words we will assume that:

- the constraint theory contains rules for the equality constraint
- the constraint solver contains the same rules that are in the IFF proof procedure, i.e., the function $infer(CS)$ (see Section 19.2.5) performs the following substitutions in the constraint store:
  1. Replaces $f(t_1, \ldots, t_j) = f(s_1, \ldots, s_j)$ with $t_1 = s_1 \wedge \cdots \wedge t_j = s_j$.
  2. Replaces $f(t_1, \ldots, t_j) = g(s_1, \ldots, s_l)$ with $false$ whenever f and g are distinct or $j \neq l$.
  3. Replaces $t = t$ with $true$ for every term t.
  4. Replaces $X = t$ by $false$ whenever t is a term containing X.
  5. (a) Replaces $t = X$ with $X = t$ if X is a variable and t is not
     (b) Replaces $Y = X$ with $X = Y$ whenever X is a universally quantified variable and Y is not.
  6. If $X = t \in CS_k$, applies the substitution X/t to the entire node.

Note that the symbol = is overloaded since it is used both for representing the equality constraint in the constraint domain and for representing the unification among terms. As it is usual in CLP [62], we distinguish between the two by means of types. Some predicates, the so called constraints, do not have a definition in the program, but their semantics is embedded in the constraint solver. Also, some of the functor symbols are not simply Herbrand terms, but are associated with symbols in the constraint domain (e.g., the symbol + represents the addition, etc.), and variables can be associated with domains. In this way, the terms that are built from CLP functors and variables can be distinguished, and the correct = can be applied.

Moreover, we also have to consider that our language is more expressive than that of the IFF proof procedure, as we can abduce atoms with universally quantified variables. For this reason, we introduced *flagged* variables, and we need to deal with them in the theory of unification. We therefore add the following rules:

5. (c) If $\check{X} = t \in CS_k$, $\check{X}$ is and universally quantified in the body of an implication and is not flagged, then replace $\check{X} = t$ with *false*.

5. (d) If $X = \check{Y} \in CS_k$, $\check{Y}$ is and universally quantified in the body of an implication and is not flagged, and X has some (non trivially true) quantifier restrictions, then replace $X = \check{Y}$ with *false*.

Thus, a variable which is universally quantified with scope the body of a IC_S, unifies only with universally quantified variables that do not have quantifier restrictions. For example, consider the following IC_S:

$$\mathbf{NE}(p(\check{X})) \rightarrow \mathbf{NE}(q(\check{Y}))$$

which means [94]:

$$[\forall_{\check{X}} \mathbf{NE}(p(\check{X}))] \rightarrow \exists_{\check{Y}} \mathbf{E}(q(\check{Y})).$$

Variable $\check{X}$ may be unified (through Propagation and Case Analysis) to an abduced atom $\forall_{\hat{Z}} \mathbf{NE}(p(\hat{Z}))$. Note that it would not be correct to Propagate the IC_S with an atom $\mathbf{NE}(p(a))$ or with $\exists_{\hat{A}} \mathbf{NE}(p(\hat{A}))$: thus, only universally quantified variables unify with a non-flagged variable universally quantified in the body of a IC_S.

The IFF proof procedure also contains the following rule (called 6b in [47]):

If $X = t$ occurs in the condition of an implication, X does not occur in t and X is universally quantified, then apply the substitution X/t to the implication deleting the equality

In our proof procedure, this rule is not necessary, as it is dealt with by *case analysis*. Case analysis will create two nodes, one of which will perform the unification, and the other will trivially fail.

Logical Equivalence The rule

"$true \rightarrow A$ is equivalent to A"

of the IFF proof procedure is translated as follows. If $PSIC_k = PSIC' \cup \{true \rightarrow A\}$, we generate a new node such that:

- $PSIC_{k+1} = PSIC'$
- $R_{k+1} = R_k, A'$

where A' is obtained from A by flagging all the variables that were not already flagged.

We also have the following rules, as in the IFF proof procedure:

$$\begin{array}{rcl} A \wedge false & \leftrightarrow & false \\ A \vee false & \leftrightarrow & A \\ A \wedge true & \leftrightarrow & A \\ A \vee true & \leftrightarrow & true \\ false \rightarrow A & \leftrightarrow & true \\ \neg A & \leftrightarrow & A \rightarrow false \\ \neg A, B \rightarrow C & \leftrightarrow & B \rightarrow A \vee C \end{array}$$

The last two rules are used only when applied to negation of definite predicates; the implications are inserted in the set $PSIC_{k+1}$. The negation of atoms **E** or **NE** are dealt with through abduction, i.e., for a negative literal $\neg$**E** we have an abducible (positive) literal *non***E** that cannot be true together with the corresponding literal **E** (see rules in Section 19.2.4), and symmetrically, the negative literal $\neg$**NE** is represented by an abducible *non***NE**. Concerning the negation of atoms **H**, they are dealt with by a transition non-Happening (Section 19.2.2).

19.2.2 Dynamically growing history

A set of transitions deals with a dynamically growing history **HAP**. The transitions deal with the happening (or non-happening) of events.

Happening of Events The happening of events is considered by a transition *Happening*. We assume that the happened events are recorded in an external set (e.g., a buffer that links the proof procedure to the external world). Transition Happening takes an event **H**(*Event*) from the external set and puts it in the history **HAP**. The transition *Happening* is applicable only if an *Event* such that **H**(*Event*) $\notin$ **HAP** is in the external set.

Given a node N_k in which the history is closed[28]

[28] We assumed in Section 19.1 that there is a *closed* predicate that is true if the history is closed.

- $closed(\mathbf{HAP}_k) = false$

the transition *Happening* produces a single successor

$$\mathbf{HAP}_{k+1} = \mathbf{HAP}_k \cup \{\mathbf{H}(Event)\}.$$

Otherwise, given a node in which

- $closed(\overline{\mathbf{HAP}_k}) = true$

the transition *Happening* produces a single successor

$$false$$

Note that transition *happening* should be applied to all the non-failure nodes (in the frontier).

Non-happening Considering $\neg\mathbf{H}$ literals in the body of a PSIC, we apply a transition *Non-happening*, that can be considered as an application of *constructive negation*. Constructive negation is preferred to Negation as Failure since this latter one only allows ground negative goals to execute. This would be too constraining in our case, where some variables appearing in a derivation never become ground. In [112], Stuckey has described a scheme for constructive negation in constraint logic programming, so to obtain a sound and complete operational model for negation in this class of languages. Constructive negation was first formulated for logic programming in the Herbrand Universe and involves introducing disequality constraints. Constraint logic programming thus provides a much more natural framework for describing constructive negation, as proposed in [112], where constructive negation for constraing logic programming is considered over arbitrary structures, and proved sound and complete with respect to the three-valued consequences of the completion of a program.

Therefore, constructive negation is a powerful inference that is particularly well suited in CLP [112]; since our proof procedure deals with constraints, it is reasonable to exploit the same inferences.

Rule *non-happening* applies when a literal $\neg\mathbf{H}$ is in the body of a PSIC, and the history is closed:

$$\neg\mathbf{H}(E_1), L_2, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m. \tag{15}$$

Given a node:

- $PSIC_k = \{\neg\mathbf{H}(E_1), L_2, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m\} \cup PSIC'$
- $closed(\overline{\mathbf{HAP}}) = true$

non-happening produces a new node. Intuitively, we hypothesize that all the events matching E_1 that are not in the history, do not happen at all.

The child node is produced as follows; we first give the intuition, then formalize the definition. We hypothesize that every event that would be able to match with E_1, and is not in the current history, will not happen. This can be seen as abducing an atom $non\mathbf{H}(E_1')$. E_1' is E_1 with all variables replaced with universally quantified variables. We impose that the hypothesis holds in all cases except those already in the $\overline{\mathbf{HAP}}$; we can state this by means of the quantifier restrictions, i.e., we impose that the hypothesis $non\mathbf{H}(E_1')$ does not unify with any of the happened events. This is equivalent to imposing a conjunction (for all the events

in the history that match E_1') of a disjunction (for all the variables appearing in E_1') of non unification restrictions (written $\neq$).

Notice, however, that the event E_1 in the PSIC (15) may contain existentially quantified, non flagged variables. Indicating by $\check{X}$ such variables, the meaning of PSIC (15) is [94]:

$$[\forall_{\check{X}} \neg \mathbf{H}(E_1), L_2, \ldots, L_n] \rightarrow H_1 \vee \cdots \vee H_m.$$

Suppose that there is an event E in the history that matches E_1 through some substitution θ (i.e., $E_1\theta = E$). If the history contains another event E' that is identical except for variable $\check{X}$ (i.e., E' unifies with $E_1\theta'$, where $\theta' = \theta|_{vars(E_1)\setminus\{\check{X}\}}$), E' and E will have the same effect, concerning propagation of the PSIC in question. For example, if

$$\neg \mathbf{H}(p(\check{X}, \check{Y})) \rightarrow \mathbf{NE}(q(\check{Y}))$$

and the history $\mathbf{HAP} = \{\mathbf{H}(p(1,1)), \mathbf{H}(p(2,1))\}$, the result should be $\forall_{\hat{Y}' \neq 1} \mathbf{NE}(q(\hat{Y}'))$. We would have obtained the same result if the history contained only *one* of the two events.

For this reason, there is no point in imposing the quantifier restrictions also on the variables that are renaming of existentially, non flagged, variables. This issue will be clarified with an example in the following.

We now give a formal definition. Let E_1' be a renaming of E_1 (i.e., all the variables in E_1 are substituted with fresh new variables). Let all the new variables in E_1' be universally quantified and flagged. For each variable $X_j \in vars(E_1)$, let $ren(X_j)$ be the corresponding, renamed variable in $vars(E_1')$. For all atoms $\mathbf{H}(E) \in \overline{\mathbf{HAP}}$ that unify with $\mathbf{H}(E_1')$, we impose the quantifier restrictions on the variables in E_1' given by the following disjunction:

$$\bigwedge_{\substack{\mathbf{H}(E) \in \overline{\mathbf{HAP}} \\ s.t. unifies(E, E_1')}} \left(\bigvee_{X_j \in vars_\forall(E_1) \cup vars_{flag}(E_1)} ren(X_j) \neq t_j \right)$$

where t_j is the term in E corresponding to X_j in E_1.

The child node, $k+1$, is then defined by:

- $PSIC_{k+1} = \{E_1 = E_1', L_2, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m\} \cup PSIC'$

Example 10. *From D5:*

$$\neg \mathbf{H}(tell(\check{A}, \check{B}, propose(\check{I})), \check{T}_p) \rightarrow \mathbf{NE}(tell(\check{B}, \check{A}, accept(\check{I})), \check{T}_a) \tag{16}$$

Remember [94] that variable $\check{T}_p$ is universally quantified with scope the body, or, equivalently, existentially quantified with scope the whole IC_S.

Suppose that the history contains $\mathbf{H}(tell(yves, thomas, propose(nail)), 1)$. *The condition in the body of the IC_S (16) is true, thus the IC_S triggers and the head is evaluated.*

$$\begin{array}{c} \mathbf{HAP}_k = \{\mathbf{H}(tell(yves, thomas, propose(nail)), 1)\} \\ | \\ \forall_{\hat{A}' \neq yves \vee \hat{B}' \neq thomas \vee \hat{I}' \neq nail} \ \mathbf{EXP}_{k+1} = \{\mathbf{NE}(tell(\hat{B}', \hat{A}', accept(\hat{I}')), \hat{T}_a)\} \end{array}$$

Notice that we did not impose a quantifier restriction on variable $\hat{T}'_p$, that is, we do not impose the quantifier restriction $\forall_{\hat{A}' \neq yves \vee \hat{B}' \neq thomas \vee \hat{I}' \neq nail \vee \hat{T}'_p \neq 1}$. Variable $\hat{T}'_p$ is a renaming of an existentially quantified, non-flagged variable. The fact that we do not impose restrictions on $\hat{T}'_p$ is sound, because we are making a more restrictive hypothesis: we are supposing that nobody will propose anything to anybody in any time, except Yves that proposes a nail to Thomas at some time*. This is more restrictive than hypothesizing that even Yves will not propose a nail to Thomas at another time, different from 1.*

If we imposed the quantifier restriction also on $\hat{T}'_p$ we would not have the desired behavior. Suppose that Thomas accepts the nail offered by Yves: $\mathbf{H}(tell(thomas, yves, accept(nail)), 3)$. Of course, this is perfectly acceptable and should not raise violations. But, if we check for a possible violation of the imposed **NE** *(see transition Violation* **NE***), we will try the unification: $\hat{B}' = thomas \wedge \hat{A}' = yves \wedge \hat{I}' = nail \wedge \hat{T}_a = 3$ hoping that the derivation will fail (in order not to detect a wrong violation). However, this unification does not clash with the quantifier restrictions: $\hat{A}' \neq yves \vee \hat{B}' \neq thomas \vee \hat{I}' \neq nail \vee \hat{T}'_p \neq 1$. In fact, it is enough that $\hat{T}'_p$ is not unified with* 1*; thus we would wrongly detect a violation.*

Closure This transition enforces the closure of the history. It nondeterministically imposes that no more events will happen.

Transition *Closure* is not applicable until there is no other transition applicable (and $N_k \neq false$). In other words, it is only applicable at the quiescence of the set of the other transitions.

Given a node:

- $closed(\mathbf{HAP}_k) = false$

in which no other transition is applicable, transition *Closure* produces two nodes. Node N^1 is the following:

- $\mathbf{HAP}_{k+1} = \overline{\mathbf{HAP}_k}$

and node N^2 is identical to its parent. In order to avoid infinite loops, transition *Closure* cannot be again applied to the node N^2 before a Happening transition has been applied.

19.2.3 Fulfillment and Violation

Violation NE Given a node N_k as follows:

- $\mathbf{EXP}_k = \mathbf{EXP'} \cup \{\mathbf{NE}(E_1)\}$
- $\mathbf{HAP}_k = \mathbf{HAP'} \cup \{\mathbf{H}(E_2)\}$

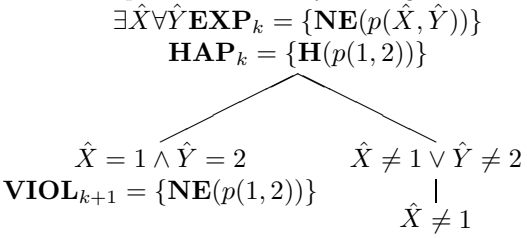
violation **NE** produces two nodes N^1_{k+1} and N^2_{k+1}, where N^1_{k+1} is as follows:

- $\mathbf{VIOL}_{k+1} = \mathbf{VIOL}_k \cup \{\mathbf{NE}(E_1)\}$
- $CS_{k+1} = CS_k \cup \{E_1 = E_2\}$

and N^2_{k+1} is as follows:

- $\mathbf{VIOL}_{k+1} = \mathbf{VIOL}_k$
- $CS_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Example 11. *Suppose that* $\mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\}$ *and* $\exists\hat{X}\forall\hat{Y}\mathbf{EXP}_k = \{\mathbf{NE}(p(\hat{X},\hat{Y}))\}$. Violation **NE** *will produce the two following nodes:*



where the last simplification in the right branch is due to the rules of the constraint solver (see Section 19.2.5).

Fulfillment E Starting from a node N_k as follows:

- $\mathbf{EXP}_k = \mathbf{EXP}' \cup \{\mathbf{E}(Event_1)\}$
- $\mathbf{HAP}_k = \mathbf{HAP}' \cup \{\mathbf{H}(Event_2)\}$

Fulfillment **E** builds two nodes, N^1_{k+1} and N^2_{k+1}, that are identical to their parent except for the following.

In node N^1_{k+1} we assume that the expectation and the happened event unify:

- $\mathbf{EXP}_{k+1} = \mathbf{EXP}'$
- $\mathbf{FULF}_{k+1} = \mathbf{FULF}_k \cup \{\mathbf{E}(Event_1)\}$
- $CS_{k+1} = CS_k \cup \{Event_1 = Event_2\}$

In node N^2_{k+1} we assume that the two will not unify:

- $\mathbf{EXP}_{k+1} = \mathbf{EXP}_k$
- $\mathbf{FULF}_{k+1} = \mathbf{FULF}_k$
- $CS_{k+1} = CS_k \cup \{Event_1 \neq Event_2\}$

Violation E Given a node N_k such that:

- $closed(\overline{\mathbf{HAP}_k}) = true$
- $\mathbf{EXP}_k = \mathbf{EXP}' \cup \{\mathbf{E}(Event_1)\}$
- $\forall Event_2 : \mathbf{H}(Event_2) \in \mathbf{HAP}_k$, $Event_2$ does not unify with $Event_1$

transition *Violation* **E** creates a successor node in which

- $\mathbf{VIOL}_{k+1} = \mathbf{VIOL}_k \cup \{\mathbf{E}(Event_1)\}$.
- $\mathbf{EXP}_{k+1} = \mathbf{EXP}'$

With *does not unify* we mean here that unification leads to failure; in CLP this means that trying to unify the two terms triggers CLP propagation of constraints, and this propagation gives a failure. For example, we mean that $p(\hat{X})$ does not unify with $p(10)$ if the constraint store contains $\hat{X} < 5$. Stated otherwise,

$$\forall_{\mathbf{H}(Event_2)\in\mathbf{HAP}_k}, CS_k \cup \{Event_2 = Event_1\} \models \bot.$$

The condition *"$Event_1$ should not unify with any event in the history"* seems rather expensive to check. Operationally, this expensive check can often be avoided. For example, the $\mathcal{S}$CIFF proof procedure can be used both for *on-line* or for *a posteriori* check of compliance. In the *on-line* check, the starting history is often empty, then some events will happen, transition *closure* will be eventually applied, and other transitions will be finally applied. A typical derivation will have the following structure:

$$N_0 \longrightarrow \cdots \overset{closure}{\longrightarrow} N_c \longrightarrow \ldots \longrightarrow N_f$$

Transition *Violation* **E** can be applied only when the history is closed, thus after N_c. In this situation, the check is redundant: in fact, since transition *closure* can be applied only if no other transition is applicable, the transition *Fulfillment* **E** has been already tried for each event in the history *before* node N_c, and has moved all the fulfilled expectations in the set **FULF**. The expectations of type **E** remaining in the set **EXP** are, thus, violated.

Other types of closure can be done; e.g., by considering the semantics of *time*. Even in an open history, if we make the hypothesis of full temporal knowledge (Definition 19.7), we can infer that an expected event for which the deadline is passed, raises a violation.

Given a node N_k:

- $\mathbf{EXP}_k = \{\mathbf{E}(X,T)\} \cup \mathbf{EXP'}$
- $\mathbf{HAP}_k = \{\mathbf{H}(Y,T_c)\} \cup \mathbf{HAP'}$
- $\forall Event_2, T_2 : \mathbf{H}(Event_2, T_2) \in \mathbf{HAP}$, $(Event_2, T_2)$ does not unify with (X,T)
- $closed(\mathbf{HAP}_k) = false$
- $CS_k \models T < T_c$

transition Violation **E** is applicable and creates the following node:

- $\mathbf{EXP}_{k+1} = \mathbf{EXP'}$
- $\mathbf{VIOL}_{k+1} = \mathbf{VIOL}_k \cup \{\mathbf{E}(X,T)\}$.

In this case, as for the case of the closed history, one can avoid the expensive check of unification with all the elements in the history by choosing a preferred order of application of the transitions. By applying *Violation* **E** only if no other transition is applicable (except, eventually, for *closure*), the check of expectations against the history can be safely avoided.

Notice that this transition infers the current time from any happened event; i.e., it infers that the current time cannot be less than the time of a happened event. In particular, there can be an event *current_time* that happens at every time tick. Of course, it is not necessary to perform the check for every event in the history; checking the last element in the history is enough.

A brief discussion is helpful here for the entailment $CS_k \models T < T_c$. The entailment of constraints from a constraint store is, in general, not easy to verify. In this particular case, however, we have that:

- the constraint $T < T_c$ is unary (T_c is always ground), thus a CLP(FD) (CLP for finite domains) solver is able to verify the entailment very easily if the store contains only unary constraints (it is enough to check the maximum value in the domain of T);
- even if the store contains non-unary constraints (thus the solver performs, in general, incomplete propagation), the transition will not compromise the soundness and completeness of the proof procedure. If the solver performs a powerful propagation (including pruning, in CLP(FD)), the violation will be early detected. If the solver does not perform propagation at all, the violation will be detected very late, when the history of the society gets closed.

Fulfillment NE Given a node N_k where

- $closed(\overline{\mathbf{HAP}_k}) = true$
- $\mathbf{EXP}_k = \mathbf{EXP}' \cup \{\mathbf{NE}(Event_1)\}$
- $\forall Event_2 : \mathbf{H}(Event_2) \in \mathbf{HAP}$, $Event_2$ does not unify with $Event_1$

transition *Fulfillment* **NE** creates a successor node in which

- $\mathbf{FULF}_{k+1} = \mathbf{FULF}_k \cup \{\mathbf{NE}(Event_1)\}$
- $\mathbf{EXP}_{k+1} = \mathbf{EXP}'$.

As for transition Violation **E**, the check of unification with all the atoms in the history can be avoided by establishing a preferred order of application of the transitions.

19.2.4 Consistency

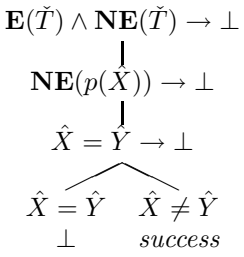
E-Consistency In order to ensure **E**-consistency of the set of expectations, we impose the following integrity constraint:

$$\mathbf{E}(\check{T}) \wedge \mathbf{NE}(\check{T}) \rightarrow \bot \tag{17}$$

Example 12. *Suppose that* $(\exists \hat{X})\mathbf{E}(p(\hat{X}))$ *and* $(\forall \hat{Y})\mathbf{NE}(p(\hat{Y}))$ *have been abduced. By triggering the integrity constraint (17) we have that:*

$$\begin{array}{c} \mathbf{E}(\check{T}) \wedge \mathbf{NE}(\check{T}) \rightarrow \bot \\ \downarrow \\ \mathbf{NE}(p(\hat{X})) \rightarrow \bot \\ \downarrow \\ \hat{X} = Y \rightarrow \bot \\ \downarrow \\ \bot \end{array}$$

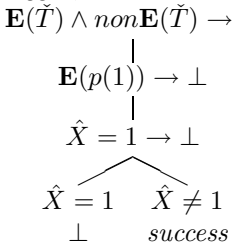
Example 13. *Suppose that* $(\exists \hat{X})\mathbf{E}(p(\hat{X}))$ *and* $(\exists \hat{Y})\mathbf{NE}(p(\hat{Y}))$ *have been abduced. By triggering the integrity constraint (17) we have that:*



¬-Consistency In order to ensure ¬-consistency of the set of expectations, we impose the following integrity constraints:

$$\begin{array}{lcll} \mathbf{E}(\check{T}) & \wedge & non\mathbf{E}(\check{T}) & \rightarrow \bot \\ \mathbf{NE}(\check{T}) & \wedge & non\mathbf{NE}(\check{T}) & \rightarrow \bot \end{array} \tag{18}$$

Example 14. *Suppose that* $\mathbf{EXP}_k = \{\mathbf{E}(p(\hat{X})), non\mathbf{E}(p(1))\}$. *The integrity constraint (18) can trigger, and we have that:*



19.2.5 Constraint Solving

As in the C-IFF, the Society proof procedure has transitions of *constraint solving*. In the C-IFF, CLP constraints are kept with other conjuncts in a node. In our approach, we have kept the set of CLP constraints distinct from the other atoms, in a Constraint Store, as in the definition by Jaffar and Maher [61]. For this reason, we use the same transitions given in CLP [61] (namely, *Constrain, Infer* and *Consistent*), while the C-IFF does not need to explicitly give them.

In the terminology of Constraint Logic Programming [61], we assume that the symbols $=$ and $\neq$ are in the constraint language and the theory underlining them is, for equality, the one described in Section 19.2.1. Concerning $\neq$, we will again assume that it is possible to syntactically distinguish the CLP-interpreted terms and atoms; the solver will perform some inference on the interpreted terms (typically, depending on the CLP sort), and will, moreover, contain the following rules, for uninterpreted terms:

1. Replace $f(t_1, \ldots, t_j) \neq f(s_1, \ldots, s_j)$ with $t_1 \neq s_1 \vee \cdots \vee t_j \neq s_j$.
2. Replace $f(t_1, \ldots, t_j) \neq g(s_1, \ldots, s_l)$ with *true* whenever f and g are distinct or $j \neq l$.
3. Replace $t \neq t$ with $false$ for every term t.
4. Replace $X \neq t$ by *true* whenever t is a term containing X.
5. (a) Replace $t \neq X$ with $X \neq t$ if X is a variable and t is not

(b) Replace $Y \neq X$ with $X \neq Y$ whenever X is a universally quantified variable and Y is not.

6. (a) Replace $A \neq B$ with $false$ if A is a universally quantified variable without quantifier restrictions (i.e., $QR(A) = \emptyset$)

   (b) If A is a universally quantified variable with quantifier restrictions $QR(A) = \{c_1(A), \ldots, c_d(A)\}$, and B is not universally quantified, replace $A \neq B$ with $\neg c_1(B) \vee \cdots \vee \neg c_d(B)$.[29]

   (c) If A and B are universally quantified, with quantifier restrictions $QR(A)$ and $QR(B)$ then

   - if $\neg QR(A) \cap \neg QR(B) = \emptyset$, replace $A \neq B$ with $true$.[30]
   - otherwise, replace $A \neq B$ with false.

Notice that the rules 1-6 do not deal with all possible combinations of quantifiers (e.g., $\exists A, B.A \neq B$). For those not dealt with, we appeal to the underlying constraint solver. Some solvers can easily propagate constraints of this type. E.g., given $X \neq 1$ a Finite Domain solver can delete the value 1 from the domain of X. In the case of $X \neq t$, where t is not ground, the constraint is typically suspended (thus we do not have a transition). We will delay the $\neq$ constraint until it can be processed by the rules 1-6 above.

The constraint solver deals also with quantifier restrictions. If a quantifier restriction (due to unification) gets all the variables existentially quantified, then we replace it with the corresponding constraint. For example, if in a node we have:

$$\exists \hat{Y}, \forall_{\hat{X} \neq 1}, \hat{X} = \hat{Y}$$

we obtain that $\exists \hat{Y}, \hat{Y} \neq 1$ (the quantifier restriction $\hat{X} \neq 1$ becomes a constraint on the variable $\hat{Y}$).

Example 15. *Consider the IC_S*

$$\mathbf{NE}(p(\check{X})), \mathbf{E}(q(\check{X})) \rightarrow \mathbf{NE}(r(\check{X}))$$

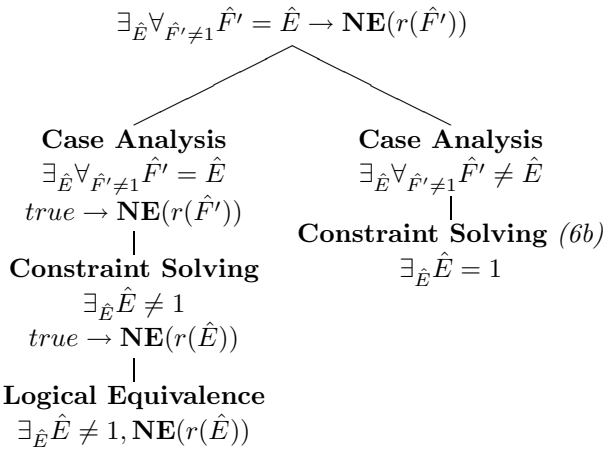
and suppose that $\exists_{\hat{E}} \forall_{\hat{F} \neq 1} \mathbf{EXP}_k = \{\mathbf{NE}(p(\hat{F})), \mathbf{E}(q(E))\}$. By applying Propagation of the IC_S with $\mathbf{NE}(p(\hat{F}))$, we first copy the abduced atom (i.e., $\mathbf{NE}(p(\hat{F}'))$), then we obtain (after Case Analysis):

$$\forall_{\hat{F}' \neq 1} \mathbf{E}(q(\hat{F}')) \rightarrow \mathbf{NE}(r(\hat{F}'))$$

Now we can Propagate this PSIC with the atom $\mathbf{E}(q(\hat{E}))$, and obtain:

[29] Intuitively, A is universally quantified, thus it assumes every possible value except the ones forbidden by one of the c_i. Thus, the only way to satisfy this constraint is to impose that B assumes one of the values excluded for A.

[30] Intuitively, if the values taken by A have no intersection with the values taken by B, then $A \neq B$ is true.



Constrain Given a node with

- $R_k = L_1, \ldots, L_r$

and the selected literal, L_i is a constraint, *constrain* produces a node with

- $R_{k+1} = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_r$
- $CS_{k+1} = CS_k \cup \{L_i\}$

Infer Given a node, the transition *Infer* modifies the constraint store by means of a function *infer(CS)*. This function is typical of the adopted constraint sort. E.g., the function *infer* in a FD (Finite Domain) sort will typically compute (generalized) arc-consistency, while for equality and disequality, will use the rules given earlier.

- $CS_{k+1} = \mathit{infer}(CS_k)$

Consistent Given a node, the transition *Consistent* will check the consistency of the constraint store (by means of a solver of the domain) and will generate a new node. The new node can either be the special node *false* or a node identical to its parent. Again, we will use the rules given earlier for checking failure of equality and disequality, and will rely upon the chosen solver for the other constraints. For example, a solver in CLP(FD) will typically detect inconsistency when the domain of one of the variables is empty.

If $\mathit{consistent}(CS_k)$ then

- $N_{k+1} = N_k$

If $\neg \mathit{consistent}(CS_k)$ then

- $N_{k+1} = false.$

19.3 Sample Derivation

Let us take an example from the examples document [4]. The protocol definition is given by means of the following Social Integrity Constraints:

IC_1:

$$\begin{array}{l} \mathbf{H}(tell(A, B, \textit{query-ref}(\textit{Info}), D), T) \quad \Rightarrow \\ \quad \mathbf{E}(tell(B, A, \textit{inform}(\textit{Info}, Answer), D), T_1), T_1 < T + 10 \quad \vee \\ \quad \mathbf{E}(tell(B, A, \textit{refuse}(\textit{Info}), D), T_1), T_1 < T + 10 \end{array}$$

IC_2:

$$\begin{array}{l} \mathbf{H}(tell(A, B, \textit{inform}(\textit{Info}, Answer), D), T) \quad \Rightarrow \\ \quad \mathbf{NE}(tell(A, B, \textit{refuse}(\textit{Info}), D), T_1), T_1 > T \end{array}$$

IC_3:

$$\begin{array}{l} \mathbf{H}(tell(A, B, \textit{refuse}(\textit{Info}), D), T) \quad \Rightarrow \\ \quad \mathbf{NE}(tell(A, B, \textit{inform}(\textit{Info}, Answer), D), T_1), T_1 > T \end{array}$$

and let us suppose that the history evolves from an empty history to a final history $\mathbf{HAP}^f$. Two events are in the final history; the first happens at time 1, and the second at time 2; i.e., the external set of happened events contains the following elements:

$$\begin{array}{l} \mathbf{H}(tell(yves, david, \textit{query-ref}(\textit{train\_info}), d_1), 1). \\ \mathbf{H}(tell(david, yves, \textit{inform}(\textit{train\_info}, \text{`}\textit{departs(sv,rm,10:15)}'), d_1), 2). \end{array}$$

The first node of the derivation tree is

$$N_0 \equiv \langle \emptyset, \emptyset, \mathcal{IC}_S, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

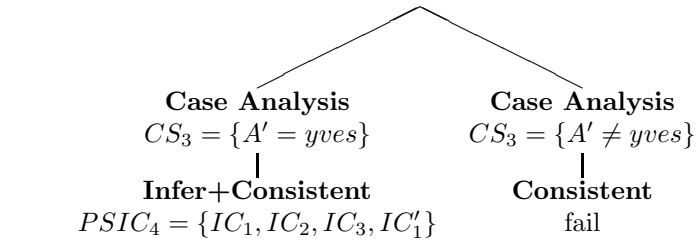
i.e., $R_0 = \emptyset$, $CS_0 = \emptyset$, $PSIC_0 = \{IC_1, IC_2, IC_3\}$ and the sets **EXP**, **HAP**, **FULF**, and **VIOL** are all empty. The only applicable transition is *Happening* with one of the events in the external set of happened events; in this example we will take the events in chronological order (as it is usually the case):

$$N_1 \equiv \langle \emptyset, \emptyset, PSIC, \emptyset, \{\mathbf{H}(tell(yves, david, \textit{query-ref}(\textit{train\_info}), d_1), 1)\}, \emptyset, \emptyset \rangle.$$

Now transition *Propagation* is applicable to the element in the history together with IC_1. First the happened event and the PSIC are copied, then the atoms are unified, and we obtain:

$$\begin{array}{lll} PSIC_2 = \{ & IC_1, IC_2, IC3, & \\ & A' = yves, B' = david, Info' = train\_info, D' = d_1, T' = 1 & \\ & \Rightarrow \quad \mathbf{E}(tell(B', A', \textit{inform}(\textit{Info'}, \textit{Answer'}), D'), T_1'), T_1' < T' + 10 & \\ & \vee \quad \mathbf{E}(tell(B', A', \textit{refuse}(\textit{Info'}), D'), T_1'), T_1' < T' + 10 & \\ & \} & \end{array}$$

Each of the equalities in the body of the implication is dealt with by *case analysis*. Let us consider the first: $A' = yves$; case analysis generates two nodes: in the first $A' = yves$ and in the second $A' \neq yves$ is put in the constraint store. Since A' is universally quantified and non flagged, $A' = yves$ succeeds when applying transition *Consistent*, and $A' \neq yves$ fails.



where

$$IC_1' = \begin{cases} B' = david, Info' = train\_info, D' = d_1, T' = 1 \\ \quad \rightarrow \mathbf{E}(tell(B', yves, inform(Info', Answer'), D'), T_1'), T_1' < T' + 10 \\ \quad \vee \mathbf{E}(tell(B', yves, refuse(Info'), D'), T_1'), T_1' < T' + 10 \end{cases}$$

After applying case analysis for each equality in the body, and the successive constraint solving step, we have only one non-failure node:

$$N_{10} = \langle \emptyset, \emptyset, PSIC_{10}, \emptyset, \mathbf{HAP}_{10}, \emptyset, \emptyset \rangle$$

where

$$\begin{array}{rcl} PSIC_{10} & = & \{IC_1, IC_2, IC_3, \\ & & true \rightarrow \mathbf{E}(tell(david, yves, inform(train\_info, Answer'), d_1), T_1'), T_1' < 1 + 10 \\ & & \vee \mathbf{E}(tell(david, yves, refuse(train\_info), d_1), T_1'), T_1' < 1 + 10\} \\ \mathbf{HAP}_{10} & = & \{\mathbf{H}(tell(yves, david, query\text{-}ref(train\_info), d_1), 1)\} \end{array}$$

Now, we can apply *Logical Equivalence* to the implication with *true* antecedent. As explained in Section 19.2.1, the consequent of the implication is moved to the conjunction R, and the variables become flagged. Since all variables occur in **E** atoms, they are existentially quantified on the whole node.

$$\begin{array}{rcl} PSIC_{11} & = & \{IC_1, IC_2, IC_3\} \\ R_{11} & = & \{\mathbf{E}(tell(david, yves, inform(train\_info, \hat{Answer'}), d_1), \hat{T'}_1), \hat{T}_1' < 1 + 10 \\ & & \vee \mathbf{E}(tell(david, yves, refuse(train\_info), d_1), \hat{T}_1'), \hat{T}_1' < 1 + 10\} \end{array}$$

Since R contains a disjunction, *splitting* can be applied, and we generate two nodes; let us consider the first node:

$$R_{12} = \{\mathbf{E}(tell(david, yves, inform(train\_info, \hat{Answer'}), d_1), \hat{T}_1'), \hat{T'}_1 < 1 + 10\}$$

Through *Abduction* and *Constrain* steps, we reach the node:

$$\begin{array}{rcl} R_{14} & = & \emptyset \\ \mathbf{EXP}_{14} & = & \{\mathbf{E}(tell(david, yves, inform(train\_info, \hat{Answer'}), d_1), \hat{T}_1')\} \\ CS_{14} & = & \{\hat{T}_1' < 1 + 10\} \end{array}$$

The declarative reading of this node is

$$\exists \hat{Answer'}, \exists \hat{T}_1'.\ \hat{T}_1' < 1 + 10 \wedge \mathbf{E}(tell(david, yves, inform(train\_info, \hat{Answer'}), d_1), \hat{T}_1')$$

Suppose that now *happening* transition is applied with the second event in the external set of happened events[31].

$$\begin{array}{lcl}\mathbf{HAP}_{15} & = & \{\mathbf{H}(tell(yves, david, \textit{query-ref}(train\_info), d_1), 1). \\ & & \mathbf{H}(tell(david, yves, inform(train\_info, `departs(sv,rm,10:15)'), d_1), 2).\}\end{array}$$

We can now apply transition *fulfillment* $\mathbf{E}$ with the event $\mathbf{H}(tell(david, yves, inform\ldots))$ in the history. The transition opens two alternative nodes: either the event in the expectation unifies with the event in the history and becomes fulfilled, or it does not unify and remains pending.

- $$\begin{array}{lcl} CS_{16} & = & \{\hat{Answer'} = `departs(sv,rm,10:15)' \wedge \hat{T}'_1 = 2 \\ & & \wedge \hat{T}'_1 < 1 + 10\} \\ \mathbf{FULF}_{16} & = & \{\mathbf{E}(tell(david, yves, inform(train\_info, \hat{Answer'}), d_1), \hat{T}'_1)\} \\ \mathbf{EXP}_{16} & = & \emptyset \end{array}$$
- $$\begin{array}{lcl} CS_{16} & = & \{(\hat{Answer'} \neq `departs(sv,rm,10:15)' \vee \hat{T}'_1 \neq 2) \\ & & \wedge \hat{T}'_1 < 1 + 10\} \\ \mathbf{FULF}_{16} & = & \emptyset \\ \mathbf{EXP}_{16} & = & \{\mathbf{E}(tell(david, yves, inform(train\_info, \hat{Answer'}), d_1), \hat{T}'_1)\} \end{array}$$

The second node can be fulfilled if the history is still open, as other events may happen matching the pending expectation. If the history gets closed, the pending expectation will become violated, so the second will be a violation node. This does not mean that the proof is in a global violation; as in SLD resolution we have a global failure if all the leaves of the proof tree are failed, in the same way in $\mathcal{S}$CIFF we have a global violation if all the leaves are violation (which is not our case, since in the first node the expectations are fulfilled).

Other transitions are applicable to this node; we do not continue the example because their application is very similar to the ones already presented. E.g., transition *Propagation* will be applied to IC_2 and the event $\mathbf{H}(tell(david, yves, inform\ldots))$ in the history, providing a new expectation $\mathbf{NE}(tell(david, yves, refuse(train\_info), d_1), \hat{T}'_1), \hat{T}'_1 > 2$.

19.4 Implementation of $\mathcal{S}$CIFF

The $\mathcal{S}$CIFF proof procedure has been implemented in SICStus Prolog [110], exploiting, in particular, the Constraint Handling Rules (CHR) [45] library. The transitions of $\mathcal{S}$CIFF have been mapped to CHR rules. The CLP(FD) solver of SICStus has been extended to deal with explicit quantification of variables, and with flagged variables. The implementation is given in detail in Deliverable D9 [5].

20 Correctness Properties of $\mathcal{S}$CIFF Proof Procedure

20.1 Soundness and completeness

We state the desirable properties of soundness and completeness for the $\mathcal{S}$CIFF proof procedure (Section 19) with respect to the declarative semantics (Section 18.2) by considering, in the following, a given society instance $\mathcal{S}_{\mathbf{HAP}}$ and a goal G for it. Depending on the openness or closure

[31] Of course, the *happening* transition was applicable also to the previous nodes. We are giving here a sample derivation, but others may be possible.

of the society instance, we state in the following the desirable properties of correctness for the proof procedure. For the sake of simplicity we do not consider CLP constraints in the program, except for equality and disequality, that are dealt with the rules given in Sections 19.2.1 and 19.2.5.

The following proposition relates the operational notion of open successful derivation with the corresponding declarative notion of goal achievability.

Proposition 20.1. Open Soundness. *Given an open society instance* $\mathcal{S}_{\mathbf{HAP}^i}$*, if*

$$\mathcal{S}_{\mathbf{HAP}^i} \mathrel{\sim\!\!\!\vdash}^{\mathbf{HAP}^f}_{\mathbf{EXP}\cup\mathbf{FULF}} G$$

with expectation answer (Definition 19.6) $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$ *then*

$$\mathcal{S}_{\mathbf{HAP}^f} \mathrel{\tilde{\models}}_{(\mathbf{EXP}\cup\mathbf{FULF})\sigma} G\sigma$$

i.e., given Definition 18.9, $Comp(SOKB \cup \Delta\sigma) \cup \mathbf{HAP}^f \cup CET \models G\sigma$*.*

Proposition above states that if there exists an open successful derivation for a goal G starting from an initial history $\mathbf{HAP}^i$ and leading to the (open) society instance $\mathcal{S}_{\mathbf{HAP}^f}$ with abduced expectation set $\mathbf{EXP} \cup \mathbf{FULF}$, and with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$, then $G\sigma$ is achievable in $\mathcal{S}_{\mathbf{HAP}^f}$ (with the expectation set $(\mathbf{EXP} \cup \mathbf{FULF})\sigma$).

In the closed case, the soundness property is stated as follows, relating the operational notion of closed successful derivation with the corresponding declarative notion of goal achievement.

Proposition 20.2. Closed Soundness. *Given a closed society instance* $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$*, if*

$$\mathcal{S}_{\mathbf{HAP}^i} \vdash^{\overline{\mathbf{HAP}^f}}_{\mathbf{EXP}\cup\mathbf{FULF}} G$$

with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$ *then*

$$\mathcal{S}_{\overline{\mathbf{HAP}^f}} \models_{(\mathbf{EXP}\cup\mathbf{FULF})\sigma} G\sigma$$

Soundness in the closed case states that if there exists a closed successful derivation for a goal G starting from an initial history $\mathbf{HAP}^i$ and leading to the (closed) society instance $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$ with abduced expectation set $\mathbf{EXP} \cup \mathbf{FULF}$, and with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$, then $G\sigma$ is achieved in $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$ (with the expectation set $(\mathbf{EXP} \cup \mathbf{FULF})\sigma$).

In Appendix D.1, we introduce some lemmas useful to prove the property of open and closed soundness. These Lemmas allow us to establish a corresponding $\mathcal{S}$CIFF computation where all the incoming events are considered at the beginning of the computation, instead of interleaving Happening transitions with the other ones.

We first prove soundness property for the open case (Proposition 20.1) in Appendix D.3, then for the closed case (Proposition 20.2) in Appendix D.4.

The proof of correctness (soundness, in particular) is given by exploiting soundness results of the IFF proof procedure with respect to three-valued completion semantics. We map $\mathcal{S}$CIFF programs into IFF (rewritten) programs, and then prove that open/closed $\mathcal{S}$CIFF successful derivations have a counterpart in IFF derivations. We can therefore exploit soundness of this latter proof procedure, in order to prove (open and closed) soundness of $\mathcal{S}$CIFF. The proof of

soundness is given - at this stage - when the final node of the $\mathcal{S}$CIFF computation contains no literal abduced with a universally quantified variable (we do not restrict the syntax of the SOKB or of $\mathcal{IC}_S$, but we give correctness results in some of the derivations). The extension to the case with universally quantified variables in abducibles is work in progress.

The following two propositions state the converse properties, respectively for the open and closed case. The proof of these properties is work in progress.

Proposition 20.3. Open Completeness. *Given an open society instance $\mathcal{S}_{\mathbf{HAP}}$, and a (ground) goal G, for any set of ground expectations, $\Delta = \mathbf{EXP} \cup \mathbf{FULF}$, such that $\mathcal{S}_{\mathbf{HAP}} \mid\!\approx_\Delta G$ then $\exists \Delta'$ such that $\mathcal{S}_\emptyset \mid\!\sim^{\mathbf{HAP}}_{\Delta'} G$ with an expectation answer (Δ', σ) such that $\Delta'\sigma \subseteq \Delta$.*

Completeness in the open case states that if goal G is achievable in an open society instance under the expectation set Δ, then an open successful derivation can be obtained for G, possibly computing a set Δ' of the expectations whose grounding (according to the expectation answer) is a subset of Δ.

Proposition 20.4. Closed Completeness. *Given a closed society instance $\mathcal{S}_{\overline{\mathbf{HAP}}}$, a (ground) goal G, for any set of ground expectations, $\Delta = \mathbf{EXP} \cup \mathbf{FULF}$ such that $\mathcal{S}_{\overline{\mathbf{HAP}}} \models_\Delta G$ then $\exists \Delta'$ such that $\mathcal{S}_\emptyset \vdash^{\mathbf{HAP}}_{\Delta'} G$ with an expectation answer (Δ', σ) such that $\Delta'\sigma \subseteq \Delta$.*

Completeness in the closed case states that if goal G is achieved in a closed society instance under the expectation set Δ, then a closed successful derivation can be obtained for G, possibly computing a set Δ' of the expectations whose grounding (according to the expectation answer) is a subset of Δ.

21 Discussion and planned future activity

We have described the proof procedure to be adopted by the society component. The proof procedure is capable of checking the conformance of computees' interactions to specified protocols, raise expectations (on the basis of the social integrity constraints), and identify fulfillment or violation of expectations.

The proof procedure has been formally defined by a set of transitions, some of which are inherited from the IFF proof procedure [47] with constraints (C-IFF), and some of which are original. Its main features are:

- it accepts new happened events;
- it contains transitions for the treatment of constraints [61], as C-IFF;
- it contains transitions for the concepts of *fulfillment* and *violation* [94].

Soundness has been proven for the open and closed cases, when abduced atoms do not contain universally quantified variables.

In future work we have to:

- establish (preferred) orderings among transitions, and in particular determine when the *Happening* transition has to be performed;

- consider non-unary domain restrictions for the universally quantified variables;
- prove further properties of the proof procedure (at least the open and closed soundness property for computations involving abducibles with universally quantified variables, and possibly completeness).

As concerns proofs of correctness, up to now we have considered the case of (possibly non-ground) goals, and expectation sets without universally quantified variables. A matter of future work is to consider the case on raised expectations containing such variables. To this purpose, we have shown a lemma (Lemma D.12), that will be the backbone for the proof of soundness in the more general case.

A further issue to be considered concerns the computational complexity of the proof procedure, with reference also to its implementation. This will possibly be addressed in Workpackage 5.

Finally, an interesting issue, relating societies to its members, is represented by publication of expectations, so to make members aware of what the society expects them to do. With this awareness, members might apply inner reasoning (in particular Goal Decision, and Planning) in order to act appropriately. This is subject of future work, too.

A further issue to be investigated (in particular, in the third year of the project, inside Workpackage 5) is which properties can be stated (and proved) for societies of computees on the basis of the structure of Social Integrity Constraints. For instance, a property could be the guaranteed satisfaction of **E**-consistency (and $\neg$-consistency) of (raised) expectations.

22 Related work

In this document we presented the operational counterpart of a social framework introduced and discussed in Deliverable D5 [94]. From the model viewpoint, in D5 we have presented our work in relationship with other approaches to protocol and communicative act semantic specification.

Noteworthily, Artikis et al. [9] present a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called open computational societies, and they present a formal framework for specifying and animating systems where the behavior of the members and their interactions cannot be predicted in advance, and for reasoning about and verifying the properties of such systems. A noteworthy difference with [9] is that we do not explicitly represent the institutional power of the members and the concept of valid action. Permitted are all social events that do not determine a violation, i.e., all events that are not explicitly forbidden are allowed.

Yolum and Singh [123] apply a variant of Event Calculus [82] to commitment-based protocol specification. The semantics of messages (i.e., their effect on commitments) is described by a set of *operations* whose semantics, in turn, is described by *predicates* on *events* and *fluents*; in addition, commitments can evolve, independently of communicative acts, in relation to *events* and *fluents* as prescribed by a set of *postulates*. Such a way of specifying protocols is more flexible than traditional approaches based on action sequences in that it prescribes no initial and final states or transitions explicitly, but it only restricts the agent interaction in that, at the end of a protocol run, no commitment must be pending; agents with reasoning capabilities can themselves plan an execution path suitable for their purposes (which, in that work, is implemented by an abductive event calculus planner). Our notion of expectation is more

general than that of commitment proposed by Yolum and Singh [123] or in other commitment-based works, such as the proposal by Fornara and Colombetti [44]: it represents the necessity of a (past or future) event, and is not bound to have a debtor or a creditor, or to be brought about by an agent.

From the operational framework viewpoint, the $\mathcal{S}$CIFF is an extension of the IFF abductive proof procedure [47]. Various abductive proof procedures have been proposed in the past. We will only mention the ones closest to our work, the interested reader can refer to the survey by Kakas et al. [68] for a complete discussion.

The IFF proof procedure [47] is probably the one most related to our work. Our operational semantics can be considered as an extension of the IFF proof procedure that also:

- abduces atoms with variables universally quantified;
- deals with CLP constraints, also imposed as quantifier restrictions on universally quantified variables;
- is more dynamic, in fact new events may arrive, and the proof procedure dynamically takes them into consideration in the knowledge base;
- has the new concepts, related to on-line verification, of *fulfillment* and *violation*.

Other proof procedures deal with constraints; in particular we mention ACLP [72] and its successor, the $\mathcal{A}$-system [76], that are deeply focussed on efficiency issues. Both of these proof procedures use integrity constraints in the form of denials (e.g., $A, B, C \rightarrow \bot$), instead of forward rules as the IFF (and the proof procedure of the Compliance Verifier). Both of these proof procedures only abduce existentially quantified atoms, and do not consider quantifier restrictions, that give more expressivity to our proof procedure.

Relevant work has been done in the past with the integration of the IFF proof procedure with constraints by Kowalski et al. [78]; however the integration is more focussed on a theoretical uniform view of abducibles and constraints than to an implementation of a proof procedure with constraints.

Other implementations have been given of abductive proof procedures in Constraint Handling Rules [2, 48]. Our implementation is more adherent to the theoretical operational semantics (in fact, every transition is mapped to CHR rules) and exploits the uniform understanding of constraints and abducibles noted by Kowalski et al. [78].

The use of abduction for verification is an idea which can be found also in other work. Noteworthily, Russo et al. [103] use an abductive proof procedure for analyzing event-based requirements specifications. In their approach, the system has a declarative specification given through the Event Calculus [82] axioms, and the goal is proving that some invariant I is true in all cases. The events that may happen are mapped to abducible atoms, and the goal $\neg I$ is given to an abductive proof procedure. In this way, if the refutation succeeds, the set Δ of abduced atoms contains a counterexample, i.e., a possible history for which the invariant I does not hold. This method uses abduction for analyzing the correctness of specifications, while our system is more focussed on the on-line check of compliance of a set of agents.

Part IV
Conclusions

Abstract. In this part we briefly conclude the document by summarising its achievements and those of the SOCS' consortium within WP3 in the second year of the project more generally.

23 Summary and evaluation

In this deliverable we reported the activities of the consortium during the second year of the project within workpackage WP3, which concerns the definition of the computational model of computees and societies of computees. The computational models that we have defined continue the spirit of the modular design of computees and societies embarked in the first year, by using a number of proof procedures for the different aspects appropriately integrated.

A range of candidate proof procedures were studied and decisions made about the specific ones to be used. Investigations were carried out on the specific applications of these to the individual computee capabilities and cycle and the society model, and in each case a set of necessary and desirable extensions and modifications have been identified.

We defined three proof procedures:

- C-IFF, an abductive proof procedure for the abductive logic programming tasks underlying the computee model, namely the planning, reactivity, identification of preconditions and temporal reasoning capabilities;
- $\mathcal{S}$CIFF, another abductive proof procedure for the society model;
- an argumentation-based proof procedure for $LPwNF$, the logic programming with priorities framework used in the computee model for all the tasks requiring preferential reasoning, namely the goal decision capability and the cycle theory.

Both C-IFF and $\mathcal{S}$CIFF are extensions of the IFF proof procedure [47] to deal with constraint predicates. C-IFF keeps all the rewrite rules of IFF, and adds a constraint solving rule and an additional case analysis rule, which applies to constraint atoms. C-IFF also relaxes the allowedness restriction of IFF. $\mathcal{S}$CIFF keeps some of the rewrite rules of the IFF (e.g. unfolding), modifies some others (e.g. equality rewriting), drops some of them (e.g. factoring) and adds some rewrite rules (dealing with the history, the fullfillment set, and the consistency of the set of expectations, and performing constraint solving as well as case analysis over constraints). Both procedures return answers consisting of sets of abducibles and sets of constraint atoms. Despite their similarities, the proof procedures differ considerably, in order to cope with the different tasks they are set to perform. The main differences are as follows:

- the treatment of variables in $\mathcal{S}$CIFF is more sophisticated due to the need to deal with universally quantified variables in abduced atoms. Instead C-IFF imposes allowdness restrictions ensuring that abduced atoms have only existentially quantified or free variables.
- C-IFF needs to deal with more sophisticated integraty constraints than social integrity constraints, in that unfolding might need to be applied to their bodies in order to check their satisfaction.

- C-IFF is implemented in SICStus, $\mathcal{S}$CIFF is implemented in SICStus via CHR [45] (see the companion deliverable D9 [5]). Both implementations use the built-in SICStus constraint solver.

On the basis of these three proof procedures we defined (all the components of) the computational counterparts of the KGP model for computees defined in deliverable D4 [63] and for the society model defined in deliverable D5 [94].

The main contributions of the work done in WP3 regarding the computational model of a single *computee* can be summarised as follows:

Extension of the proof procedure for $LPwNF$. The framework $LPwNF$ for reasoning with logic programs with priorities has been enhanced in order to provide computational counterparts of the Cycle Theory and the Goal Decision capability of a computee. The computational model has been directly built from the argumentation based semantics of $LPwNF$, exploiting the techniques of [75]. The proof theory is given in terms of derivations of trees where each node in a tree contains an argument against its corresponding parent node. The proposed argumentation framework has been implemented in the `Gorgias` system.

The original $LPwNF$ framework and its computational model have been extended with *dynamic priorities* as required by the computee model. Moreover, *abduction* has been incorporated within $LPwNF$, despite the fact that this is not required to realise the basic computee model as currently specified in [63], since it can provide a useful extension of the model to cope more fully with the demands of the Global Computing environment in which computees operate.

Definition of the C-IFF proof procedure. In order to support the capabilities of the computee, and specifically those based on the Abductive Event Calculus, like Planning, Reactivity, and Temporal Reasoning, the C-IFF abductive proof procedure has been defined and implemented. C-IFF is an extension of the IFF abductive proof procedure [47], which encompasses constraint satisfaction within conventional abductive logic programming. Constraints are needed in order to deal with temporal information, a relevant feature of the KGP model of computees, which allows them to reason about changes in an evolving open environment where they typically operate.

Constraint satisfaction has been embedded into the definition of the IFF proof procedure, by appropriately extending the set of rewriting rules on which it is based. In the implementaion of C-IFF within D9, concrete constraint satisfaction is delegated to the constraint solver provided by the SICStus Prolog environment, on which C-IFF has been implemented.

Besides constraint handling, C-IFF also generalises IFF to deal with non-allowed abductive logic programs, such as the Abductive Event Calculus that C-IFF is applied to within the computee model.

Integration of proof procedures into a unique computational model. This is a specific aspect of the SOCS project which, starting from the plethora of specialised proof procedures targeted to solve specific problems, recognised the needs for a general model, capable of integrating the procedures possibly needed by a computational entity, within the same conceptual model.

Currently, the two developed and implemented proof procedures fully support the computational logic based reasoning of a computee in an integrated computational model. Such a model corresponds, up to some simplifying assumptions, to the integrated formal model defined within WP1.

Advances in integrated multi-reasoning. Building on the defined proof procedures, the computational model of the single computee exhibit forms of non-monotonic reasoning, like temporal and preferential reasoning, that are state-of-the-art in themselves, and an advance as an integrated form of reasoning for an intelligent computational entity.

Correspondences of the computational and formal models. The computational models of each capability are provided with a sketch proof of soundness with respect to the relative formal model. On top of these, Transitions, Selection functions and Cycle computational models are proven to be sound with respect to the corresponding formal models. Formally stating the relations between formal and computational models goes into the direction of facilitating the formal assessment of properties of the behaviour of a computee. Completing the sketched proof and searching for more general results of correctness and possible forms of completeness results need further investigation.

For an evaluation of the computee model in the context of the Global Computing programme see deliverable D11 [13].

The main contributions of the work done in WP3 regarding the computational model of the *society* can be summarised as follows:

Definition of the $\mathcal{S}$CIFF proof procedure. We have interpreted the protocol conformance checks and the normative control performed by the society as abductive tasks, and defined an extension of the IFF abductive proof procedure to deal with this task. The extension is non-trivial, and deals with complex forms of variables quantification in abductive logic programs, as well as constraint predicates.

Advances in agent communication and norm-based reasoning.

Correspondences of the computational and formal model. We have proven that $\mathcal{S}$CIFF is a correct computaitional counterpart of the society model of D5 [94] (under some assumptions).

For an evaluation of the society model in the context of the Global Computing programme see deliverable D11 [13].

Finally, we believe that a further contribution of the work presented in this document is that we have devised a computational model for the single computees and a computational model for the societies which are compatible with one another and thus can be easily integrated.

References

[1] The DLV Project: A Disjunctive Datalog System (and more). Electronically available at http://www.dbai.tuwien.ac.at/proj/dlv/.

[2] S. Abdennadher and H. Christiansen. An experimental CLP platform for integrity constraints and abduction. In H. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen, and H. Christiansen, editors, *FQAS, Flexible Query Answering Systems*, LNCS, pages 141–152, Warsaw, Poland, Oct. 25 - 28 2000. Springer-Verlag.

[3] ACLP: Abductive Constraint Logic Programming. Electronically available at http://www.cs.ucy.ac.cy/aclp/.

[4] M. Alberti, A. Bracciali, F. Chesani, N. Demetriou, U. Endriss, W. Lu, F. Sadri, A. Kakas, E. Lamma, P. Mello, M. Milano, K. Stathis, F. Toni, and P. Torroni. Examples of the functioning of computees and their societies. Discussion Note IST3250/ICSTM//DN/I/a2, SOCS Consortium, Dec. 2003. http://lia.deis.unibo.it/Research/Projects/SOCS/.

[5] M. Alberti, A. Bracciali, F. Chesani, U. Endriss, M. Gavanelli, W. Lu, K. Stathis, and P. Torroni. SOCS prototype. Technical report, SOCS Consortium, 2003. Deliverable D9.

[6] J. Alferes, L. M. Pereira, and T. Swift. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theory and Practice of Logic Programming*, 2003.

[7] K. R. Apt. Logic programming. In *Handbook of Theoretical Computer Science*, volume B, pages 493–574. Elsevier Science Publishers, 1990.

[8] K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, March/April 1999.

[9] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III*, pages 1053–1061, Bologna, Italy, July 15–19 2002. ACM Press. http://portal.acm.org/ft_gateway.cfm?id=545070&type=pdf&dl=GUIDE&dl=ACM%&CFID=4415868&CFTOKEN=57395936.

[10] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

[11] R. Bordini, A. L. C. Bazzan, R. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. Agentspeak(xl): Efficient intention selection in bdi agents via decision-theoretic task scheduling. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III*, pages 1294 – 1302, Bologna, Italy, July 15–19 2002. ACM Press. http://portal.acm.org/ft_gateway.cfm?id=545122&type=pdf&dl=GUIDE&dl=ACM%&CFID=4415868&CFTOKEN=57395936.

[12] R. H. Bordini and Á. F. Moreira. Proving the asymmetry thesis principles for a BDI agent-oriented programming language. In J. Dix, J. A. Leite, and K. Satoh, editors, *Computational Logic in Multi-Agent Systems: 3rd International Workshop, CLIMA'02, Copenhagen, Denmark, August 1, 2002, Proceedings*, number 93 in Datalogiske Skrifter (Writings on Computer Science), pages 94–108. Roskilde University, Denmark, 2002.

[13] A. Bracciali, A. C. Kakas, E. Lamma, P. Mello, K. Stathis, F. Toni, and P. Torroni. Evaluation and self-assesment. Technical report, SOCS Consortium, 2003. Deliverable D11.

[14] M. Bratman. *Intensions, plans and practical reason*. Harvard University Press, 1987.

[15] M. Bratman, D. Israel, and M. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4, 1988.

[16] F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur. Formal specification of multi-agent systems: a real-world case. In *Proceedings of the 1st International Conference on Multiagent Systems, San Francisco, California*, pages 25–32, San Francisco, CA, USA, 1995. AAAI Press. `http://citeseer.nj.nec.com/brazier95formal.html`.

[17] F. M. T. Brazier, B. Dunin-Keplicz, J. Treur, and R. Verbrugge. Modelling internal dynamic behaviour of BDI agents. In *ModelAge Workshop*, pages 36–56, 1997. `http://citeseer.nj.nec.com/103009.html`.

[18] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(1):67–94, 1997. `http://citeseer.nj.nec.com/brazier97desire.html`.

[19] F. M. T. Brazier, J. Treur, N. J. E. Wijngaards, and M. Willems. Temporal semantics of complex reasoning tasks. In *Proc. of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW'95*, pages 15/1–15/17. Calgary:SRDG Publications, 1995.

[20] G. Brewka. Reasoning about priorities in default logic. In *AAAI-94, pp. 940-945*, 1994.

[21] G. Brewka. Well founded semantics for extended logic programs with dynamic preferences. *Artificial Intelligence Research*, 4:19–36, 1996.

[22] G. Brewka. Dynamic argument systems: a formal model of argumentation process based on situation calculus. In *Journal of Logic and Computation, 11(2), pp. 257-282*, 2001.

[23] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 86–97. Morgan Kaufmann, San Francisco, California, 1998.

[24] H. Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66:235–271, 1994.

[25] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, January 1996.

[26] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[27] P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[28] L. Console, D. T. Dupré, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[29] P. T. Cox and T. Pietrzykowski. Causes for events: Their computation and applications. In *Proceedings CADE-86*, pages 608–621, 1986.

[30] M. Dastani, F. S. de Boer, F. Dignum, W. van der Hoek, M. Kroese, and J. C. Meyer. Programming the deliberation cycle of cognitive robots. In *Proc. of 3rd International Cognitive Robotics Workshop (CogRob2002)*, Edmonton, Alberta, Canada, 2002.

[31] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[32] M. Denecker and D. D. Schreye. SLDNFA: An abductive procedure for normal abductive programs. In K. R. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 686–702, Cambridge, Nov. 9–13 1992. MIT Press.

[33] M. Denecker and D. D. Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In *Logic Programming, Proceedings of the 1993 International Symposium, Vancouver, British Columbia, Canada*, pages 147–163. The MIT Press, 1993.

[34] Y. Dimopoulos and A. C. Kakas. Logic programming without negation as failure. In *Logic Programming, Proceedings of the 1995 International Symposium, Portland, Oregon*, pages 369–384, 1995.

[35] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents V, Agent Theories, Architectures, and Languages, 5th International Workshop, ATAL '98, Paris, France, Proceedings*, number 1365 in Lecture Notes in Artificial Intelligence, pages 155–176. Springer-Verlag, 1998.

[36] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[37] B. M. Dunin-Keplicz and J. Treur. Compositional formal specification of multi-agent systems. In *Proc. of the ECAI'94 Workshop on Agent Theories, Architectures and Languages, Lecture Notes in AI*, volume 890, pages 102–117. Springer-Verlag, 1995.

[38] T. Eiter, V. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, March 1999.

[39] T. Eiter and V. S. Subrahmanian. Deontic action programs. In *Workshop on Foundations of Models and Languages for Data and Objects*, pages 37–54, 1998. `http://citeseer.nj.nec.com/eiter98deontic.html`.

[40] T. Eiter and V. S. Subrahmanian. Heterogeneous active agents, II: Algorithms and complexity. *Artificial Intelligence*, 108(1–2):257–307, 1999. `http://citeseer.nj.nec.com/eiter99heterogeneous.html`.

[41] T. Eiter, V. S. Subrahmanian, and T. J. Rodgers. Heterogeneous active agents, III: Polynomially implementable agents. *Artificial Intelligence*, 117(1):107–167, 2000. `http://citeseer.nj.nec.com/eiter99heterogeneous.html`.

[42] K. Eshghi. Abductive planning with event calculus. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming — ICLP'88*, pages 562–579, 1988.

[43] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proceedings of the 6th International Conference on Logic Programming*, pages 234–255. MIT Press, 1989.

[44] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II*, pages 535–542, Bologna, Italy, July 15–19 2002. ACM Press. `http://portal.acm.org/ft_gateway.cfm?id=544868&type=pdf&dl=GUIDE&dl=ACM%&CFID=4415868&CFTOKEN=57395936`.

[45] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3):95–138, Oct. 1998.

[46] T. H. Fung. *Abduction by Deduction*. PhD thesis, Imperial College London, 1996.

[47] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, Nov. 1997.

[48] M. Gavanelli, E. Lamma, P. Mello, M. Milano, and P. Torroni. Interpreting abduction in CLP. In F. Buccafurri, editor, *APPIA-GULP-PRODE Joint Conference on Declarative Programming*, pages 25–35, Reggio Calabria, Italy, `http://www.informatica.ing.unirc.it/agp03/`, Sept. 3-5 2003. Università Mediterranea di Reggio Calabria. `http://www.ing.unife.it/docenti/MarcoGavanelli/papers/AGP03.pdf`.

[49] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 972–978, 1989.

[50] M. P. Georgeff and F. F. Ingrand. Monitoring and control of spacecraft systems using procedural reasoning. In *Workshop of the Space Operations-Automation and Robotics*, Houston, Texas, July 1989.

[51] M. P. Georgeff and A. L. Lansky. Procedural knowledge. In *Proceedings of the IEEE Special Issue on Knowledge Representation*, volume 74, pages 1383–1398, 1986.

[52] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence, AAAI'87*, pages 677–682, Seattle, WA, USA, July 1987. Morgan Kaufmann Publishers.

[53] G. D. Giacomo, Y. Lesperance, and H. J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000. `http://citeseer.nj.nec.com/degiacomo00congolog.html`.

[54] G. D. Giacomo, H. J. Levesque, and S. Sardia. Incremental execution of guarded theories. *ACM Transactions on Computational Logic*, 2(4):495–525, October 2001.

[55] G. Governatori, M. J. Maher, G. Antoniou, and D. Billington. Argumentation semantics for defeasible logics. In R. Mizoguchi and J. Slaney, editors, *PRICAI 2000: Topics in Artificial Intelligence*, volume 1886 of *LNAI*, pages 27–37, Berlin, 2000. Springer-Verlag.

[56] K. V. Hindriks, F. S. D. Boer, W. van der Hoek, and J.-J. C. Meyer. Formal semantics for an abstract agent programming language. In *Intelligent Agents IV, Agent Theories, Architectures, and Languages, 4th International Workshop, ATAL '97, Providence, Rhode Island, Proceedings*, volume 1365 of *Lecture Notes in Computer Science*, pages 215–229. Springer-Verlag, 1998.

[57] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. C. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[58] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. C. Meyer. Semantics of communicating agents based on deduction and abduction. In *In IJCAI'99 Workshop on Agent Communication Languages*, 1999.

[59] K. V. Hindriks, M. d'Inverno, and M. Luck. An formal architecture for 3APL. *ZB 2000*, pages 168–187, 2000.

[60] F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, 1992. `ftp://ftp.laas.fr/pub/Publications/1992/92521.ps`.

[61] J. Jaffar and M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[62] J. Jaffar, M. Maher, K. Marriott, and P. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.

[63] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. A logic-based approach to model computees. Technical report, SOCS Consortium, 2003. Deliverable D4.

[64] A. Kakas and L. Michael. On the qualification problem and elaboration tolerance. In *Proceedings of the 6th AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*, Stanford, Palo Alto, alifornia, U.S.A, 2003.

[65] A. Kakas, R. Miller, and F. Toni. E-res: Reasoning about actions, events and observations. In *Proc. of LPNMR-01*, 2001.

[66] A. C. Kakas. ACLP: integrating abduction and constraint solving. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'00, Breckenridge, CO*, 2000.

[67] A. C. Kakas and M. Denecker. Abduction in logic programming. In A. C. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond. Part I*, number 2407 in Lecture Notes in Artificial Intelligence, pages 402–436. Springer-Verlag, 2002.

[68] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[69] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.

[70] A. C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443. Ohmsha Ltd., 1990.

[71] A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics for logic programs. In *Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy*, pages 504–519, 1994.

[72] A. C. Kakas, A. Michael, and C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, 44(1-3):129–177, July 2000.

[73] A. C. Kakas and P. Moraïtis. Argumentative agent deliberation, roles and context. In J. L. J. Dix and K. Satoh, editors, *CLIMA 2002: 3rd International Workshop on Computational Logics in Multi-Agent Systems. Electronic Notes on Theoretical Computer Science*, volume 70. Elsevier Science Publishers, 2002.

[74] A. C. Kakas and P. Moraitis. Argumentation based decision making for autonomous agents. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, pages 883–890, Melbourne, Victoria, July 14–18 2003. ACM Press.

[75] A. C. Kakas and F. Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9:515–562, 1999.

[76] A. C. Kakas, B. van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 591–596, Seattle, Washington, USA, August 2001. Morgan Kaufmann Publishers.

[77] R. Kowalski and F. Toni. Abstract argumentation. *Artificial Intelligence and Law Journal, Special Issue on Logical Models of Argumentation*, 4:275–296, 1996.

[78] R. Kowalski, F. Toni, and G. Wetzel. Executing suspended logic programs. *Fundamenta Informaticae*, 34:203–224, 1998. `http://www-lp.doc.ic.ac.uk/UserPages/staff/ft/PAPERS/slp.ps.Z`.

[79] R. A. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. In *Proc of the International Workshop on Logic in Databases,*

San Miniato, Italy, volume 1154 of *Lecture Notes in Computer Science*, pages 137–149. Springer-Verlag, 1996.

[80] R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25(3/4):391–419, 1999.

[81] R. A. Kowalski, F. Sadri, and F. Toni. An agent architecture that combines backward and forward reasoning. In B. Gramlich and F. Pfenning, editors, *Proceedings of the CADE-15 Workshop on Strategies in Automated Deduction*, pages 49–56, November 1998.

[82] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[83] R. A. Kowalski and F. Toni. Argument and reconciliation. In *Proceedings of Workshop on Legal Reasoning International Symposium on FGCS Tokyo*, 1994.

[84] K. Kunen. Negation in logic programming. In *Journal of Logic Programming*, volume 4, pages 289–308, 1987.

[85] J. Leite, J. Alferes, and L. M. Pereira. Minerva - combining societal agents knowledge. Technical report, Dept. Informática, Universidade Nova de Lisboa, 2001.

[86] J. A. Leite. *Evolving Knowledge Bases*. IOS Press, 2003.

[87] J. A. Leite, J. J. Alferes, and L. M. Pereira. $\mathcal{MINERVA}$: A dynamic logic programming agent architecture. In *Intelligent Agents VIII: 8th International Workshop, ATAL 2001, Seattle, WA, USA, Revised Papers*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 141–157, 2002.

[88] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997. `http://citeseer.nj.nec.com/article/levesque97golog.html`.

[89] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd extended edition, 1987.

[90] P. Mancarella and G. Terreni. Extensions of kakas-mancarella logic-abductive proof procedure. Internal note (forthcoming).

[91] P. Mancarella, G. Terreni, and F. Toni. Abductive proof procedures: A multiagent oriented survey. Technical report, Dept. Computer Science, University of Pisa, 2002.

[92] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[93] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969. `http://citeseer.nj.nec.com/article/levesque97golog.html`.

[94] P. Mello, P. Torroni, M. Gavanelli, M. Alberti, A. Ciampolini, M. Milano, A. Roli, E. Lamma, F. Riguzzi, and N. Maudet. A logic-based approach to model interaction amongst computees. Technical report, SOCS Consortium, 2003. Deliverable D5.

[95] S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguying. In *Logic and Computation 8 (3), 261-292*, 1998.

[96] H. Prakken and G. Sartor. A dialectical model of assessing conflicting arguments in legal reasoning. In *Artficial Intelligence and Law*, volume 4, pages 331–368, 1996.

[97] H. Prakken and G. Sartor. A system for defeasible argumentation, with defeasible priorities. In *International Conference on Formal and Applied Practical Reasoning*, LNAI 1085, pages 510–524. Springer-Verlag, 1996.

[98] A. Rao and M. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the International Workshop on Knowledge Representation, KR'92*, pages 439–449, 1992.

[99] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In R. van Hoe, editor, *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer-Verlag, 1996.

[100] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the 1st International Conference on Multiagent Systems, San Francisco, California*, pages 312–319, San Francisco, CA, June 1995.

[101] A. S. Rao and M. P. Georgeff. Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In J. Myopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 498–505, Sydney, Australia, 1991. Morgan Kaufmann Publishers. `http://citeseer.nj.nec.com/rao91asymmetry.html`.

[102] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 317–328. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

[103] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer. An abductive approach for analysing event-based requirements specifications. In P. Stuckey, editor, *Logic Programming, 18th International Conference, ICLP 2002*, volume 2401 of *Lecture Notes in Computer Science*, pages 22–37, Berlin Heidelberg, 2002. Springer-Verlag.

[104] F. Sadri and F. Toni. Abduction with negation as failure for active and reactive rules. In E. Lamma and P. Mello, editors, *AI*IA'99: Advances in Artificial Intelligence, Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence, Bologna*, number 1792 in Lecture Notes in Artificial Intelligence, pages 49–60. Springer-Verlag, 2000.

[105] F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 419–431. Springer-Verlag, Sept. 2002.

[106] K. Satoh and N. Iwayama. A Query Evaluation Method for Abductive Logic Programming. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 671–685. The MIT Press, 1992.

[107] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1055–1060, 1989.

[108] M. Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44(1-3):207–240, 2000.

[109] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[110] SICStus prolog user manual, release 3.8.4, May 2000. `http://www.sics.se/isl/sicstus/`.

[111] K. Stathis. Location-aware SOCS: The Leaving San Vincenzo scenario. Technical Report IST32530/CITY/002/IN/PP/a1, SOCS consortium, 2002.

[112] P. Stuckey. Negation and constraint logic programming. *Information and Computation*, 118(1):12–33, 1995.

[113] V. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, and R. Ross. *Heterogenous Active Agents*. MIT-Press, 2000.

[114] S. R. Thomas. The PLACA agent programming language. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents*, Berlin, 1995. Springer-Verlag. `http://citeseer.nj.nec.com/rao96agentspeakl.html`.

[115] P. van Hentenryck and Y. Deville. The Cardinality Operator: A new Logical Connective for Constraint Logic Programming. In K. Furukawa, editor, *Logic Programming, Proceedings of the Eigth International Conference, Paris, France*, volume 2, pages 745–759, 1991.

[116] P. van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(fd). Technical Report CS-93-02, Department of Computer Sciences, Brown University, Jan. 1993.

[117] B. van Nuffelen. personal communication.

[118] B. van Nuffelen and M. Denecker. Problem solving in ID-logic with aggregates. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'00, Breckenridge, CO*, pages 1–9, 2000.

[119] P. VanBeek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[120] D. Weerasooriya, A. Rao, and K. Ramamohanarao. Design of a concurrent agent-oriented language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents II, Agent Theories, Architectures, and Languages, IJCAI '95, Workshop (ATAL), Montreal, Canada, Proceedings*, pages 386–402. Springer-Verlag, 1995. `http://citeseer.nj.nec.com/weerasooriya94design.html`.

[121] M. Witkowski and K. Stathis. A dialectic architecture for computational autonomy. In M. Nickles and M. Rovatsos, editors, *AUTONOMY 2003*, to appear as LNCS in 2003.

[122] Y. Xanthakos. *Semantic integration of information by abduction*. PhD thesis, Department of Computing, Imperial College London, UK, 2003.

[123] P. Yolum and M. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II*, pages 527–534, Bologna, Italy, July 15–19 2002. ACM Press. `http://portal.acm.org/ft_gateway.cfm?id=544867&type=pdf&dl=GUIDE&dl=ACM%&CFID=4415868&CFTOKEN=57395936`.

Part V
Appendices

A Extensions of the TR computational model

A.1 Theories with actions that happen within intervals

In this section we show how Assumption 2 of Section 11.4.2 can be relaxed. We extend the KB_{TR} theory by allowing an action to occur within a given interval, at an unknown but fixed time point. This will be useful in dealing with incomplete information about the environment, where possible inconsistencies can be recovered by assuming that an action occurred in a given interval, even without being able to say exactly when. Moreover, other capabilities or transitions of computees may require referring to actions that the computee is committed, in its Plan, to execute at an unspecified, but constrained, future time.

Informally speaking, an interval where an action occurs at an unspecified time point is a sort of fuzzy interval, where it might not be possible to assert whether a given fluent holds or it does not hold. In this context we interpret credulous reasoning as the existence of a *correct placement* of the action, or of the actions if there are more than one, which allows the fluent to be credulously proved against the grounded theory. Skeptical reasoning, instead, is interpreted as the fact that all the possible correct placements allow the fluent to be skeptically proved (at the end of this section we also discuss a weaker notion of skeptical reasoning in this context).

In the rest of this section we introduce the necessary notation, define the notion of correct placement and show how this allows us to define a computational model.

An interval $[s, e]$, or $[s, \infty]$, of possible values for a time variable T is represented by a set of constraints over the variable, like $\{s \leq T, T \leq e\}$, or $\{s \leq T\}$, respectively. A constraint can also relate two distinct variables, like $\{T_i \leq T_j\}$. We assume that each time variable T_i do have an associated ground interval $[s_i, e_i]$.[32] The only admitted constraints are the binary operators $\leq$ and $<$, defined over ground time points and time variables. An action that occurs at an unspecified instant within an interval is expressed as $happens(a, T), C$,[33] where C is the constraint set associated to the variable (containing at least the definition of the interval itself).

Definition A.1 (KB_{TRint}). *The theory KB_{TRint} consists of the theory KB_{TR} extended with a set of non-ground predicates $happens(a, T)$ and a set of temporal constraints C. Time variables are quantified from the outside, so that*

$$\begin{array}{c} happens(a_1, T_1), C_1(T_1, \ldots, T_n). \\ \vdots \\ happens(a_n, T_n), C_n(T_1, \ldots, T_n). \end{array}$$

[32] This assumption facilitates the definition of the computational model, while it is not restrictive, in the sense that either such an interval can be derived by the set of constraints, or it can be assumed to be $[0, \infty]$.

[33] In this section, with a small abuse of notation, we refer to *happens* as if they were directly occurring in $KB0$, in place of the expected *executed* and relative bridge rule $happens \leftarrow executed$.

stands for

$$\begin{array}{c} \exists\ T_1 \ldots, T_n \\ happens(a_1, T_1). \\ \vdots \\ happens(a_n, T_n). \\ C(T_1, \ldots, T_n). \end{array}$$

with $C(T_1, \ldots, T_n)$ *the union of the constraint sets. A predicate* $happens(a, T), C(T_1, \ldots, T_n)$. *is called an* interval action.

The set of temporal constraints $C(T_1, \ldots T_n)$, assumed to be satisfiable, induces a partial order over time variables. Each "placement" of actions onto specific time points, i.e. a total order, must respect this partial order.

Definition A.2. *Given a (satisfiable) set of temporal constraints* $C(T_1, \ldots T_n)$*, we indicate with* Ω *the (unique minimal)* partial order *induced over the time variables* $T_1, \ldots T_n$*. Given a total order* Θ*, we write* $\Omega \gg \Theta$ *if and only if* $\forall\ i, j\ T_i\ \Omega\ T_j \Rightarrow T_i\ \Theta\ T_j$*, i.e. the total order* fulfills *the partial order.*

For the problem of deriving total orders from the partial order, papers like [31, 119] show how this kind of temporal information can be represented in terms of our kind of temporal constraints, which can be dealt with by efficient constraint solving techniques.

Clearly, every (total) order Θ such that $\Omega \gg \Theta$, satisfies the original set of temporal constraints.

In exploring the possible consequences of an interval action, it is necessary to describe the *significant* time points where it may be executed. This is given by the notion of *placement*, i.e. a substitution for constrained time variables that fulfills the constraints induced by the set $C(T_1, \ldots T_n)$. The notion of placement relies on that of order fulfillment.

The effectiveness of the computational model relies on only reasoning about significant placements, where, as usual, the notion of significance is given according to the deserts and oases approach.

Definition A.3 (Placement). *Let* Ω *be the partial order induced by the set* $C(T_1, \ldots T_n)$*. Let* T_i *be a time variable occurring in an interval action* $happens(a_i, T_i)$.

A placement π *is an assignment from time variables to time values, such that,* $forall\ i, j$ $T_i\ \Omega\ T_j \Rightarrow T_i\pi\ \Omega\ T_j\pi$*, also written as* $\Omega \gg \pi$ *(the placement fulfills the partial order).*

The application of a placement to an interval action is defined as $happens(a, T)\pi = happens(a, T\pi)$, and it transforms an extended theory KB_{TRint} into a ground KB_{TR} one.

A.1.1 Credulously reasoning with interval actions

The definition of placement allows us to define the notion of entailment for KB_{TRint} in terms of that for KB_{TR}, and hence to define the computational model for KB_{TRint} in terms of that for KB_{TR}. These settings add another dimension to the definition of believing something credulously, that now depends both on the existence of a set Δ and a placement π.

Definition A.4 ($\models^{cred}_{TRint}$). *Given a theory* KB_{TRint}*, a placement* π *and a ground fluent literal* $fl[t]$*, then*

$$KB_{TRint} \models^{cred}_{TRint} fl[t] \quad \Leftrightarrow \quad \exists\ \pi\ KB_{TRint}\pi \models^{cred}_{TR} fl[t].$$

Again, an appropriate partition of the time line into deserts and oases helps us in defining an effective computational model for KB_{TRint}. Other than usual oases, the extremes of the intervals of interval actions must be considered as oases. Basically, this is due to the fact that putting an action before or after the query might affect the query itself, and this makes necessary to check integrity constraints at the extremes of an interval, when they occur in the middle of a desert. Moreover, also the time points relative to observation of fluents in KB_0 must be added to the time line as the placement of actions relatively to observations can change the state of affairs that can be derived.

Definition A.5. *Given a fluent literal $fl[t]$ and a set of observations KB_0, containing observed/2, executed/2 and observed/3 predicates, and a set of interval actions referring to intervals $[s_j, e_j]$, the relative* extended time line *ETL is the (maximal) totally ordered sequence*

$$ETL = [0 = t_0, t_1, ..., t_n],$$

where $\forall\ i\ (\exists\ observed(f, t_i) \in KB_0 \vee \exists\ observed(c, a, t_i) \in KB_0 \vee \exists\ executed(a, t_i) \in KB_0 \vee t_i = t \vee \exists\ j\ t_i = s_j \vee t_i = e_j)$.

Finally, we give the definition of *essential placement* and show how proofs can be computed by only taking into consideration essential placements, which, not surprisingly, are finite in number. We write substitution composition as $\bigcirc$, which is a commutative and associative operator for substitutions with disjoint domains and ground ranges, and the empty substitution as ϵ.

Definition A.6 (Essential placement). *Given a theory KB_{TRint}, in which m interval actions (i.e. m temporal variables) occur, a ground fluent literal $fl[t]$, and the relative extended time line $ETL = [0 = t_0, t_1, ..., t_n]$, then a placement π is* essential *if $\pi = \bigcirc_i \pi_i$, with $i \in [0, n]$, and*

1. *$\forall\ i, j \in [0, n]\ dom(\pi_i) \cap dom(\pi_j) = \emptyset\ \wedge\ \forall\ i \in [1, m]\ \pi(T_i) \in [0, \infty)$,*
2. *$\Omega \gg \pi$,*
3. *$\forall\ i$*
   (a) *$\pi_i = \epsilon$, or*
   (b) *$\exists\ k, o\ \pi_i : [T_{i_1}, \ldots, T_{i_k}] \rightarrow [t_i + 1, \ldots, t_i + 1 + o]$, with $o \leq k\ \wedge\ t_i + o \leq t_{i+1}$.*

Intuitively speaking, the definition requires that an essential placements can be partitioned in n placements, one for each desert, such that

1. they are disjoint and π is total over time variables of interval actions in KB_{TRint},
2. π fulfills the partial order induced by temporal constraints in KB_{TRint}, and
3. each π_i either
   (a) is the empty substitution (no actions are mapped in that desert), or
   (b) maps, in any order, a set of k temporal variables into $o \leq k$ contiguous time points starting at the beginning $(t_i + 1)$ of the desert, and not spilling out into the next desert.

It is easy to note that the definition of essential placement encompasses all the possible orders in which interval actions can be placed at the beginning of a desert, respecting the time constraints of the interval actions, but without imposing any further order to the placement of interval actions (which hence can be interleaved in all the possible ways that respect time constraints).

It is worth reminding here, that reasoning with a set of actions whose placements are not beforehand defined, requires to take into consideration all the possible way in which they may influence each other.

The definition of essential placement justify the following proposition which straightforward leads to the computational model.

Proposition A.1. *Given a theory KB_{TRint}, a ground fluent literal $fl[t]$, and a placement π, then*

$$\exists\ \pi\ KB_{TRint}\pi \models_{TR}^{cred} fl[t] \quad \Leftrightarrow \quad \exists\ \tilde{\pi}\ KB_{TRint}\tilde{\pi} \models_{TR}^{cred} fl[t],$$

with $\tilde{\pi}$ an essential placement.

Proposition A.1 provides a computational model for credulously reasoning with interval actions: in order to check for the existence of a placement which allows a fluent literal to be proved, it is sufficient to check for the existence of an essential placement, as done by the following program.

$query\_int\_credulous\_TR(\langle P_{TRint}, C, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A)\ \leftarrow$
$\quad extract\_extended\_oases(KB_0, EO),$
$\quad generate\_the\_representative\_partial\_order(EO, P_{TRint}, PO),$
$\quad generate\_a\_total\_order(PO, TO),$
$\quad generate\_a\_partition\_of\_total\_order\_over\_oases(TO, EO, PTO),$
$\quad generate\_an\_essential\_placements\_from\_a\_partition(PTO, EPL),$
$\quad apply\_essential\_placement(P_{TRint}, EPL, P_{TR}),$
$\quad query\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A).$

$extract\_extended\_oases(KB_0, EO)\ \leftarrow$
works as expected.

$generate\_the\_representative\_partial\_order(EO, P_{TRint}, PO)\ \leftarrow$
produces the minimal partial order induced by the set of temporal constraints C.
This order may contain ground values in order to account for constraints like $3 \leq T_i$.

$generate\_a\_total\_order(P_{TRint}, TO)\ \leftarrow$
generates one (all) the existing total orders fulfilling the partial order.

$generate\_a\_partition\_of\_total\_order\_over\_oases(TO, EO, PTO)\ \leftarrow$
generates one (all) the existing partitions of a (each) total order over deserts, distributing actions over deserts without violating desert "capienza",according to Definition A.6.

$generate\_an\_essential\_placements\_from\_a\_partition(PTO, EPL)\ \leftarrow$
works according to Definition A.6.

$apply\_essential\_placement(P_{TRint}, EPL, P_{TR}) \leftarrow$
works as expected.

$query\_credulous\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A) \leftarrow$
Works as expected. Note that the theory now is ground.
Note also that the number of essential placements that can be generated and passed to this call is finite.

Definition A.7 ($\vdash^{cred}_{TRint}$). *Given the abductive logic program KB_{TRint}, the* computational model *for $\models^{cred}_{TRint}$, indicated as $\vdash^{cred}_{TRint}$, is defined as follows:*

$$KB_{TR} \vdash^{cred^{ngq}}_{TR} fl[t] \Leftrightarrow query\_int\_credulous\_TR(KB_{TRint}, KB_0, fl[t], A).$$

Theorem A.1 ($\models^{cred}_{TRint} \Leftrightarrow \vdash^{cred}_{TRint}$). *Assuming that the abductive proof procedure used in the definition of the predicate $query\_credulous\_TR/4$ is correct and complete, the computational model $\vdash^{cred}_{TRint}$ is correct and complete with respect to the formal model $\models^{cred}_{TRint}$:*

$$KB_{TRint} \models^{cred}_{TRint} fl[t] \Leftrightarrow KB_{TRint} \vdash^{cred}_{TRint} fl[t].$$

A.1.2 Skeptically reasoning with interval actions

Building on credulous reasoning, believing something skeptically is interpreted as the absolute certainty that, not matter which placement is chosen, the fluent of interest holds. Again, the definition of skeptical entailment for KB_{TRint} is given in terms of that for KB_{TR}.

Definition A.8 ($\models^{skep}_{TRint}$). *Given a theory KB_{TRint}, a placement π and a ground fluent literal $fl[t]$, then*

$$KB_{TRint} \models^{skep}_{TRint} fl[t] \Leftrightarrow \forall\, \pi\ KB_{TRint}\pi \models^{cred}_{TR} fl[t] \wedge KB_{TRint}\pi \not\models^{cred}_{TR} \overline{fl[t]}.$$

Again, the notion of essential placement is used to define the computational model for $\models^{skep}_{TRint}$.

Proposition A.2. *Given a theory KB_{TRint}, such that C is satifiable, and a ground fluent literal $fl[t]$, then*

$$\forall\, \pi\ KB_{TRint}\pi \models^{cred}_{TR} fl[t] \Leftrightarrow \forall\, \tilde{\pi}\ KB_{TRint}\tilde{\pi} \models^{cred}_{TR} fl[t],$$

with π placement, and $\tilde{\pi}$ essential placement.

Proposition A.1 provides a computational model for skeptically reasoning with interval actions, as formalised by the following program.

$query\_int\_skeptically\_TR(\langle P_{TRint}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A) \leftarrow$
$extract\_extended\_cases(KB_0, EO)$,
$generate\_all\_essential\_placements(P_{TRinit}, EO, LP)$,
$for\_each\_placement\_query\_skeptically\_TR(LP, \langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A)$.

$for\_each\_placement\_query\_skeptically\_TR([], \langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], []).$

$for\_each\_placement\_query\_skeptically\_TR([P|LP], \langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], [A_1|A_2]) \leftarrow$
$apply\_placement(P_{TRint}, P, P_{TR}),$
$query\_skeptically\_TR(\langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A_1),$
$for\_each\_placement\_query\_skeptically\_TR(LP, \langle P_{TR}, A_{TR}, I_{TR}\rangle, KB_0, fl[t], A_2).$

$generate\_all\_essential\_placements(P_{TRinit}, EO, LP), \leftarrow$
Works as expected, building on the predicated used to non-deterministically construct an essential placement for the case of credulous reasoning.
Note that the number of essential placements is finite, and they can be determined as solutions of the constraint set C.

$apply\_placement(P_{TRint}, P, P_{TR}) \leftarrow$
Works as expected.

Definition A.9 ($\vdash^{skep}_{TRint}$). *Given the abductive logic program KB_{TRint}, the* computational model *for $\models^{skep}_{TRint}$, indicated as $\vdash^{skep}_{TRint}$, is defined as follows:*

$$KB_{TR} \vdash^{skep^{ngq}}_{TR} \ fl[t] \ \Leftrightarrow \ query\_int\_skeptically\_TR(KB_{TRint}, KB_0, fl[t], A).$$

Theorem A.2 ($\models^{skep}_{TRint} \Leftrightarrow \vdash^{skep}_{TRint}$). *Assuming that the abductive proof procedure used in the definition of the predicate $query\_skeptically\_TR/4$ is correct and complete, the computational model $\vdash^{skep}_{TRint}$ is correct and complete with respect to the formal model $\models^{skep}_{TRint}$:*

$$KB_{TRint} \models^{skep}_{TRint} \ fl[t] \ \Leftrightarrow \ KB_{TRint} \vdash^{skep}_{TRint} \ fl[t].$$

Note that a weaker definition of skeptically reasoning over interval actions could be given, namely

Definition A.10 ($\models^{skep}_{TRint}$alt.). *Given a theory KB_{TRint}, a placement π and a ground fluent literal $fl[t]$, then*

$$KB_{TRint} \models^{skep}_{TRint} fl[t] \quad \Leftrightarrow \quad \exists \ \pi \ KB_{TRint}\pi \models^{skep}_{TR} holds\_at(fl, t),$$

that means that at least for a placement, the fluent can be proved skeptically. We adopted the other definition arguing that the essence of skeptical reasoning is to guarantee that the fluent holds under all circumstances, as our definition requires, but we do not exclude that for some particular case, the weaker one may turn out to be more useful. Computationally, the weaker one is one instance of the stronger one, and hence clearly more efficient.

A.2 Reasoning with non-ground queries and interval actions

In order to reason with non-ground queries we can combine in a simple way the approach of the previous section for dealing with interval actions and the one in the section relative to non-ground queries.

Quantified queries are treated as interval actions by considering the time of the query uniformly along with the other existential times in the narrative. Placements also assign the time

variable of the query, which, as interval actions, has associated a constraint with it, namely to belong to the interval specified in the query.

Credulously proving an existential query, then, consists of exhibiting an (essential) placement, among those that correctly place the variable within the interval, such that the query credulously holds, according to the (ground) narrative determined by the placement itself. Following the interpretation given in the previous section, skeptically reasoning for existentially quantified queries means that there exists a point in the query interval, such that all the placements that place the query in that point make the query skeptically hold.

Analogously, credulously (skeptically) reasoning for universally quantified queries, amounts to checking whether, for each point in the query interval, all the placements that place the query in that point make the query credulously (skeptically) hold. This case, therefore requires the full combination of the techniques developed for the treatment of the previous cases and is currently under further investigation.

A.3 Inconsistent theories that need extra unknown event occurrences

In this section we discuss how Assumption 3 can be relaxed exploiting the extension in the previous section A.1. We will present here only an initial study of the problem indicating the broad approach that one can take to tackle this. Further study of this will be carried in the third year of the project in order to fill in some of the missing details.

A narrative KB_0 can render the Temporal Reasoning theory inconsistent as a computee may only partially knowledge of its open environment. We can distinguish two types of inconsistency (see [64]): *classical* inconsistency where KB_0 alone is inconsistent with the integrity constraints of the Temporal Reasoning theory and *frame* inconsistency where the theory is classically consistent but becomes inconsistent when the whole of Temporal Reasoning theory is taken into account. In the later case, the inconsistency arises from the persistence of properties whereas in the first case the observations themselves alone are inconsistent at some time point. In this section we will deal only with the problem of frame inconsistency. The problem of classical inconsistency links with the wide area of Belief Revision which is beyond the scope of this document.

Frame inconsistency arises only if we have fluent observations in the narrative KB_0. We will assume that a computee starts with a consistent KB_0, e.g. when this is empty, and accumulates its observations one by one in this. Each time a new observation is added the computee can check whether its theory is frame consistent or not by asking (before adding the observation) if this is a credulous consequence of the theory so far. If this fails then the theory will become (frame) inconsistent when this observation is added.

The specification of Temporal Reasoning as given in D4 [63] requires, when the theory is frame inconsistent, to first find all minimal explanations of the observations in the current KB_0 and then reason with each of the theories obtained by adding one explanation in KB_0, e.g. for skeptical conclusions we will need to be able to derive a query skeptically in all such extended theories. In our case here, given the simplification that observations are accummulated one by one and that we can detect the first time the theory becomes inconsistent, we are going to simplify this requirement and ask only that we find all minimal explanations of the *culprit observation* Obs^{inc} just acquired.

Definition A.11. *Let $T = \langle P_{TR}, A_{TR}, I_{TR}\rangle$ be a temporal reasoning theory and KB_0 its current narrative. Then $Obs^{inc} \in KB_0$ is a* culprit *observation of T iff*

- *the theory T is frame inconsistent,*
- *the theory T', obtained from T by deleting from KB_0 the observation Obs^{inc}, is frame consistent,*
- *there is no observation in KB_0 which satisfies the second bullet condition and whose time is greater than that of Obs^{inc}.*

The last condition here can be restrictive as it will force us to recover from the inconsistency only via such latest observations. Note also that in general we need to extend this definition to allow for sets of culprit observations.

We will assume that every Temporal Reasoning theory contains additionally the following domain independent rules:

$$initiates(start(F), T, F).$$
$$terminates(stop(F), T, F).$$

for any ground literal F, where $start(F)$ and $stop(F)$ are two actions not used in the domain dependent part of the theory and have no other effect rules. The theory also contains he bridge rule

$$happens(A, T) : -ass\_happens(A, T).$$

Definition A.12. *Let $T = \langle P_{TR}, A_{TR}, I_{TR}\rangle$ be a temporal reasoning theory, Obs^{inc} a culprit observation of this and P'_{TR} the theory obtained by deleting Obs^{inc} from P_{TR}. Then an explanation, $\langle E(Obs^{inc}), C\rangle$, of Obs^{inc} is a set of abducibles $E(Obs^{inc})$ drawn from the extended set:*

- $A^{Ext}_{TR} = A_{TR} \cup$
  $\{ass\_happens(start(F), T) | F\ ground\ literal,\ T\ a\ ground\ time\ or\ an\ existential\ variable\} \cup$
  $\{ass\_happens(stop(F), T) | F\ ground\ literal,\ T\ a\ ground\ time\ or\ an\ existential\ variable\}$

together with a set of temporal (interval) constraints, C, on the time variables appearing in $E(Obs^{inc})$ such that, for every valuation σ of these time variables satisfying C ($\sigma \models C$):

1. $P'_{TR} \cup E(Obs^{inc})\sigma \models_{LP} Obs^{inc}$,
2. $P'_{TR} \cup E(Obs^{inc})\sigma \models_{LP} I_{TR}$.

Note that these explanations do not depend essentially on assumptions *assume_holds* as the start and stop actions and their effect laws do not have any preconditions. In other words, when we add to the theory the part of the explanation relating to *ass_happens* only then the culprit observation will be a skeptical consequence of this extended theory.

The previous definition of *explanation*, naturally extends to the case in which $T = \langle P_{TR}, A_{TR}, I_{TR}\rangle$ is a theory already containing one (or more) explanation of one (or more) different culprit observation, $\overline{Obs^{inc}}$. Indeed, such explanation $E(\overline{Obs'^{inc}})$ consists of a set of $ass\_happens(start(F), T)$ and $ass\_happens(stop(F), T)$ predicates together with a set of temporal constraints C. Note, from the bridge rule, that assuming that an action happened works as if the action actually occurred. Coherently with the Definition A.12 $E(Obs^{inc})$ is then an explanation for the culprit observation Obs^{inc}, if $T \cup E(Obs^{inc}) \models^{skep}_{TRint} Obs^{inc}$, that is, for all the possible (essential) placements for $T \cup E(Obs^{inc})$, satisfying the set of temporal constraints C , the ground culprit observation skeptically holds.

We can therefore adopt the more general definition of explanation of culprit observations as follows.

Definition A.13. *Let $T = \langle P_{TR}, A_{TR}, I_{TR}\rangle$ be a temporal reasoning theory, Obs^{inc} a culprit observation of this and T' the theory obtained by deleting Obs^{inc} from P_{TR}. Then an explanation, $\langle E(Obs^{inc}), C\rangle$, of Obs^{inc} is a set of abducibles $E(Obs^{inc})$ drawn from the extended set:*

- $A_{TR}^{Ext} = A_{TR} \cup$
  $\{ass\_happens(start(F), T) | F$ *ground literal*, T *a ground time or an existential variable*$\}\cup$
  $\{ass\_happens(stop(F), T) | F$ *ground literal*, T *a ground time or an existential variable*$\}$

together with a set of temporal (interval) constraints, C, on the time variables appearing in $E(Obs^{inc})$ such that:

- $T' \cup E(Obs^{inc}) \models^{skep}_{TRint} Obs^{inc}$.

Such explanations are required to be *minimal* and in addition we will assume here that they are *causal*.

Definition A.14. *Given an observation Obs^{inc} an explanation E is* causal *iff the time variables that appear in the ass_happens abducibles contained in E are all constraint to be earlier than the time of Obs^{inc}.*

For minimality in the general case we will adopt the following definition.

Definition A.15. *Given an observation Obs^{inc}, an explanation $\langle E(Obs^{inc}), C\rangle$ is* minimal *iff there exists no explanation, $\langle E(Obs^{inc})', C'\rangle$ of Obs^{inc}, such that:*

- *for any ground literal F, $E(Obs^{inc})'$ contains a strictly smaller number of $ass\_happens(start(F), T')$ and of $ass\_happens(stop(F), T')$ abducilbes as $E(Obs^{inc})$ does,*
- *or, if they contain the same number of such abducibles, there exists $ass\_happens(start(F), T') \in E'$ and $ass\_happens(start(F), T) \in E$ (or $ass\_happens(stop(F), T') \in E'$ and $ass\_happens(stop(F), T) \in E$) where, if I is the set of values given to the variable T by all valuations σ of C and analogously I' is the set of values given to the variable T' by all valuations σ' of C', then $I \subset I'$.*

In other words, an explanation is minimal iff whenever it makes an assumption of some start or stop event occurrence it allows the time of this occurrence to be as big as possible.

Causal minimal explanations are unique when the given narrative does not contain any non-ground events.

We will introduce here a final simplification that will allow us to develop a simple algorithm for computing explanations. We will assume that there are no observations in the narrative whose time is later than the culprit observation(s).

Then we can simplify the notion of minimality and require that explanations contain only *one* action of $stop(F)$ or $start(F)$ depending on the culprit observation. Minimality then just reduces to ensuring that the interval constraint for this action is maximal, in fact maximum..

Given the above analysis and assumptions, minimal causal explanations can be computed directly from their specification as follows:

Let $T = \langle P_{TR}, A_{TR}, I_{TR}\rangle$ be a temporal reasoning theory, Obs^{inc} a culprit observation of this and T' the theory obtained by deleting Obs^{inc} from P_{TR}. Let TL_π denote be the time line

of the theory under some essential placement π of its non-ground actions in KB_0. If no such actions exist the time line is drawn from all the events and observations in T.

Let $Obs^{inc} = observed(f, t)$ where f and t are a ground fluent and time point respectively (the case of $observed(f, t)$ is analogous):

- **Base case: Ground time line** $TL = [t_0, t_1, ..., t_n, t]$ **-** Start from $\langle E(Obs^{inc}), C\rangle = \langle\{ass\_happens(stop(f), t_s)\}, \{t_s \in [s_i, t-1]\}\rangle$ with $s_i = t_0$ and move the value of s_i successively to t_i until $T' \cup E(Obs^{inc}) \vdash^{cred}_{TRint} Obs^{inc}$ or have reached t in which case the algorithm fails to find an explanation.

- **General case:** T' **contains interval actions of** *stop* **and** *start* **-** Let t_0 be the greatest time between the time of the latest observation of f in T' and the latest lower-bound of the $start(f)$ actions in T'. If no such observations of actions exist in T' then $t_0 = 0$. [34]

  Take any placement of T' and let $TL_s = [t_0 + 1, t_1, ..., t_n, t]$ be the ground time (sub)line from $t_0 + 1$ onwards. Apply the base case on this time line.

Note that in the algorithm we are using the credulous reasoning with interval actions as the assumptions *assume_holds* do not affect the effects of start and stop actions and hence if the observation is credulously derived it will also be sceptically derived.

In general, we will assume that the computee selects one explanation for a culprit observation (maybe under some preference criteria) and commits to this, i.e. we are relaxing further the requirement of D4 of finding all explanations and reasoning with all of these simultaneously. We assume that if the explanation to which the computee has committed is false then the computee will realize this sometime in the future and revise this explanation away. The details of this are beyond the scope of this document.

We are currently studying how to extend the algorithm for computing explantions of culprit observations when we relax some of the assumptions made above.

A.4 An example about the use of the extensions of $\vdash_{TR}$

We report here an example illustrating how Temporal Reasoning deals with narratives with existential actions.

Example 16 (Narrative with existential actions, ground queries). *Let us reconsider Example 3, where the Temporal Reasoning capability was required to prove (skeptically) the query* $holds\_at(booked\_room, T), 50 < T,\ T < 50 + 300$*, against the following (inconsistent !) narrative:*

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$
$$observed(neg(booked\_room), 60).$$

Let us suppose, as in the original example, that the narrative is completed with a recovery *action, able to explain why the room, that was booked at time* 40*, is not booked anymore at time* 60*. We do not enter here in the problem of how this action has been generated (see section A.3).*

[34] Note that for any essential placement π in T' the time line after t_0 is such that all observations and actions relating to the fluent f are placed at the same times.
Note also that the placement of the actions for other fluents does not affect the explanation for $Obs^{inc} = observed(f, t)$.

The action does not have a fix time associated with it. Note that, informally speaking, this existentially quantified action represents a minimal *explanation of the inconsistency, encompassing all the possible occurrence of the action that may have occur between time* 40 *and* 60*. This justifies the importance of being able to deal with interval actions in the theory.*

The narrative is then

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$
$$observed(neg(booked\_room), 60).$$
$$executed(cancel\_reservation, T1), 40 \leq T1,\ T1 \leq 60.$$

where T *is an existentially quantified variable, together with the domain dependent knowledge*

$$terminates(cancel\_reservation, T, booked\_room).$$

Hence here $cancel\_reservation$ *is a realization of the abstract action* $stop(booked\_room)$ *presented in section A.3.*

Let us consider the time line $[0, 10, 20, 40, 50, 60, 350]$*, which also includes the extremes of the query interval, and apply the computational model for non-ground actions. Note that the possible interleaving between the placed action and the ground goal will be taken into considerations by the cases of total temporal ordering between the two corresponding variables. In addition, the different cases relative to putting the action (and the query) in the various deserts will be taken into consideration by exhaustively generating all the possible placements (for each total order), as done in the following.*

The partial order induced by the constraints $50 < T,\ T < 50 + 300$ *and* $40 \leq T1,\ T1 \leq 60$ *does not impose any ordering between the two variables, then the following three cases must be taken into consideration in order to generate the possible essential placements.*

$T1 < T$ *There are two essential placements fulfilling the total order and the set of constraints:* $\tilde{\pi}1 = \{T1 = 41, T = 51\}$ *and* $\tilde{\pi}2 = \{T1 = 41, T = 61\}$*. Let us consider the first one, and the corresponding ground narrative:*

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$
$$observed(neg(booked\_room), 60).$$
$$executed(cancel\_reservation, 41).$$

In this case, the now ground query $holds\_at(neg(booked\_room), 51)$*, can be skeptically proved, according to the basic case of the computational model, the one operating with ground narrative and ground query. It is easy to see that placing forward the action in the current desert* $[41, 50]$*, according to the time line, does not change the obtained result.*

The case for $\tilde{\pi}2$ *works exactly the same.*

$T1 = T$ *The only possible essential placement is* $\tilde{\pi}3 = \{T1 = 51, T = 51\}$*, and the narrative is*

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$
$$observed(neg(booked\_room), 60).$$
$$executed(cancel\_reservation, 51).$$

Again the ground query $holds\_at(neg(booked\_room), 51)$*, can be skeptically proved.*

$T1 > T$ *In this case there is again only one essential placement,* $\tilde{\pi}4 = \{T1 = 52, T = 51\}$*. In this case, however, with the narrative*

$$executed(switch\_on, 10).$$
$$observed(neg(light), 20).$$
$$observed(booked\_room, 40).$$
$$observed(neg(booked\_room), 60).$$
$$executed(cancel\_reservation, 52).$$

the ground query $holds\_at(neg(booked\_room), 51)$*, can not be skeptically (nor credulously) proved.*

It follows that not for all the possible essential placements, the query skeptically holds, due to the case of $\tilde{\pi}4$*, and hence the query*

$$holds\_at(booked\_room, T), 50 < T,\ T < 50 + 300$$

is not skeptically entailed by the narrative containing the interval action

$$executed(cancel\_reservation, T1), 40 \leq T1,\ T1 \leq 60.$$

B Proofs for Proof Procedures

B.1 Proofs for C-IFF

B.1.1 Auxiliary lemmas

To prove the soundness results of Theorems 10.3 and 10.4, we first need to show that our proof rules are *equivalence preserving* in the sense that the frontier F_{i+1} obtained from a frontier F_i through the application of a proof rule is always logically equivalent to F_i. As a proof rule affects only the node which is applied to, the above result can be obtained showing that the disjunction of the nodes obtained through the application of a proof rule is always logically equivalent to the node that rule has been applied to.

Lemma B.1 (Equivalence preservation). *If* N *is a node in a derivation with respect to the theory* $Comp_{\overline{A} \cup \overline{C}}(P)$*, and* $\mathcal{N}$ *is the disjunction of the immediate successor nodes of* N *in that derivation,*[35] *then* $Comp_{\overline{A} \cup \overline{C}}(P) \models^3_{\Re} N \leftrightarrow \mathcal{N}$*.*

Proof. We are going to sketch proofs for three separate claims which together entail the claim of the lemma:

(1) Let N be a node and let N' be its successor node, obtained by the application of one of the unfolding rules with respect to $Comp_{\overline{A} \cup \overline{C}}(P)$. Then $Comp_{\overline{A} \cup \overline{C}}(P) \models^3_{\Re} N \leftrightarrow N'$ holds.

(2) Let N be a node and let $\mathcal{N}$ be the disjunction of nodes obtained by applying either splitting or case analysis for equalities to N. Then $\models^3_{\Re} N \leftrightarrow \mathcal{N}$ holds.

[35] Note that the disjunction $\mathcal{N}$ will have only a single disjunct whenever the rule applied to N is neither splitting nor case analysis for equalities.

(3) Let N be a node and let N' be the successor node of N, obtained by applying any of the remaining rules. Then $\models^3_{\Re} N \leftrightarrow N'$ holds.

It is easy to see that these three claims together entail the claim of the lemma as equivalence of two formulas entails equivalence of these two formulas with respect to a particular theory (such as $Comp_{\overline{A} \cup \overline{C}}(P)$). Also note that the disjunction $\mathcal{N}$ referred to in the lemma will be unary and only contain the single successor node N' of N for all proof rules except splitting and case analysis for equalities, i.e. for most rules the statement of the lemma reduces to the statement that $Comp_{\overline{A} \cup \overline{C}}(P) \models^3_{\Re} N \leftrightarrow N'$ holds for *the* successor node N' of N.

Equivalence preservation can easily be verified for most rules. In particular, claim (1) immediately follows from the fact that applying an unfolding rule amounts to rewriting a subformula in N according to one of the equivalences in $Comp_{\overline{A} \cup \overline{C}}(P)$. Claim (2) covers the two rules that generate more than one successor node. In the case of the splitting rule, the claim simply states the fact that distributing a disjunction to the outside is an equivalence preserving operation. For the case analysis rule for equalities, a node containing an implication of the form $X = t \wedge A \rightarrow B$, where X is an existentially quantified variable and t is not, a universally quantified variable itself (in which case another rule would apply), is split depending on whether there exist values for the universally quantified variables in t such that $X = t$ holds or whether this is not the case. To see that this is also an equivalence preserving operation, consider that there can be at most one such vector of values (for any given value of X).

For the proof of claim (3), we are going to consider here only those rules that differ from the original framework of Fung and Kowalski [47], in particular case analysis for constraints and constraint solving. Recall that the former replaces an implication of the form $Con \wedge A \rightarrow B$ with the disjunction $[Con \wedge (A \rightarrow B)] \vee \overline{Con}$, provided the constraint Con does not contain any universally quantified variables. The following sequence of transformations establishes equivalence preservation of the rule of case analysis for constraints:[36]

$$\begin{array}{ll} & Con \wedge A \rightarrow B \\ \equiv & Con \rightarrow (A \rightarrow B) \\ \equiv & Con \rightarrow (Con \wedge (A \rightarrow B)) \\ \equiv & \neg Con \vee (Con \wedge (A \rightarrow B)) \\ \equiv & (Con \wedge (A \rightarrow B)) \vee \overline{Con} \end{array}$$

We now turn to the constraint solving rule and the two related rules for rewriting equalities and disequalities as constraints. The two rewrite rules are equivalence preserving, because they merely tag an equality or disequality as a formula that may be submitted to the constraint solver. By checking that one of the arguments of the (dis)equality in question already appears within a constraint, we ensure that the input to the constraint solver is always well-defined.[37] The constraint solving step itself replaces a set of constraint atoms by $\bot$ whenever that set of constraints is not satisfiable. This is clearly an equivalence preserving operation. In case unsatisfiability cannot be established, the constraint solving rule has no effect, i.e. it is certainly equivalence preserving also in this case.

Equivalence preservation for the remaining proof rules of the procedure may be checked in a similar manner. □

[36] These transformations are applied to the matrix of the first formula, i.e. we do not need to take quantifiers into account. Note that the argument for the case analysis rule for equalities is slightly more complex as it can involve changes in quantification.

[37] At this point, we rely on the fact that the input $\langle Comp_{\overline{A} \cup \overline{C}}(P), IC, Q \rangle$ has been well-defined in the first place; in particular, no non-constraint term occurs in an argument position reserved for constraint arguments.

We are now able to show that, with respect to the input theory P, both the integrity constraints IC and the initial query Q will be logical consequences of any success node:

Lemma B.2 (Final nodes entail initial node). *If N is a final success node for the input $\langle P, IC, Q\rangle$, then $Comp_{\overline{A}\cup\overline{C}}(P) \models^3_{\Re} N \rightarrow (IC \wedge Q)$.*

Proof. Observe that $IC \wedge Q$ is the initial node of any derivation. The claim then follows by induction over the number of proof steps leading to the final node N. For the induction step we exploit the equivalence preservation result proved in Lemma B.1. □

The third and final lemma required relates to answer extraction:

Lemma B.3 (Answer extraction). *If N is a final success node and Δ is the set of abducible atoms in N, then there exists a substitution σ such that $Comp(\Delta\sigma) \models^3_{\Re} N\sigma$.*

Proof. Let $\langle \Delta, \Phi, \Gamma\rangle$ be the answer extracted from the node N. By construction, there exists a substitution σ satisfying both Φ, the set of equalities and disequalities, and Γ, the set of constraints,[38] i.e. we have $\models^3_{\Re} \Phi\sigma$ and $\models^3_{\Re} \Gamma\sigma$. Hence, we certainly also have $Comp(\Delta\sigma) \models^3_{\Re} \Phi\sigma$ and $Comp(\Delta\sigma) \models^3_{\Re} \Gamma\sigma$. Of course, we also have $Comp(\Delta\sigma) \models^3_{\Re} \Delta\sigma$. Besides the abducible atoms in Δ, the (dis)equalities in Φ, and the constraints in Γ, the node N will typically also contain a number of other formulas. The claim of the lemma is that, under the substitution σ, these additional formulas are redundant in the sense of being satisfied by $Comp(\Delta\sigma)$ alone.

Because N is a final node, all proof rules will have been applied exhaustively. This allows us to narrow down the range of formulas in N:

- It does not contain any disjunctions (splitting).
- It does not contains any defined predicates as atoms (unfolding atoms).
- It does not contain either $\top$ or $\bot$ occurring as an atom (logical simplification and the fact that N is not a failure node, respectively).
- It does not contain any implications with defined predicates in the antecedent (unfolding within implications).
- It does not contain any implications with negative literals in the antecedent (negation rewriting).
- It does not contain any implications with $\top$ or $\bot$ in the antecedent (logical simplifications).

Also note that there can be no implications in N where all conjuncts in the antecedent are either equalities or constraints (the only exception are formulas encoding disequalities and these are contained in the set Φ). Otherwise, either one of the case analysis rules, equality rewriting, or the substitution rule for implications would have applied; or the dynamic allowedness rule would have been triggered. The latter contradicts our assumption of N being a success node.

Hence, the only type of formula in N (except those in $\Delta \cup \Phi \cup \Gamma$) are implications where the antecedent includes an abducible atom and no negative literals. To prove the claim of the lemma for this type of implication, we are going to distinguish two cases:

[38] The substitution rule for atoms guarantees the existence of a substitution satisfying Φ. That σ satisfies Γ depends on our assumption on the availability of a *complete* constraint solver. The fact that both Φ and Γ can be satisfied by the *same* substitution follows from the way in which we rewrite (dis)equalities as constraints.

(1) The propagation rule has been applied to the implication in question.

(2) The propagation rule has not been applied to the implication in question.

Let us first consider case (1). Suppose $p(\vec{X}) \wedge A \rightarrow B$ is an implication in N and the propagation rule has been applied to it with respect to the abducible p. Here $\vec{X}$ represents the vector of universally quantified variables occurring anywhere in p. Furthermore, let Δ_p denote the set of instances of p in Δ. The propagation rule must have been applied to $p(\vec{X}) \wedge A \rightarrow B$ with respect to *every* member of Δ_p. The residues of these rule applications are of the form $\vec{X} = \vec{t} \wedge A \rightarrow B$. By Lemma B.1, each of these residues, if it is not itself a member of N, will have been replaced by a set of formulas entailing the residue in question. By the nature of our proof rules, if any of these formulas is itself an implication, it must be an implication with fewer abducible atoms in the antecedent than $p(\vec{X}) \wedge A \rightarrow B$. But given $Comp(\Delta_p\sigma)$, the implication $[p(\vec{X}) \wedge A \rightarrow B]\sigma$ follows from the set of residues of the form $[\vec{X} = \vec{t} \wedge A \rightarrow B]\sigma$, i.e. we have reduced the problem to showing that the claim of the lemma holds for any implication with fewer abducible atoms in the antecedent than $p(\vec{X}) \wedge A \rightarrow B$. Antecedents are of course finite; hence, the claim follows by induction.

Now suppose case (2) applies, i.e. suppose N contains an implication with an abducible atom as a conjunct in the antecedent to which the propagation rule has *not* been applied. This is only possible if Δ does not contain an instance of this atom. But then the implication will be a logical consequence of $Comp(\Delta\sigma)$ under any substitution σ. This concludes our proof of Lemma B.3. □

Observe that *factoring* is the only the proof rule which we did *not* have to appeal to in the proof of Lemma B.3. Indeed, factoring is not required to ensure soundness. However, as can easily be verified, factoring is equivalence preserving in the sense of Lemma B.1; that is, our soundness results presented in the sequel apply both to the system with and to the system without the factoring rule.

B.1.2 Soundness of success

Proof of Theorem 10.3. A derivation is successful if it yields a (at least one) success node N. Let $\langle \Delta, \Phi, \Gamma \rangle$ be the answer extracted from that node. By Lemma B.3, there exists a substitution σ such that $Comp(\Delta\sigma) \models^3_{\Re} N\sigma$.

Together with Lemma B.2 this entails $Comp_{\overline{A} \cup \overline{C}}(P) \cup Comp(\Delta\sigma) \models^3_{\Re} (IC \wedge Q)\sigma$. As σ is, by definition, a substitution over the set of all the variables which occur in abducible atoms and constraint literals in N, we have that
$Comp_{\overline{A} \cup \overline{C}}(P) \cup Comp(\Delta\sigma) \equiv Comp(P \cup \Delta\sigma)$
. So, we also have that $Comp(P \cup \Delta\sigma) \models^3_{\Re} (IC \wedge Q)\sigma$ which reduces to $Comp(P \cup \Delta\sigma) \models^3_{\Re} IC \wedge Q\sigma$, because there are no free variables in IC.

Now let σ' be the restriction of σ to variables occurring in the query Q, and write Δ' for the set $\Delta\sigma$, i.e. Δ' is a (finite) set of ground abducible atoms. We obtain $Comp(P \cup \Delta') \models^3_{\Re} IC \wedge Q\sigma'$. Hence, there does indeed exist an answer $\langle \Delta', \sigma' \rangle$ to the query Q given the abductive logic program $\langle Th, IC \rangle$. □

B.1.3 Soundness of failure

Proof of Theorem 10.4. Let $\mathcal{N}$ be the disjunction of all the leaf nodes in the derivation. Using Lemma B.1 and by induction over the number of proof steps in a derivation, we can prove $Comp_{\overline{A}\cup\overline{C}}(P) \models^3_{\Re} (IC \wedge Q) \leftrightarrow \mathcal{N}$. Now recall that a node is called a failure node iff it is logically equivalent to $\bot$, i.e. we get $Comp_{\overline{A}\cup\overline{C}}(P) \models^3_{\Re} (IC \wedge Q) \leftrightarrow \bot$. Hence, $Comp_{\overline{A}\cup\overline{C}}(P)$ $cupIC \models^3_{\Re} (Q \leftrightarrow false)$, that is, there can indeed be no answer for the query Q given the abductive logic program $\langle P, IC\rangle$. □

B.2 Proofs for $LPwNF$

The proofs are adapted from [75].

Proof of Theorem 10.6 (Soundness)

Suppose there is a successful derivation computing Δ from Δ_0. By definition of derivation, it follows that $\Delta_0 \subseteq \Delta$. We will show that:

- *"Δ is not self-attacking, i.e. it is consistent."* Note that, if Δ is self-attacking and N is a defence node, then $closure(N) \cap culprits(\Delta) \neq \emptyset$ and thus no T_{i+1} can be built.
- *"If A attacks Δ then Δ attacks A."* Assume, by way of contradiction, that A attacks Δ but Δ does not attack A. Since the attacking relation is compact, it follows that there exists a minimal subset A' of A that attacks Δ. However, by definition of a successful derivation, A' is marked, thus, there exists an argument $D' \subseteq \Delta$ against a culprit $c \in closure(A')$ such that D' attacks A'.

□

Proof of Theorem 10.7 (Completeness)

We show that there is a successful derivation $\mathcal{T}_0, ..., \mathcal{T}_n$ computing Δ' from Δ_0, such that $\Delta_0 \subseteq \Delta' \subseteq \Delta$ and Δ' is admissible. Let Δ_i denote the root of $\mathcal{T}_i$. We construct a derivation $\mathcal{T}_0, ..., \mathcal{T}_n$ so that $\Delta_i \subseteq \Delta$ for all i, $0 \leq i \leq n$:

- $\mathcal{T}_0$ consists only of the (unmarked) root Δ_0, labelled as defence.
- Given $\mathcal{T}_i$ $(i \geq 1)$, let N be any unmarked node in $\mathcal{T}_i$. Then, $\mathcal{T}_{i+1}$ is obtained as follows:
  - If N is an attack node, let D' be a minimal subset of Δ such that D' attacks against N. Then, D' is added as the (unmarked) child of N in $\mathcal{T}_{i+1}$, labelled as defence, and N is marked. Moreover, if c is the culprit in the closure of N, then c is recorded as the culprit of N in $\mathcal{T}_{i+1}$.
  - If N is a defence node and $closure(N) \cap culprits(T_i) = \emptyset$, then $\mathcal{T}_{i+1}$ is $\mathcal{T}_i$ where N is marked, the root is extended by N, and if $A_1, ..., A_m (m \geq 0)$ are all minimal attacks against N then $A_1, ..., A_m$ are added as additional (unmarked) attack nodes children of the root.

Let Δ' be the root of $\mathcal{T}'$. By soundness, Δ' is admissible and $\Delta_0 \subseteq \Delta'$. Therefore, we need only prove that a minimal defence $D' \subseteq \Delta$ exists for every attack node N in $\mathcal{T}_i$. By construction

of $\mathcal{T}_i$, it holds that $\Delta_i \subseteq \Delta$ and thus, by monotonicity of the attacking relation, N attacks Δ. It follows that Δ attacks N since Δ is an admissible set. By compactness of the attacking relation, it is concluded that there exists a subset D' of Δ that attacks N.

□

C Proofs for Capabilities

C.1 Proofs for Planning

We sketch the proof of soundness of the computational model devised above for a single goal. In this case, we can prove that, whenever $\vdash^{\tau}_{plan}$ returns an answer which is not $\perp, \perp$, conditions (i), (ii) and (iii) in the specification of planning (see Section 11.1.1) are satisfied by the answer. These conditions are simplified as follows when a single goal G is taken into account.

Specification of $\models^{\tau}_{plan}$ for a single goal

Let $S = \langle KB, Goals, Plan\rangle$ be a state, and G be a mental goal $\langle l[t], G', Tc\rangle$. Then:

$$KB, Plan, Goals, \{G\} \models^{\tau}_{plan} \quad \{\langle G, \mathcal{A}s, \mathcal{G}s\rangle\}$$

where,

- either $\mathcal{A}s = \mathcal{G}s = \perp$,
- or

  $\mathcal{A}s = \{(a_1[t_1], T_1), \ldots, (a_m[t_m], T_m)\}$, $m \geq 0$, each $a_i[t_i]$ is a timed operator and T_i are temporal constraints and

  $\mathcal{G}s = \{(l_1[t_1], S_1), \ldots, (l_k[t_k], S_k)\}$, $k \geq 0$, each $l_i[s_i]$ is a timed literal, and S_i are temporal constraints,

such that:

(i) if $\mathcal{T}$ is the set of all temporal constraints in $G, \mathcal{A}s, \mathcal{G}s$, together with additional constraints ensuring that each new action must be executable in the future, namely
$\mathcal{T} = Tc \cup \bigcup_{i=1,\ldots m} T_i \cup \bigcup_{i=1,\ldots k} S_i \cup \bigcup_{i=1,\ldots m} t_i > \tau$
then there exists a total $\Sigma-$valuation σ such that $\sigma \models_{\Re} \mathcal{T} \cup TCS$

(ii) $P_{plan} \wedge [\bigwedge_{i=1,\ldots,m} assume\_happens(a_i, t_i) \wedge \mathcal{EC}(Plan) \wedge \bigwedge_{\ell=1,\ldots,k} holds(l_\ell, s_\ell) \wedge$
$\mathcal{EC}(Goals \setminus \{Gs\})]\sigma \models_{LP} holds\_at(l, t)\sigma$

(iii) $P_{plan} \wedge \bigwedge_{i=1,\ldots,m} assume\_happens(a_i, t_i) \wedge \mathcal{EC}(Plan)]\sigma \wedge \bigwedge_{i=1,\ldots,k} holds\_at(l_i, s_i) \wedge$
$\mathcal{EC}(Goals)]\sigma \models_{LP} I_{plan}$

Hence we basically need to ensure that, whenever an answer $\mathcal{G}s, \mathcal{A}s$ is returned, conditions (i), (ii) and (iii) above are satisfied.

Lemma C.1.
Let

$$\{\langle l[t], \_, Tc\rangle\} \vdash^{\tau}_{plan} \{\langle l_1[s_1], S_1\rangle, \ldots, \langle l_k[s_k], S_k\rangle\}, \{\langle a_1[t_1], T_1\rangle, \ldots, \langle a_m[t_m], T_m\rangle\}$$

and let

$$\mathcal{T} = Tc \cup \bigcup_{i=1,\ldots m} T_i \cup \bigcup_{j=1,\ldots k} S_j \cup \bigcup_{i=1,\ldots m} t_i > \tau$$

Then there exists a total valuation σ such that

$$\sigma \models \mathcal{T} \wedge TCS.$$

Proof The proof follows immediately from the soundness of CIFF and the following observations:

- each equality $t' = \tau'$ in Σ is contained by construction in the initial query;
- each conjunct of TCS is contained in the initial query;
- for each $i = 1, \ldots, k$ (resp. $j = 1, \ldots, m$) the constraint T_i (resp. S_j) is contained in the answer returned by CIFF;
- for each $i = 1, \ldots, m$, the constraint $t_i > \tau$ is satisfied, since all the constraints in I^+_{plan} are contained in the initial query and so the constraint (iii) added to I^+_{plan} is satisfied for each action returned in the answer. □

The above Lemma guarantees that condition (i) in the specification is satisfied.

Given a set of goals $\mathcal{G}$ and a set of actions $\mathcal{A}$ we denote by $\mathcal{AEC}(\mathcal{G})$, $\mathcal{EC}(\mathcal{G})$, $\mathcal{AEC}(\mathcal{A})$ and $\mathcal{EC}(\mathcal{A})$ the following sets:
$\mathcal{AEC}(\mathcal{G}) = \{assume\_holds(l, s) \mid \langle l[s], \_, \_\rangle \in \mathcal{G}\}$
$\mathcal{EC}(\mathcal{G}) = \{holds(l, s) \mid \langle l[s], \_, \_\rangle \in \mathcal{G}\}$
$\mathcal{AEC}(\mathcal{A}) = \{assume\_happens(a, t) \mid \langle a[t], \_, \_, \_\rangle \in \mathcal{A}\}$.
$\mathcal{EC}(\mathcal{A}) = \{happens(a, t) \mid \langle a[t], \_, \_, \_\rangle \in \mathcal{A}\}$.

Given a set $\mathcal{G}$ (resp. $\mathcal{A}$) containing pairs of the form $\langle l[t], Tc\rangle$ (resp. $\langle a[t], Tc\rangle$), where $l[t]$ is a timed fluent literal (resp. where a is a timed action operator) $\mathcal{AEC}(\mathcal{G})$ and $\mathcal{EC}(\mathcal{G})$ (resp. $\mathcal{AEC}(\mathcal{A})$ and $\mathcal{EC}(\mathcal{A}$ are defined in a similar way.

Lemma C.2.
Let

$$\{\langle l[t], \_, Tc\rangle\} \vdash^{\tau}_{plan} \mathcal{G}s, \mathcal{A}s$$

Then

$$P^{plan} \cup \mathcal{AEC}(Plan) \cup \mathcal{EC}(Goals \setminus \{G\}) \cup \mathcal{EC}(\mathcal{G}s) \cup \mathcal{AEC}(\mathcal{A}s) \models_{LP} holds\_at(l, t).$$

Proof By soundness of CIFF we have

$$P^+_{plan} \cup \mathcal{AEC}(Plan) \cup \mathcal{AEC}(Goals \setminus \{G\}) \cup \mathcal{AEC}(\mathcal{G}s) \cup \mathcal{AEC}(\mathcal{A}s) \models_{LP} holds\_at(l, t).$$

Moreover, since P^+_{plan} contains the rules

$holds\_at(P,T) \leftarrow assume\_holds(P,T)$
$holds\_at(not\ P,T) \leftarrow assume\_holds(not\ P,T)$
and $assume\_holds$ atoms occur nowhere else in the rules of P^+_{plan}, we also have

$$P^+_{plan} \cup \mathcal{AEC}(Plan) \cup \mathcal{EC}(Goals \setminus \{G\}) \cup \mathcal{EC}(\mathcal{G}s) \cup \mathcal{AEC}(\mathcal{A}s) \models_{LP} holds\_at(l,t).$$

Hence, by construction of P^+_{plan} we have clearly

$$P_{plan} \cup \mathcal{AEC}(Plan) \cup \mathcal{EC}(Goals \setminus \{G\}) \cup \mathcal{EC}(\mathcal{G}s) \cup \mathcal{AEC}(\mathcal{A}s) \models_{LP} holds\_at(l,t).$$

□

Lemma C.3.
Let

$$\{\langle l[t], \_, Tc\rangle\} \vdash^{\tau}_{plan} \mathcal{G}s, \mathcal{A}s$$

Then

$$P_{plan} \cup \mathcal{AEC}(Plan) \cup \mathcal{EC}(Goals \setminus \{G\}) \cup \mathcal{EC}(\mathcal{G}s) \cup \mathcal{AEC}(\mathcal{A}s) \models_{LP} IC_{plan}$$

Proof The proof follows from soundness of CIFF, the observation that P^+_{plan} is obtained from P_{plan} by adding the rules
$holds\_at(P,T) \leftarrow assume\_holds(P,T)$
$holds\_at(not\ P,T) \leftarrow assume\_holds(not\ P,T)$
(as in the proof of the previous Lemma), and finally from the fact that $IC_{plan} \subset IC^+_{plan}$. □

C.2 Proofs for Temporal reasoning

Proof of Proposition 11.1

(Sketch)
⇒ Trivial.

⇐
The atom $holds\_at(fl, t_i + 1)$ holds either because

- an action happened in the past that initiated (resp.,terminated, if the fluent is negative) it, and it has not been clipped (declipped) in the meantime, or
- it has been observed in the past and not clipped (declipped) in the meantime, or
- it has been assumed initially and not clipped (declipped) in the meantime.

Throughout the desert, the dynamic subset of the axioms of the theory, i.e. $executed(a,t)$ and $observed(c,a,t)$ does not change. Moreover by the definition of desert, all the effects of the actions happened at the initial instant of the desert (t_i) hold throughout the whole desert, (from $t_i + 1$ to t_{i+1}). It follows that the same proof used to prove $holds\_at(fl, t_i + 1)$, whichever of the previous three, also proves $holds\_at(fl,t)$, with t ground such that $t_i + 1 \leq t \leq t_{i+1}$. Hence, if exists a Δ satisfying $P_{TR} \cup\ \Delta \models_{LP} holds\_at(fl,t)$, it must also satisfy $\forall\ i = 0, ..., n\ \forall\ t \in [t_i + 1, t_{i+1}]\ \ P_{TR} \cup\ \Delta \models_{LP} holds\_at(fl, t_i + 1)$. Indeed, note that since the query time point belongs to the time line, it is not influent with respect to constraint satisfaction checking.

□

Proof of Proposition 11.2

(Sketch) Directly from Proposition 11.1: the theory does not change in the desert, hence a set of grounded integrity constraints (with respect to universally quantified variables) is satisfied in every time point of the desert if and only it is satisfied in one the time points, and hence also in the initial one.

□

Proof of Proposition 11.3

$\Rightarrow$ Trivial.

$\Leftarrow$ From Proposition 11.2, follows that $\forall i \; \forall \; t \in [t_i + 1, t_{i+1}] \;\; P_{TR} \cup \Delta \models_{LP} I_{TR}(t)$ and hence $\forall \; t \in [0, maxtime] \; P_{TR} \cup \Delta \models_{LP} I_{TR}(t)$, that implies $P_{TR} \cup \Delta \models_{LP} I_{TR}$, and hence the thesis.

□

Proof of Theorem 11.3 ($\models^{cred}_{TR} \Leftrightarrow \vdash^{cred}_{TR}$)

By Proposition 11.3

$$\langle P_{TR}, A_{TR}, I_{TR} \rangle \models^{cred}_{TR} fl[t] \Leftrightarrow \langle P_{TR}, A_{TR}, I_{TR}(t_0 + 1) \wedge \cdots \wedge I_{TR}(t_n + 1) \rangle \models^{cred}_{TR} fl[t],$$

and by assuming that the proof procedure is correct and complete[39], and by the definition of the predicate *query_credulous*/4,

$$\begin{gathered} \langle P_{TR}, A_{TR}, I_{TR}(t_0) \wedge \cdots \wedge I_{TR}(t_n + 1) \rangle \models^{cred}_{TR} fl[t] \\ \Leftrightarrow \\ query\_credulous\_TR(KB_{TR}, KB_0, fl[t], A) \text{ succeeds} \\ \Leftrightarrow \\ KB_{TR} \vdash^{cred}_{TR} \;\; fl[t], \end{gathered}$$

where $t_0, \ldots \; t_{n+1}$ are the oases of the time line extracted from KB_0.

□

Proof of Theorem 11.4 ($\models^{skep}_{TR} \Leftrightarrow \vdash^{skep}_{TR}$)

Soundness and completeness of $\vdash^{skep}_{TR}$ with respect to $\models^{skep}_{TR}$ straightforwardly follow from the ones of $\vdash^{cred}_{TR}$ with respect to $\models^{cred}_{TR}$ stated in Theorem 11.3.

□

[39] We need to study syntactic restrictions on the theories KB_{TR} in order to guarantee completeness of ALP proof procedures. For theories that do not involve the predicate *contrary*/2, the discrete time on the set of *holds*(F, T) provides a level mapping, and hence finiteness of the theory is sufficient to guarantee termination. For theories with the *contrary*/2 predicate, we suspect that the only loops will be even loops through NAF.

Proof of Proposition 11.4

It trivially holds that

$$\exists\ t \text{ ground } t \in [a,b]\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t]$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \exists\ t \in [t_i+1, t_{i+1}]\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t]$$
$$\Leftrightarrow$$
$$\text{(by applying Proposition 11.2)}$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1].$$

□

Proof of Proposition 11.5

It trivially holds that

$$\forall\ t \text{ ground } t \in [a,b]\ KBtr\ \models_{TR}^{cred}\ fl[t]$$
$$\Leftrightarrow$$
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \exists\ t \in [t_i+1, t_{i+1}]\ KBtr\ \models_{TR}^{cred}\ fl[t]$$
$$\Leftrightarrow$$
$$\text{(by applying Proposition 11.2)}$$
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1].$$

□

Proof of Theorem 11.5 ($\models_{TR}^{cred^{ngq}} \Leftrightarrow \vdash_{TR}^{cred^{ngq}}$)

(Along the line of Theorem 11.3). We show the case for existential queries (the one for universal is analogous). By Proposition 11.4 (respectively Proposition 11.5 for universal queries)

$$KB_{TR} \models_{TR}^{cred^{ngq}} \exists\ T \in [a,b]\ fl[T] \Leftrightarrow \exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1].$$

and by assuming that the proof procedure is correct and complete, and by the definition of the predicate *query_ngq_credulous*/4,

$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1]$$
$$\Leftrightarrow$$
$$query\_ngq\_credulous\_TR(KB_{TR}, KB_0, \exists\ T \in [a,b]\ fl[T], A) \text{ succeeds}$$
$$\Leftrightarrow$$
$$KB_{TR} \vdash_{TR}^{cred^{ngq}}\ \exists\ T \in [a,b]\ fl[T],$$

where $t_0, \ldots\ t_{n+1}$ are the oases of the time line extracted from KB_0.

□

Proof of Proposition 11.6

It holds that

$$\exists\ t \text{ ground } t \in [a,b]\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t]$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \exists\ t \in [t_i+1, t_{i+1}]\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t]$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \exists\ t \in [t_i+1, t_{i+1}]\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t]\ \wedge\ KBtr\ \not\models_{TR}^{cred}\ \overline{fl[t]}$$
$$\Leftrightarrow$$
(by applying Proposition 11.2)
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1]\ \wedge\ KBtr\ \not\models_{TR}^{cred}\ \overline{fl[t_i+1]}$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t_i+1]$$

□

Proof of Proposition 11.7

It holds that

$$\forall\ t \text{ ground } t \in [a,b]\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t]$$
$$\Leftrightarrow$$
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \forall\ t \in [t_i+1, t_{i+1}]\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t]$$
$$\Leftrightarrow$$
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \forall\ t \in [t_i+1, t_{i+1}]\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t]\ \wedge\ KBtr\ \not\models_{TR}^{cred}\ \overline{fl[t]}$$
$$\Leftrightarrow$$
(by applying Proposition 11.2)
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{cred}\ fl[t_i+1]\ \wedge\ KBtr\ \not\models_{TR}^{cred}\ \overline{fl[t_i+1]}$$
$$\Leftrightarrow$$
$$\forall\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t_i+1]$$

□

Proof of Theorem 11.6 ($\models_{TR}^{skep^{ngq}} \Leftrightarrow \vdash_{TR}^{skep^{ngq}}$)

(Along the line of Theorem 11.3). We show the case for existential queries (the one for universal is analogous). By Proposition 11.6 (respectively Proposition 11.7 for universal queries)

$$KB_{TR} \models_{TR}^{skep^{ngq}}\ \exists\ T \in [a,b]\ fl[T]$$
$$\Leftrightarrow$$
$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t_i+1]$$

and by assuming that the proof procedure is correct and complete, and by the definition of the predicate *query_ngq_skeptically*/4,

$$\exists\ i\ [t_i+1, t_{i+1}] \cap [a,b] \neq \emptyset\ \wedge\ KBtr\ \models_{TR}^{skep}\ fl[t_i+1]$$
$$\Leftrightarrow$$
$$query\_ngq\_skeptically\_TR(KB_{TR}, KB_0, \exists\ T \in [a,b]\ fl[T], A) \text{ succeeds}$$
$$\Leftrightarrow$$
$$KB_{TR} \vdash_{TR}^{skep^{ngq}}\ \exists\ T \in [a,b]\ fl[T],$$

where $t_0, \ldots\ t_{n+1}$ are the oases of the time line extracted from KB_0.

□

Proof of Theorem 11.7

(Sketch)

The proof of Theorem 11.7 builds on an appropriately revised versions of the results for $\vdash_{TR}^{skep}$ and $\vdash_{TR}^{skep^{ngq}}$, Theorem 11.4 and Theorem 11.6, respectively (which were soundness and completeness results, under the assumption of a sound and *complete* underlying proof procedure). The proof is based on the following three lemmas.

Lemma C.4. *Let KB_{TR}, KB_0 and $fl[t]$ be as in Definition 11.6 (C-IFF is the proof procedure underlying query_credulously_TR/4), then*

$$query\_credulously\_TR(KB_{TR}, KB_0, fl[t], (\Delta, \Gamma)) \quad \Rightarrow \quad KB_{TR} \models_{TR}^{cred} fl[t] \quad (i)$$

$$query\_credulously\_TR(KB_{TR}, KB_0, fl[t], fail) \quad \Rightarrow \quad KB_{TR} \not\models_{TR}^{cred} fl[t] \quad (ii)$$

(where $\models_{TR}^{cred}$, according to its definition, is based on the semantical entailment of the underling proof procedure, i.e. $\models_{\Re}^{3}$ in this case based on C-IFF).

Proof. The lemma follows by combining the success and failure soundness results for C-IFF (Section 10.2.2) with Theorem 11.3 (soundness —and completeness— results for $\vdash_{TR}^{cred}$), appropriately restated for the case of C-IFF. The sound answer returned by C-IFF, and the sound (and complete) transformation performed by *query_credulously_TR*/4 guarantee the semantical entailment in (i) and (ii). □

Lemma C.5. *Let KB_{TR}, KB_0 and $fl[t]$ be as in Definition 11.6 (C-IFF is the underlying proof procedure for $\vdash_{TR}^{skep}$), then*

$$KB_{TR} \vdash_{TR}^{skep} fl[t] \text{ with Answer} = (\Delta, \Gamma) \quad \Rightarrow \quad KB_{TR} \models_{TR} fl[t] \quad (i)$$

$$KB_{TR} \vdash_{TR}^{skep} \text{(finitely) fails to prove } fl[t] \quad \Rightarrow \quad KB_{TR} \not\models_{TR} fl[t] \quad (ii)$$

Proof. According to the definition of $\vdash_{TR}^{skep}$, for the case of C-IFF (the redefined predicate *query_skeptically_TR*/4)), the left-hand side of (i) implies, by Lemma C.4, $KB_{TR} \models_{TR}^{cred} fl[t] \ \wedge \ KB_{TR} \not\models_{TR}^{cred} \overline{fl[t]}$, that is $KB_{TR} \models_{TR} fl[t]$.

According to the definition of $\vdash_{TR}^{skep}$, for the case of C-IFF (the redefined predicate *query_skeptically_TR*/4)), the left-hand side of (ii) implies, by Lemma C.4, either $KB_{TR} \not\models_{TR}^{cred} fl[t]$ or $KB_{TR} \models_{TR}^{cred} fl[t] \ \wedge \ KB_{TR} \models_{TR}^{cred} \overline{fl[t]}$, that is, in both the cases, $KB_{TR} \not\models_{TR} fl[t]$ □

Lemma C.6. *Let KB_{TR}, KB_0, $fl[T]$ and $TCS(T)$ be as in Definition 11.6 (C-IFF is the underlying proof procedure for $\vdash_{TR}^{skep^{ngq}}$), then*

$$KB_{TR} \vdash_{TR}^{skep^{ngq}} \; exists(TCS(T), fl[t]) \; with \; Answer = (\Delta, \Gamma) \Rightarrow \exists\, \sigma.\; KB_{TR} \models_{TR} fl[T]\sigma \;\wedge\; \sigma \models_{\Re} TCS(T) \qquad (i)$$

$$KB_{TR} \vdash_{TR}^{skep^{ngq}} \; exists(TCS(T), fl[t]) \; with \; Answer = fail \Rightarrow \nexists\, \sigma.\; KB_{TR} \models_{TR} fl[T]\sigma \;\wedge\; \sigma \models_{\Re} TCS(T) \qquad (ii)$$

Proof. (Sketch). The proof of this lemma is analogous to the proof of Lemma C.5, and is based both on the definition of $\vdash_{TR}^{skep^{ngq}}$, for the case of C-IFF (which again is based on the redefined predicate *query_skeptically_TR*/4), and on the soundness results for C-IFF (in particular, as far as they regard existentially quantified queries and temporal constraints).

The left-hand side of (i), the definition of $\vdash_{TR}^{skep^{ngq}}$ and the soundness of C-IFF imply that $\exists\sigma.\; KB_{TR} \vdash_{TR}^{skep}\; fl[T]\sigma$ with $Answer = (\Delta, \Gamma)$, and $\sigma \models_{\Re} TCS(T)$. By Lemma C.5, it holds that $\exists\sigma.\; KB_{TR} \models_{TR} fl[T]\sigma \;\wedge\; \sigma \models_{\Re} TCS(T)$.

The case for (ii) is analogous.

□

Theorem 11.7 follows form Lemma C.5 and Lemma C.6, which comprise all the cases of the definition of $\vdash_{TR}$.

□

Proof of Proposition A.1

$\Rightarrow$ (Sketch).
The placement π can be expressed as $\bigcirc_i \pi_i,\; i \in [0, n]$, i.e. a composition of substitutions, such that $\forall\; i, T\; \pi_i(T) \in [t_i + 1, t_{i+1}]$, i.e. the range of each substitution is contained in a different desert of the extended time line.

Let us show that an essential placement, build upon the original placement, allows to prove the same fluent. If π_i is ϵ then define $\tilde{\pi}_i$ also as ϵ. It π_i is defined over the k variables $\{T_{i_1}, \ldots, T_{i_k}\}$, then $\tilde{\pi}_i$ is defined as

$$\forall\; j \in [1, k] \;\; \tilde{\pi}_i(T_{i_j}) \in [t_j + 1, \ldots, t_j + 1 + o],$$

with $o \leq k \;\wedge\; t_j + o \leq t_{j+1}$, and

$$\forall\; m, n \in [1, k] \quad \begin{array}{l} \pi(T_{i_m}) < \pi(T_{i_n}) \Leftrightarrow \tilde{\pi}_i(T_{i_m}) < \tilde{\pi}_i(T_{i_n}) \wedge \\ \pi(T_{i_m}) = \pi(T_{i_n}) \Leftrightarrow \tilde{\pi}_i(T_{i_m}) = \tilde{\pi}_i(T_{i_n}). \end{array}$$

Basically, the parameter o is determined by how π_i maps variables, since $\tilde{\pi}_i$ is constrained to preserve exactly the same order, while squashing points towards the beginning of the desert.

Observe that $KB_{TR}\tilde{\pi}$, with $\tilde{\pi} = \bigcirc_i \tilde{\pi}_i,\ i \in [0, n]$ is now a ground theory, with action time points and observation time points as oases of the extended time line. The deserts can be populated only by placed actions. Let us now consider the two grounded theories $KB_{TR}\pi, KB_{TR}\tilde{\pi}$ and their relative time lines $[s_0, \ldots, s_m]$ and $[r_0, \ldots, r_m]$, which obviously have the same number of oases. In particular, both the time lines have the time point t of the query $fl[t]$ as, say, j-th oasis.

The following facts hold (along the line of the proof for Proposition 11.1 and Proposition 11.2):

1. $\forall\ i \in [1, m], \exists\ \Delta KB_{TR}\pi \cup \Delta \models_{LP} fl[s_i] \Leftrightarrow KB_{TR}\tilde{\pi} \cup \Delta \models_{LP} fl[r_i]$.

   This can be proved by induction on i by observing that s_i, r_i are the final point of each desert of the two theories, and that all the actions placed by π before the i-th desert of $KB_{TR}\pi$, are placed by $\tilde{\pi}$, in the same order, before the i-th desert of $KB_{TR}\tilde{\pi}$. Assuming that for all the previous deserts the implication holds, at the end of the i-th desert the same fluents must hence hold in both the two cases (given the same set of assumptions Δ). Indeed, they depend on the previously executed actions and observations and their interleaving, that are the same in the two cases.

   Note, hence, that $\exists\ \Delta\ KB_{TR}\pi \cup \Delta \models_{LP} fl[t] \Leftrightarrow KB_{TR}\tilde{\pi} \cup \Delta \models_{LP} fl[t]$.

2. $\forall\ i \in [1, m]$,
   $\exists\, t \in [s_i{+}1, s_{i+1}]\ \exists\Delta\ KB_{TR}\pi{\cup}\Delta \models_{LP} fl[t]\ \Leftrightarrow\ \exists\, t' \in [r_i{+}1, r_{i+1}]\ KB_{TR}\tilde{\pi}{\cup}\Delta \models_{LP} fl[t']$.

   This follows from the standard results for deserts of ground theories, according to which a fluent holds in any point of the desert if it holds at the initial point of the same desert. The fact that at the initial points of each of the i-th deserts of the two theories the same fluents holds is guaranteed by the previous point.

3. $\forall\ t \in [0, \infty]$,
   $\exists\Delta\ KB_{TR}\pi \cup \Delta \not\models_{LP} I_{TR}(t) \Leftrightarrow\ \exists\ t' \in [0, \infty]\ KB_{TR}\tilde{\pi} \cup \Delta \not\models_{LP} I_{TR}(t')$

   Indeed, a point in a desert of $KB_{TR}\pi$ where an integrity constraint is violated, by the previous two points, must have a corresponding point in a desert of $KB_{TR}\tilde{\pi}$ where the same constraint is violated, too. Moreover, as standard, this is the case if and only if the points are both initial points of the relative deserts, justifying again the computational model for ground theories, based on the grounding of integrity constraints over the initial points of each desert.

4. Let Ω be the partial order induced by the temporal constraints of action intervals, then $\Omega \gg \tilde{\pi}$, since $\Omega \gg \pi$ and $\pi \gg \tilde{\pi}$. It follows that $\tilde{\pi}$ satisfies temporal constraints.

From the previous points follows that, given $fl[t]$,

$$\exists\ \pi\ KB_{TRint}\pi \models_{TR}^{cred} fl[t] \Rightarrow \exists\ \tilde{\pi}\ KB_{TRint}\tilde{\pi} \models_{TR}^{cred} fl[t].$$

Finally, it is easy to check that $\tilde{\pi}$ fulfills the definition of essential placement.

$\Leftarrow$ Trivial.

□

Proof of Theorem A.1 ($\models^{cred}_{TRint} \Leftrightarrow \vdash^{cred}_{TRint}$)

(Sketch. Along the line of Theorem 11.3). By Proposition A.1

$$KB_{TRint} \models^{cred}_{TRint} fl[t]$$
$$\Leftrightarrow$$
$$\exists\ \tilde{\pi}\ KB_{TRint}\tilde{\pi} \models^{cred}_{TR} fl[t],$$
$$\Leftrightarrow$$
(by the definition of the predicate $query\_int\_credulous/4$)
$$KB_{TR} \vdash^{cred}_{TRint}\ fl[t].$$

□

Proof of Proposition A.2

$\Rightarrow$ Trivial.

$\Leftarrow$ Analogous to the proof of Proposition A.1.

□

Proof of Theorem A.2 ($\models^{skep}_{TRint} \Leftrightarrow \vdash^{skep}_{TRint}$)

(Along the line of Theorem 11.3).

$$KB_{TRint} \models^{skep}_{TRint} fl[t] \quad \Leftrightarrow \quad \forall\ \pi\ KB_{TRint}\pi \models^{cred}_{TR} fl[t]\ \wedge\ KB_{TRint}\pi \not\models^{cred}_{TR} \neg fl[t]$$

$$\Leftrightarrow$$
(By Proposition A.2)

$$\forall\ \tilde{\pi}\ KB_{TRint}\tilde{\pi} \models^{cred}_{TR} fl[t]\ \wedge\ KB_{TRint}\tilde{\pi} \not\models^{cred}_{TR} \neg fl[t]$$

$$\Leftrightarrow$$
(By the definition of the predicate $query\_int\_skeptically/4$)
$$KB_{TR} \vdash^{skep}_{TRint}\ fl[t].$$

□

C.3 Proof for Reactivity

Proof of Theorem 11.2

The proof follows directly from the correctness of C-IFF, and the following two Lemmas.

Lemma C.7. *Let*

$$\vdash^{\tau}_{react} \{\langle l_1[s_1], S_1\rangle, \ldots, \langle l_k[s_k], S_k\rangle\}, \{\langle a_1[t_1], T_1\rangle, \ldots, \langle a_m[t_m], T_m\rangle\}$$

and let

$$\mathcal{T} = \cup \bigcup_{i=1,\ldots m} T_i \cup \bigcup_{j=1,\ldots k} S_j \cup \bigcup_{i=1,\ldots m} t_i > \tau$$

Then there exists a total valuation σ such that

$$\sigma \models \mathcal{T} \wedge TCS^{nr}.$$

Proof. Similar to the proof of Lemma C.1. □

Lemma C.8. *Let $\mathcal{G}s$ and $\mathcal{A}s$ be sets of pairs of the form $\langle l[t], Tc\rangle$ and $\langle a[t], Tc\rangle$ respectively, and let σ be a total valuation. If*

$$P^{+}_{react} \wedge [\mathcal{AEC}(Plan^{nr}) \wedge \mathcal{AEC}(Goals^{nr}) \wedge \mathcal{AEC}(\mathcal{G}s) \wedge \mathcal{AEC}(\mathcal{A}s)]\sigma \models_{LP} I^{+}_{react}$$

then

$$P_{react} \wedge [\mathcal{EC}(Plan^{nr}) \wedge \mathcal{EC}(Goals^{nr}) \wedge \mathcal{EC}(\mathcal{G}s) \wedge \mathcal{EC}(\mathcal{A}s)]\sigma \models_{LP} I_{react}.$$

Proof. The proof is a direct consequence of the transformation from P_{plan} to P^{+}_{plan} and of arguments similar to the ones adopted in the proof of Lemma C.3. □

C.4 Proofs for Goal decision

Proof of Theorem 11.8

$\mathcal{T} \models^{scept}_{pref} G_i$ iff $\mathcal{T} \models^{cred}_{pref} G_i$ and for each $\overline{G}$ such that $incompatible(\overline{G}, G_i)$, $\mathcal{T} \not\models^{cred}_{pref} \overline{G}$ holds. The first condition follows from the soundness of $\mathcal{T} \vdash^{cred}_{pref} G_i$. To show the second condition assume by contradiction that there exists $\overline{G}$ such that $\mathcal{T} \models^{cred}_{pref} \overline{G}$. Then by the completeness of $\vdash^{cred}_{pref}$ (and goal finiteness assumption) this goal must belong to the set Gs' generated in the first step of the algorithm for $\vdash_{GD}$. But as $\overline{G}$ and G are incompatible then goal G cannot be in the set Gs filtered out in the second step of the algorithm for $\vdash_{GD}$. Contradiction. Maximality of Gs is a straightforward conclusion of the completeness of the set Gs' generated in the first step of the algorithm for $\vdash_{GD}$.

□

D Proofs for Societies

D.1 Lemmas

In this section, we prove some lemmas that will be useful in the following proofs. These Lemmas allow us to establish a corresponding $\mathcal{S}$CIFF computation where all the incoming events are considered at the beginning of the computation, instead of interleaving Happening transitions with the other ones. These results are represented by Lemma D.7 for the open case and Lemma D.8 for the closed case.

A further useful results proved in this section is Lemmas D.4, that will prove that if in a derivation we have a node containing an abduced atom with universally quantified variables, then there will be a universally quantified variable in every non-failure successor node. Thanks to this lemma, we will be able to use results from the IFF proof procedure (in which universally quantified abducibles cannot occur *in any node* of a derivation) in $\mathcal{S}$CIFF derivations that do not *terminate* in a node with universally quantified abducibles.

We first give some intermediate results; first of all, we relate the treatment of disequality in $\mathcal{S}$CIFF and in IFF proof procedures.

Lemma D.1. *The $\mathcal{S}$CIFF proof procedure deals with disequalities in the Constraint Store (Section 19.2.5). The IFF proof procedure transforms a disequality $A \neq B$ into an implication $A = B \rightarrow false$.*

For each of the rules for disequality in $\mathcal{S}$CIFF that does not involve quantifier restrictions, there is one or more rules in IFF that lead to the same node.

Proof. Let us consider a disequality $A \neq B$ in both proof procedures. Let us assume that one of the rules for disequality is used in $\mathcal{S}$CIFF; we prove that there are one or more IFF rules applicable that lead to the same result.

1. Replace $f(t_1, \ldots, t_j) \neq f(s_1, \ldots, s_j)$ with $t_1 \neq s_1 \vee \cdots \vee t_j \neq s_j$.

   In the IFF, we could

   - rewrite $f(t_1, \ldots, t_j) \neq f(s_1, \ldots, s_j)$ as $f(t_1, \ldots, t_j) = f(s_1, \ldots, s_j) \rightarrow false$.
   - Apply the Rules for Equality obtaining $t_1 = s_1 \wedge \cdots \wedge t_j = s_j \rightarrow false$.
   - Apply j times Case Analysis; in the first application we get $t_1 = s_1 \wedge (t_2 = s_2 \wedge \cdots \wedge t_j = s_j \rightarrow false) \vee t_1 \neq s_1$. By iteratively applying Case Analysis to the implications we will get $t_1 \neq s_1 \vee \cdots \vee t_j \neq s_j \vee [t_1 = s_1 \wedge \cdots \wedge t_j = s_j \wedge (true \rightarrow false)]$
   - by applying Logical Equivalences, we have $t_1 \neq s_1 \vee \cdots \vee t_j \neq s_j$.

2. Replace $f(t_1, \ldots, t_j) \neq g(s_1, \ldots, s_l)$ with $true$ whenever f and g are distinct or $j \neq l$.

   In the IFF, we can

   - Rewrite $f(t_1, \ldots, t_j) \neq g(s_1, \ldots, s_l)$ as $f(t_1, \ldots, t_j) = g(s_1, \ldots, s_l) \rightarrow false$.
   - Apply Rewriting Rules for Equality and get $true \rightarrow false$.
   - Apply logical equivalence and get $true$.

The same reasoning can be applied for the other rules 3, 4, 5 and 6a of $\mathcal{S}$CIFF proof procedure (see Section 19.2.5). Rules 6b and 6c involve quantifier restrictions. □

Lemma D.2. *Applying the rules for disequality (Section 19.2.5) cannot change the quantification of a universally quantified variable. Moreover, after applying rules for disequality, the disequality constraints are not imposed on universally quantified variables.*

Proof. Trivial, considering the rules of disequality: either they fail, or they succeed without creating new constraints, or they impose constraints only on existentially quantified variables. □

Lemma D.3. *Let us suppose that the constraint solver only contains the rules for equality and disequality given in Sections 19.2.1 and 19.2.5 (otherwise, the behavior of the proof depends also on the type of constraint solver). Let us suppose that the rules for equality and disequality are applied before the other transitions.*

If an atom A is abduced containing a universally quantified variable $\hat{X}$, and $\hat{X}$ only occurs in abduced atoms, if Propagation is not applied, then the atom A will remain in the set of abduced atoms and variable $\hat{X}$ will remain universally quantified in any success nodes.

Proof. Since variable $\hat{X}$ only occurs in abduced atoms, its state can be changed only by transitions that affect abduced atoms. Let us consider the single transitions:

Unfolding does not affect an abduced atom; it may unify the variables appearing in a goal or in the body of an implication with the head of a clause. However, since variable $\hat{X}$ does not occur in non abducible atoms, it will not be affected.

Abduction does not affect atoms already abduced.

Splitting does not affect abduced atoms.

Case Analysis affects the variables appearing in an implication. Since variable $\hat{X}$ only occurs in abduced atoms, it will not be affected by case analysis.

Factoring is not applicable to universally quantified atoms.

Equivalence Rewriting rules apply only to equalities, thus they will not affect variables that do not occur in an equality.

Logical Equivalence The only rule that can change a (positive) atom is $A \vee true \leftrightarrow true$. This rule can only be applied to a disjunction, but, since atom A has been abduced, it cannot be argument of a disjunction (in fact, disjunctions cannot occur in **EXP**, **FULF**, and **VIOL**, but only in the Constraint Store or in R).

Happening does not change abduced atoms.

non-Happening does not change abduced atoms.

Closure only changes the history and does not change abduced atoms.

Violation NE generates two nodes. One is a violation node. The other imposes a disequality constraint. We know that a disequality constraint cannot bind a universally quantified variable (Lemma D.2). Since we chose a preferred order of application of transitions (namely, we apply the rules for equality and disequality before the other transitions), we can ensure that Violation **NE** will not bind any universally quantified variable in non failure nodes.

Fulfillment E, Violation E deal with **E** atoms, that cannot contain universally quantified variables.

Fulfillment NE does not change atoms, only moves them from **EXP** to **FULF**.

Constraint Solving We do not deal with constraints (except for disequality, for which we know that it does not bind universally quantified variables, and equality, already considered in Equivalence Rewriting rules).

□

Lemma D.4. *If, in a node N, an atom A is abduced containing a universally quantified variable $\hat{X}$, then in any node which is a descendant of N there will be such atom A in the set of abduced atoms with universally quantified variable $\hat{X}$ (unless the node is* false*).*

Proof. We prove the lemma in the following steps:

1. whenever an atom is abduced with a new, universally quantified variable $\hat{X}$, then $\hat{X}$ cannot occur elsewhere except for abducible atoms.

2. Lemma D.3 is applicable, thus the thesis holds if *Propagation* is not applied.

3. Applying *Propagation* creates new universally quantified variables, while the abduced atoms are not touched.

We now elaborate on steps 1 and 3.

1. If the proof procedure abduces an atom with universally quantified variables, then the universally quantified variables can occur only in abducibles.

In fact, the *Abduction* transition can be applied only to atoms in the set R. R may contain a universally quantified atom because it was in the initial goal of the society. In this case, the variable cannot occur in other atoms (except abducibles and constraints, see Section 17.1). We will suppose that the goal does not contain equality constraints; this is not a limitation. A universally quantified atom may be inserted in R by the following transitions:

Unfolding. In this case the universally quantified atom was in the body of a clause; the syntax [94] imposes that the universally quantified variable does not appear elsewhere (except for constraints and other abducibles). Again, we suppose that the body does not contain equality constraints (this is not a limitation).

Logical Equivalence: $(true \rightarrow A) \leftrightarrow A$**.** In this case, the body of an implication has become true. The implication was written with the syntax given in [94]. In particular, since the universally quantified variable $\hat{X}$ is new, then it occurred only in **NE** atoms and constraints. We suppose that the conclusion of the implication does not contain equality constraints (this is not restrictive); for disequality constraints we rely on Lemma D.2.

3. We still have to show that adding the *Propagation* transition does not undermine the thesis. The *Propagation* transition performs a copy of an atom and of an IC, then it operates only on the copy of the two. The universally quantified variables are renamed by the copy, thus any subsequent operation on the copied universally quantified variables will not affect the universally quantified variables occurring in the abduced atom. □

The IFF proof procedure deals with a static theory. $\mathcal{S}$CIFF deals with a dynamic theory to which new happened events may be added during a derivation. So to show a mapping between $\mathcal{S}$CIFF and IFF derivations, we need to address the following question:

> Does the success nodes in $\mathcal{S}$CIFF depend on the events arrival rate? An open successful derivation may disappear, if a new event E happens. Would we have the same success nodes if we had known event E in advance?

Lemmas D.7 and D.8 will try to answer these questions. We first need some intermediate results; we are going to prove that if a transition Tr is applicable to some elements[40] of a node N_k, and leads to a node N_{k+1}, then it can be applied to the same elements of an identical node but with a larger history, and will lead to a node identical to N_{k+1} but with a larger history, as informally suggested by the following scheme:

$$\begin{array}{ccc} N_k & \xrightarrow{Tr} & N_{k+1} \\ & \Downarrow & \\ N_k \cup \{\mathbf{H}(E)\} & \xrightarrow{Tr} & N_{k+1} \cup \{\mathbf{H}(E)\} \end{array}$$

Lemma D.5. *If a transition (except for Happening, Non-happening and Closure) is applicable to some elements of a non-closed node (i.e., a node with an open history)*

$$N_k \equiv \langle R_k, CS_k, PSIC_k, \mathbf{EXP}_k, \mathbf{HAP}_k, \mathbf{FULF}_k, \mathbf{VIOL}_k \rangle$$

and it produces a new node

$$N_{k+1} \equiv \langle R_{k+1}, CS_{k+1}, PSIC_{k+1}, \mathbf{EXP}_{k+1}, \mathbf{HAP}_{k+1}, \mathbf{FULF}_{k+1}, \mathbf{VIOL}_{k+1} \rangle$$

then the same transition is also applicable to the same elements in the (non-closed) node

$$N'_k \equiv \langle R_k, CS_k, PSIC_k, \mathbf{EXP}_k, \mathbf{HAP}_k \cup \{\mathbf{H}(E)\}, \mathbf{FULF}_k, \mathbf{VIOL}_k \rangle$$

where E is an event, and it produces a new node

$$N'_{k+1} \equiv \langle R_{k+1}, CS_{k+1}, PSIC_{k+1}, \mathbf{EXP}_{k+1}, \mathbf{HAP}_{k+1} \cup \{\mathbf{H}(E)\}, \mathbf{FULF}_{k+1}, \mathbf{VIOL}_{k+1} \rangle.$$

Proof. Let us consider the single transitions.

Unfolding is applicable when a literal in R or in the body of an IC matches with the head of one or more rules in the SOKB. It is not affected by the history.

Abduction is applicable when an abducible atom is in R. Its applicability does not directly depend on the history, nor its results.

Propagation is applicable when an atom in the body of an IC matches an atom A (that can either be in the history or in the abduced atoms). If the history is enlarged, the atom A is still its member, thus Propagation can be applied in the same way.

Splitting does not depend on the history.

[40] Recall that transitions are applicable to elements of the nodes; for example, the transition *Violation* **NE** is applied to a happened event and an abduced **NE** atom in a node.

Case Analysis does not depend on the history.

Factoring is not affected.

Equivalence Rewriting rules the history cannot contain equalities.

Logical Equivalence The applicability of these rules depend only on the presence, in the tuple, of implications, conjunctions, and disjunctions; not on the elements in the history.

Violation NE considers an atom $\mathbf{H}(A) \in \mathbf{HAP}_k$ and a $\mathbf{NE}(B) \in \mathbf{EXP}_k$. Thus, if *violation* **NE** is applicable in N_k, then $\exists \mathbf{H}(A) \in \mathbf{HAP}_k$ that has been used in the transition. But $\mathbf{H}(A) \in \mathbf{HAP}_k \cup \{\mathbf{H}(E)\}$. So the same *violation* **NE** transition is applicable in N'_k and the result will be the same.

Fulfillment E is true for the same reasons as transition Violation **NE**.

Violation E is true for the same reasons as transition Violation **NE** if we make the hypothesis of full temporal knowledge (Definition 19.7), and is not applicable otherwise.

Fulfillment NE is not applicable if the history is open.

Consistency does not depend on the history.

Constraint Solving does not depend on the history.

□

In the previous lemma we excluded transitions *Happening, Closure*, and *Non-happening*. We now extend the same result given in the previous lemma to the *Happening* transition.

Lemma D.6. *Consider two nodes N_k and N'_k, which are identical except for the history: $\mathbf{HAP}_k \cup \{\mathbf{H}(E^1)\} = \mathbf{HAP}'_k$. If transition Happening of an event E is applicable to the node N_k, leading to a history $\mathbf{HAP}_{k+1}$, then*

- *either Happening of E is not applicable to N'_k because E was already in its history (i.e., $E = E^1$)*
- *or Happening of E is applicable to N'_k but it fails (and in this case the history $\mathbf{HAP}'_k$ is closed)*
- *or the transition Happening of the event E is applicable to the node N'_k, and in the obtained node $\mathbf{HAP}'_{k+1} = \mathbf{HAP}'_k \cup \{\mathbf{H}(E^1)\}$*

Proof. Trivial, from the definition of transition Happening. □

Now we know that, given a derivation containing a sequence of transitions

$$\ldots \longrightarrow N_k \xrightarrow{Tr} N_{k+1} \xrightarrow{Happening} N_{k+2} \longrightarrow \ldots$$

where Tr is one of the transitions of the proof procedure, except for non-Happening and Closure, we can safely exchange the two transitions:

$$\ldots \longrightarrow N_k \xrightarrow{Happening} N'_{k+1} \xrightarrow{Tr} N_{k+2} \longrightarrow \ldots$$

and obtain an analogous derivation (in fact, from node N_{k+2} the two derivations are the same).

Lemma D.7. *Consider an open successful derivation D*

$$N_0 \to N_1 \to \cdots \to N_{n-1} \to N_n$$

with $\mathbf{HAP}_n$ *the history in the node* N_n *and*

$$N_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \emptyset, \emptyset, \emptyset \rangle.$$

In this case, there exists an open successful derivation D'

$$N'_0 \to N_1 \to \cdots \to N_{m-1} \to N_m$$

with

$$N'_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}_n, \emptyset, \emptyset \rangle$$

in which the final node $N_m \equiv N_n$.

Proof. Let N_h be the first node in D to which transition *happening* was applied. Suppose that *happening* inserts the event $\mathbf{H}(E_\alpha)$ in the history. Since D is an open successful derivation, there is no transition in D of type *closure*; moreover in all the nodes in D the history is open. Thus, we can apply Lemma D.5 to the transition $Tr(N_{h-1})$ (i.e., the transition that was applied to the node N_{h-1}) and get an equivalent derivation D' in which Tr and *happening* are exchanged. Again, we can apply the same method to the node N_{h-2} and so on, until the transition *happening* becomes the first; call D_α the derivation obtained in this way. Of course, D_α terminates in the same node as a derivation D'_α that starts from a history $\mathbf{H}_1 = \{\mathbf{H}(E_\alpha)\}$.

By repeatedly applying the same method for all the *happening* transitions in D, we obtain an equivalent derivation that terminates in the same node. Since no transition was applicable in the final node in D, no transition is applicable in the final node of D'. □

We can prove a similar result for closed successful derivations:

Lemma D.8. *Consider a closed successful derivation D*

$$N_0 \to N_1 \to \cdots \to N_{n-1} \to N_n$$

with $\overline{\mathbf{HAP}_n}$ *the history in the node* N_n *and*

$$N_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \emptyset, \emptyset, \emptyset \rangle.$$

In this case, there exists a closed successful derivation D'

$$N'_0 \to N_1 \to \cdots \to N_{m-1} \to N_m$$

with

$$N'_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \overline{\mathbf{HAP}_n}, \emptyset, \emptyset \rangle$$

in which the final node $N_m \equiv N_n$.

Proof. D is a closed successful derivation, thus there exists exactly one transition of type *closure* in D; call N_c the first node with closed history ($\overline{\mathbf{HAP}_c}$). Let us consider the derivation $D_o \subset D$ that starts from the same initial node N_0 up to the node N_{c-1}.

$$\underbrace{\underbrace{N_0 \longrightarrow N_1 \longrightarrow \ldots \longrightarrow N_{c-1}}_{D_o} \overset{closure}{\longrightarrow} N_c \longrightarrow N_{c+1} \longrightarrow \ldots \longrightarrow N_n}_{D}$$

Derivation D_o does not contain closed nodes, thus we can apply the same proof as in Lemma D.7.

On the rest of the derivation, from N_c to N_n, the history is closed, thus transition *happening* would give a failure. Since D is a successful closed derivation, there is no *happening* transition from N_c to N_n. Thus the lemma holds for the whole derivation D. □

D.2 IFF-like Rewritten Program

The proof of correctness (soundness, in particular) will be given by exploiting soundness results of the IFF proof procedure with respect to three-valued completion semantics. To this end, here we map $\mathcal{S}$CIFF programs into IFF-like (rewritten) programs, and then prove (in Sections D.3 and D.4) that open/closed $\mathcal{S}$CIFF successful derivations in which no literal is abduced with universally quantified variables have a counterpart in IFF derivations.

We first define the rewritten program, which is a translation in IFF syntax of the society's knowledge base. Since we know that no literal is abduced with universally quantified variables, we can replace universally quantified variables with constants.

The allowedness condition for integrity constraints of the IFF proof procedure requires that every variable in the conclusion occurs in the condition. This cannot be the case in our social integrity constraints. However, as discussed in Section 17.2, we can transform our $\mathcal{IC}_S$ into a new set of integrity constraints satisfying the allowedness condition. We give a simple example in the following. An integrity constraint of kind

$$H(p(X)) \rightarrow E(q(Z))$$

is not allowed since a new variable (Z) occurs in the conclusion. But, it can be transformed into the (IFF-like) integrity constraint:

$$H(p(X)) \rightarrow a$$

and the definition:

$$a \leftarrow E(q(Z))$$

which are both allowed.

Definition D.1. *Given an instance of a society knowledge base* $\langle SOKB \cup \mathbf{HAP}, \mathcal{E}, \mathcal{IC}_S \rangle$*, we define the* IFF rewritten program $\langle SOKB^* \cup \mathbf{HAP}, \mathcal{E}, \mathcal{IC}_S^* \rangle$ *as follows:*

- *For each* $IC_S \in \mathcal{IC}_S$ *that does not satisfy the allowedness condition of the IFF proof procedure, we rewrite it as explained earlier.*
- *For each* $IC_S \in \mathcal{IC}_S$ *with a universally quantified variable* X *occurring in the head of a social integrity constraint but not in the body,* X *is replaced in the corresponding* IC_S^* *in* $\mathcal{IC}_S^*$ *with a constant symbol not occurring elsewhere.*
- *In the same way, for each clause in* $SOKB$ *with a variable* X *which is universally quantified in the* Body *of the clause,* X *is replaced in* $SOKB^*$ *with a new constant symbol.*
- *In the same way, for each atom in the goal* G *with a variable* X *which is universally quantified,* X *is replaced in* G^* *with a new constant symbol.*
- *All* $\neg\mathbf{H}$ *atoms are considered as a new predicate without definition (i.e., always false).* $\mathbf{H}$ *events in the history are considered as a predicate in the* $SOKB^*$*.*

- *We complete the $SOKB$ with the Clark's completion to obtain $SOKB^*$.*

Notice that, by construction, given a set of abduced atoms Δ (not containing universally quantified atoms), and an open history for the society, the set of atoms that are true in the rewritten program and in the original society instance are the same.

Lemma D.9. *For every finite ground set $\Delta \subseteq \mathcal{E}$ (non containing universally quantified variables and) non containing the new constant symbols introduced in $SOKB^*$,*

$$SOKB^* \cup \mathbf{HAP} \cup \Delta \models a \Leftrightarrow SOKB \cup \mathbf{HAP} \cup \Delta \models a$$

and

$$SOKB^* \cup \mathbf{HAP} \cup \Delta \models \mathcal{IC}_S^* \Leftrightarrow SOKB \cup \mathbf{HAP} \cup \Delta \models \mathcal{IC}_S$$

where the symbol $\models$ stands for the three-valued completion semantics.

Proof. The syntax imposes that only abducible atoms can be universally quantified in the *Body* of a clause. Thus, the body of a clause

$$a \leftarrow [\forall_X p(X)]$$

(where p is a predicate symbol, in our case, only **NE** and $\neg$**NE**), is true if and only if there exists an atom $\forall_Y p(Y) \in \Delta$, or if for every possible ground atom A with functor p, $A \in \Delta$, which is not, because Δ is finite and ground. Thus, the body of any clause containing universally quantified variables is false in $SOKB$.

The corresponding rewritten clause is

$$a \leftarrow p(c)$$

where c is a new constant symbol. The body of this clause can be true only if Δ contains $p(c)$ or $p(X)$ for some variable X, which is not.

The proof is similar for the universally quantified atoms occurring in the goal, or in the $\mathcal{IC}_S$.

Moreover, atoms $\neg$**H** are all *false* in the rewritten program. In the original society instance, since it is open, atoms $\neg$**H** are new positive literals without definition (see Section 18.2), so they are false as well.

□

D.3 Proof of open soundness

We consider the case of a (possibly non-ground) goal, expectation sets without universally quantified variables, and do not consider CLP constraints in the program.

By the following Lemma D.10, we prove that for this class of programs, any $\mathcal{S}$CIFF open successful derivation has a counterpart in an IFF derivation computed on the IFF-like rewritten program.

Lemma D.10. *Let $\mathcal{S}_{\mathbf{HAP}^i}$ be $\langle SOKB, \mathcal{E}, \mathcal{IC}_S \rangle$. Let (Δ, σ) be the answer extracted from an open successful derivation ($\mathcal{S}_{\mathbf{HAP}^i} \vdash_{\Delta}^{\mathbf{HAP}^f} G$) for an initial goal G and an initial society instance $\mathcal{S}_{\mathbf{HAP}^i}$ evolving to a proper extension (see Definition 18.2) $\mathcal{S}_{\mathbf{HAP}^f}$ such that Δ does not contain universally quantified variables.*

Then (Δ, σ) is an IFF computed answer for G for the program $\langle SOKB^* \cup \mathbf{HAP}^f, \mathcal{E}, \mathcal{IC}_S^* \rangle$.

Proof. We construct a successful IFF derivation from the given successful (open) $\mathcal{S}$CIFF derivation, by mapping every step except non-happening, fulfillment, happening, closure, violation, and propagation onto itself. Propagation is slightly different in the IFF and in the $\mathcal{S}$CIFF proof procedures: in the $\mathcal{S}$CIFF it also performs a copy of the abducible. Let us consider the new transitions, namely non-happening, fulfillment, happening, closure, violation, and propagation.

1. Non-happening transition cannot occur along an open successful derivation (by definition of Non-happening);
2. Violation generates two nodes. The former leading to failure (and therefore not present along a successful open derivation) the latter reproducing the parent node plus a new inequality constraint. Therefore this transition possibly reduces the set of computed substitutions in $\mathcal{S}$CIFF compared to the IFF proof procedure.
3. Happening transition can be removed from the computation thanks to Lemma D.7 by considering the equivalent open successful derivation in $\mathcal{S}$CIFF starting from $\mathbf{HAP}^f$ and leading to the same final node.
4. Closure transition generates two nodes, the former identical to its parent, the latter identical to its parent except for the history which is closed. Therefore the latter node cannot occur along an open successful derivation.
5. Fulfillment. Since the derivation is open, fulfillment can be applied only to positive expectations and generates two nodes where $\mathbf{EXP} \cup \mathbf{FULF}$ is identical to the parent node, plus, respectively, a new equality or inequality constraint. Therefore this transition does not change the set of computed substitutions with respect to the IFF proof procedure.
6. Propagation. The only difference between propagation in $\mathcal{S}$CIFF and in C-IFF is the copy: in the IFF proof procedure *Propagation* is applied to an atom and an implication. In the $\mathcal{S}$CIFF proof procedure, first a copy of the atom is performed. The only difference stands in the case of universally quantified variables in abduced atoms (in fact, copy does not perform anything significant if the atom does not contain universally quantified variables). Since we assume that there are no universally quantified atoms in the final Δ, from Lemma D.4 we know that no literal has been abduced with universally quantified variables in the derivation. Therefore, copy has no effect on the derivation in this case.

□

Soundness in the open case requires to prove that given an open society instance $\mathcal{S}_{\mathbf{HAP}^i}$, if there exists an open successful derivation for G:

$$\mathcal{S}_{\mathbf{HAP}^i} \mid\sim^{\mathbf{HAP}^f}_{\mathbf{EXP}\cup\mathbf{FULF}} G$$

with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$ then

$$\mathcal{S}_{\mathbf{HAP}^f} \mid\approx_{(\mathbf{EXP}\cup\mathbf{FULF})\sigma} G\sigma$$

Let us consider the proof for an atomic goal (the extension to other structures of the formula G is trivial). Let us suppose that:

$$\mathcal{S}_{\mathbf{HAP}^i} \mid\sim^{\mathbf{HAP}^f}_{\mathbf{EXP}\cup\mathbf{FULF}} G$$

with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$, and prove that:

$$\mathcal{S}_{\mathbf{HAP}^f} \mid\!\approx_{(\mathbf{EXP} \cup \mathbf{FULF})\sigma} G\sigma$$

Proving this latter condition corresponds to proving the following ones, separately:

(i) $SOKB \cup \mathbf{HAP}^f \cup \mathbf{FULF} \cup \mathbf{EXP} \models G$;

(ii) $SOKB \cup \mathbf{HAP}^f \cup \mathbf{FULF} \cup \mathbf{EXP} \models \mathcal{IC}_S$;

(iii) $\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{FULF} \cup \mathbf{EXP}$ ($\neg$-consistency for $\mathbf{E}$ atoms);

(iv) $\{\mathbf{NE}(p), \neg\mathbf{NE}(p)\} \not\subseteq \mathbf{FULF} \cup \mathbf{EXP}$ ($\neg$-consistency for $\mathbf{NE}$ atoms);

(v) $\{\mathbf{E}(p), \mathbf{NE}(p)\} \not\subseteq \mathbf{FULF} \cup \mathbf{EXP}$ ($\mathbf{E}$-consistency);

(vi) $\mathbf{HAP}^f \cup \mathbf{FULF} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \nvdash \bot$ (fulfillment).

For the case of an open society, we rely upon the three-valued completion [84] of $SOKB$ and expectation sets (i.e., the set $\mathbf{HAP}$ is not completed, since the society instance is open with respect to the happening of events).

Thanks to Lemma D.10, conditions (i) and (ii) hold on the basis of the soundness results of IFF [47] for the rewritten program; i.e., $SOKB^* \cup \mathbf{FULF} \cup \mathbf{EXP} \models G^*$ and $SOKB^* \cup \mathbf{FULF} \cup \mathbf{EXP} \models \mathcal{IC}_S^*$. Since the declarative reading of the rewritten program is the same, in this case, as the society instance (Lemma D.9), conditions (i) and (ii) hold.

Notice that soundness of IFF is given with respect to a (three-valued) completion semantics of the theory adopted by the proof procedure. This is not the case in our $\mathcal{S}$CIFF open derivation, since history $\mathbf{HAP}$ has not been completed. But since negative literals of kind $\neg\mathbf{H}()$ are viewed as new positive predicates, they are never propagated as in the IFF corresponding derivation.

Let us consider the other conditions.

Conditions (iii), (iv) and (v) hold thanks to the enforcing of E-consistency and $\neg$-consistency, that generate at most two nodes. In particular, conditions (iii) and (iv) are necessary because we deal with negation of abducible atoms differently from the IFF proof procedure: recall that $\neg\mathbf{E}$ and $\neg\mathbf{NE}$ are considered as new positive atoms.

By contradiction, let us assume that $\mathbf{E}(p)$ and $\neg\mathbf{E}(p)$ belong to Δ (i.e., Δ is not $\neg$-consistent). In this case however, the requirement of $\neg$-consistency would lead to failure (and therefore that $\mathbf{E}(p)$ and $\neg\mathbf{E}(p)$ would not be present at the same time into a node along a successful open derivation). Analogously for $\mathbf{NE}(p)$ and $\neg\mathbf{NE}(p)$, and $\mathbf{E}(p)$ and $\mathbf{NE}(p)$ ($\mathbf{E}$-consistency).

Condition (vi) (fulfillment) holds thank to transitions Fulfillment and Violation. By contradiction, let us assume that condition (vi) does not hold. In a three-valued setting this can happen only if there exists an expectation $\mathbf{NE}(p)$ and the corresponding event $\mathbf{H}(p)$. In this case however, transition violation $\mathbf{NE}$ would apply leading to a node not along a successful open derivation.

D.4 Proof of closed soundness

As in the open case, we consider the case of a (possibly non-ground) goal, expectation sets without universally quantified variables, and do not consider CLP constraints in the program.

By the following Lemma D.11, we prove that for this class of programs, any $\mathcal{S}$CIFF closed successful derivation has a counterpart in an IFF derivation computed on the IFF-like rewritten program.

Lemma D.11. *Let $\mathcal{S}_{\mathbf{HAP}^i}$ be $\langle SOKB, \mathcal{E}, \mathcal{IC}_S \rangle$ where $\mathcal{IC}_S$ does not contain $\neg\mathbf{H}$ literals. Let (Δ, σ) be the answer extracted from a closed successful derivation ($\mathcal{S}_{\mathbf{HAP}^i} \vdash_{\Delta}^{\overline{\mathbf{HAP}^f}} G$) for an initial goal G and an initial society instance $\mathcal{S}_{\mathbf{HAP}^i}$ evolving to a proper extension $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$ such that Δ does not contain universally quantified variables.*

Then (Δ, σ) is an IFF computed answer for G for the program $\langle SOKB^* \cup \overline{\mathbf{HAP}^f}, \mathcal{E}, \mathcal{IC}_S^* \rangle$.

Proof. We construct a successful IFF derivation from the given successful (closed) $\mathcal{S}$CIFF derivation, by mapping every step except non-happening, fulfillment, happening, closure, violation, and propagation onto itself. Propagation is slightly different in the IFF and in the $\mathcal{S}$CIFF proof procedures: in the $\mathcal{S}$CIFF it also performs a copy of the abducible.

Let us consider the new transitions, namely violation, happening, closure, fulfillment, and propagation.

1. Violation generates two nodes. The former leading to failure (and therefore not present along a successful open derivation) the latter reproducing the parent node plus a new inequality constraint. Therefore this transition possibly reduces the set of computed substitutions in $\mathcal{S}$CIFF compared to the IFF proof procedure.

2. Happening and Closure transitions can be removed from the computation thanks to Lemma D.8 by considering the equivalent open successful derivation in $\mathcal{S}$CIFF starting from $\overline{\mathbf{HAP}^f}$ and leading to the same final node.

3. Fulfillment. In the closed case, fulfillment can be applied both to positive and negative expectations. It generates two nodes where $\mathbf{EXP} \cup \mathbf{FULF}$ is identical to their parent node, plus, respectively, a new equality or inequality constraint. Therefore this transition does not change the set of computed substitutions with respect to the IFF proof procedure.

4. Propagation. Same discussion as for the open case (Lemma D.10).

□

For the case of a closed society instance, we rely upon the 3-valued completion [84] of $SOKB$, expectation sets and the set $\mathbf{HAP}$ too, since the society instance is now closed with respect to the happening of events.

We have to prove that given a closed society instance $\mathcal{S}_{\overline{\mathbf{HAP}^f}}$, if there exists a closed successful derivation:

$$\mathcal{S}_{\mathbf{HAP}^i} \vdash_{\mathbf{EXP} \cup \mathbf{FULF}}^{\overline{\mathbf{HAP}^f}} G$$

with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$ then

$$\mathcal{S}_{\overline{\mathbf{HAP}^f}} \models_{(\mathbf{EXP} \cup \mathbf{FULF})\sigma} G\sigma$$

Let us consider the proof for an atomic goal (the extension to other structures of the formula G is trivial). Let us suppose that:

$$\mathcal{S}_{\mathbf{HAP}^i} \vdash_{\mathbf{EXP} \cup \mathbf{FULF}}^{\mathbf{HAP}^f} G$$

with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$, and prove that:

$$\mathcal{S}_{\overline{\mathbf{HAP}^f}} \models_{(\mathbf{EXP} \cup \mathbf{FULF})\sigma} G\sigma$$

Proving this latter condition correspond to prove the following ones, separately:

(*i*) $SOKB \cup \mathbf{HAP}^f \cup \mathbf{FULF} \cup \mathbf{EXP} \models G$;

(*ii*) $SOKB \cup \mathbf{HAP}^f \cup \mathbf{FULF} \cup \mathbf{EXP} \models \mathcal{IC}_S$;

(*iii*) $\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{FULF} \cup \mathbf{EXP}$ ($\neg$-consistency for **E** atoms);

(*iv*) $\{\mathbf{NE}(p), \neg\mathbf{NE}(p)\} \not\subseteq \mathbf{FULF} \cup \mathbf{EXP}$ ($\neg$-consistency for **NE** atoms);

(*v*) $\{\mathbf{E}(p), \mathbf{NE}(p)\} \not\subseteq \mathbf{FULF} \cup \mathbf{EXP}$ (**E**-consistency);

(*vi*) $\mathbf{HAP}^f \cup \mathbf{FULF} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \not\vdash \bot$ (fulfillment).

Thanks to Lemma D.11, conditions (*i*) and (*ii*) hold on the basis of the soundness results of IFF [47], in particular condition (*ii*) holds when no $\neg\mathbf{H}()$ literal occurs in the body of social integrity constraints in $\mathcal{IC}_S$. We have then proved that condition (*ii*) above holds even when literals of kind $\neg\mathbf{H}()$ occur in the body of social integrity constraints. The IFF proof procedure handles negation in the body of integrity constraints in a different manner: in particular, negated literals are turned into positive ones, and moved to the head of the constraint as additional disjunct. We apply, instead, constructive negation [112] to $\neg H()$ literals (see transition *Non-happening*), and therefore benefit of the soundness results of this procedure.

Conditions (*iii*), (*iv*), and (*v*) hold for the same reasons explained in the open case (Section D.3).

Condition (*vi*) (fulfillment) holds thank to transitions Fulfillment and Violation. By contradiction, let us suppose that condition (*vi*) does not hold. This can happen, as in the open case, if there exists an expectation $\mathbf{NE}(p)$ and the corresponding event $\mathbf{H}(p)$. In this case however, transition *violation* **NE** would apply leading to a node not along a successful closed derivation.

In the closed case, condition (*vi*) could fail to hold because there is an $\mathbf{E}(p)$ atom without a matching $\mathbf{H}(p)$. In his case, however, transition *Violation* **E** would apply, again leading to a node that cannot stand along a successful derivation.

D.5 Soundness with universally quantified abducibles

Soundness with universally quantified abducibles is work in progress. We provide a lemma (Lemma D.12) that will be used as a backbone for the full proof of soundness.

As will be clear soon, in the proof of Lemma D.12, we use a slightly different rule for *Unfolding* than the one used in the IFF and in the $\mathcal{S}$CIFF proof procedures. Let us call this transition *Unfolding**. Recall (Section 19.2.1) that Unfolding is applicable to

1. an atom in a conjunct and a clause
2. an atom in an implication and a set of clauses

Transition *Unfolding** coincides with Unfolding in the first case and is defined as follows in the second:

Definition D.2. *If*

$$PSIC_k = \{Atom, BodyIC \rightarrow HeadIC\} \cup PSIC',$$

and if the clauses $H_1 \leftarrow B_1$, ..., $H_n \leftarrow B_n$ *belong to the SOKB, and* $H_1, \ldots, H_n$ *unify with Atom,* Unfolding* *selects a clause* $H_i \leftarrow B_i$ $(1 \leq i \leq n)$ *and produces the following node:*

$$\begin{aligned} PSIC_{k+1} = \quad & \{Atom, BodyIC \rightarrow HeadIC, \\ & Atom' = H_i, B_i, BodyIC' \rightarrow HeadIC'\} \cup PSIC' \end{aligned}$$

where $Atom', BodyIC' \rightarrow HeadIC'$ *is a copy of* $Atom, BodyIC \rightarrow HeadIC$.

The proof of soundness of the IFF proof procedure is based upon the following Proposition, called Proposition 4.1 in [46]:

Proposition D.1. (Fung) *Given a node* N *and a set of computable immediate successors* S *of* N*, we have:*

$$Comp(T, P - Ab) \cup IC \models N \leftrightarrow \textit{the disjunction of the nodes in } S$$

where P *is the set of predicate symbols in the language of the program.*

The proofs that follow Proposition 4.1 in [46], up to the proof of soundness of the whole proof procedure, do not consider the various transitions anymore, but only rely on Proposition 4.1. Thus, by extending Proposition D.1 also for *Unfolding**, we prove that the IFF proof procedure is sound also if enlarged with the further transition *Unfolding**.

Proposition D.2. *Given a node* N *and a set of computable immediate successors* S *of* N *computed by transition Unfolding***, we have:*

$$Comp(T, P - Ab) \cup IC \models N \leftrightarrow \textit{the disjunction of the nodes in } S$$

where P *is the set of predicate symbols in the language of the program.*

Proof. Let us consider a node N with an implication $Atom, BodyIC \rightarrow HeadIC$ and a predicate H defined by the clauses $H_1 \leftarrow B_1, \ldots, H_n \leftarrow B_n$.

Transition *Unfolding** produces the node

$$Atom, BodyIC \rightarrow HeadIC \bigwedge Atom' = H_i, B_i, BodyIC' \rightarrow HeadIC'$$

that is obviously logically equivalent to node N. □

Thus, the IFF proof procedure extended with transition *Unfolding** is sound.

Again, since the proof of soundness of the $\mathcal{S}$CIFF proof procedure without universally quantified abducibles, call it $\mathcal{S}\text{CIFF}_{\setminus\forall}$ (Sections D.3 and D.4) was based on soundness of the IFF proof procedure, also $\mathcal{S}$CIFF with *Unfolding** is sound. We can now base the proof of soundness of $\mathcal{S}$CIFF (*with* universally quantified abducibles) on the soundness of the $\mathcal{S}\text{CIFF}_{\setminus\forall}$ enlarged with transition *Unfolding**.

Lemma D.12. *Consider an (open/closed) successful derivation* D

$$N_0 \rightarrow N_1 \rightarrow \cdots \rightarrow N_{n-1} \rightarrow N_n$$

with

$$N_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}, \emptyset, \emptyset \rangle,$$

$$N_n \equiv \langle \emptyset, CS_n, PSIC_n, \mathbf{EXP}_n, \mathbf{HAP}, \mathbf{FULF}_n, \emptyset \rangle$$

in which transition Happening is not applied. Let $\mathbf{EXP}_n^{\forall} \subseteq \mathbf{EXP}_n$ *and* $\mathbf{FULF}_n^{\forall} \subseteq \mathbf{FULF}_n$ *be the sets of (pending and fulfilled) abduced expectations of type* $[\neg]\mathbf{NE}$*. Let* σ *be the substitution applied to* $\mathbf{EXP}_n \cup \mathbf{FULF}_n$ *by answer extraction on node* N_n*; by definition of answer extraction, variables in* $[\mathbf{EXP}_n \cup \mathbf{FULF}_n]\sigma$ *are universally quantified.*

Consider a society with $SOKB' = SOKB \cup [\mathbf{EXP}_n^\forall \cup \mathbf{FULF}_n^\forall]\sigma$ *(meaning that for each atom* $A \in [\mathbf{EXP}_n^\forall \cup \mathbf{FULF}_n^\forall]\sigma$ *there is a clause* $A \leftarrow true$ *in* $SOKB'$*) in which the literals* **NE** *and* $\neg$**NE** *are considered as defined predicates*[41]*. In this case, there exists an (open/closed) successful derivation* D' *starting from the initial node*

$$N'_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}, \emptyset, \emptyset \rangle$$

terminating in a node N'_m *such that*

$$\begin{array}{rcll} R'_m\sigma & = & R_n\sigma & = \emptyset \\ CS'_m & \subseteq & CS_i & \\ PSIC'_m\sigma & = & PSIC_n\sigma & \\ \mathbf{EXP}'_m\sigma & = & \left[\mathbf{EXP}_n \setminus \mathbf{EXP}_n^\forall\right]\sigma & \\ \mathbf{FULF}'_m\sigma & = & \left[\mathbf{FULF}_n \setminus \mathbf{FULF}_n^\forall\right]\sigma & \\ \mathbf{VIOL}'_m\sigma & = & \mathbf{VIOL}_n\sigma & = \emptyset \end{array}$$

and the assignment σ *trivially satisfies the constraints* $CS_n \setminus CS'_m$*.*

Proof. Let θ be the substitution that binds each existentially quantified variable in each node of the derivation D to its final value in N_n/σ (thus, $\sigma \subseteq \theta$). Let $\Delta_n^\forall = [\mathbf{EXP}_n^\forall \cup \mathbf{FULF}_n^\forall]$.

We build the derivation D' from the derivation D; we show that for each node $N_i \in D$ there is a node $N'_j \in D'$ such that $N_i/\theta = N'_j/\theta$, except for the sets of abduced, for which

$$[\mathbf{EXP}'_j]\theta = [\mathbf{EXP}_i \setminus \mathbf{EXP}_n^\forall]\theta$$

$$[\mathbf{FULF}'_j]\theta = [\mathbf{FULF}_i \setminus \mathbf{FULF}_n^\forall]\theta$$

By induction, we will assume that the thesis holds up to node N_i in D, and that there exists a corresponding node N'_j in D', and we prove that the thesis holds for $N_{i+1} \in D$, $N'_{j+d_j} \in D'$ for some $d_j \geq 0$.

We show that, given the transition Tr from N_i to N_{i+1}, there is one (or more) transition Tr' in D' applicable to the node N'_j leading to a node N'_{j+d_j} for which the thesis holds.

Transition Tr can be one of the following:

Unfolding. In this case, Tr' is also unfolding, applied to the same atom and clause. Since the thesis holds for N_i and N'_j, it holds also for N_{i+1} and N'_{j+1}.

Abduction. If the literal selected for abduction, L, is of type $[\neg]$**NE**, then Tr' is Unfolding applied to the same literal L and to the clause C defined as follows.

- If L contains universally quantified variables, we know from Lemma D.4 that L will be in all the descendant nodes, so it will be in $\Delta_n^\forall$, and there will be a corresponding clause C in $SOKB'$. By definition of $SOKB'$ and θ, C is $A \leftarrow true$, where $A = L/\theta$. The selected clause is C.

  In fact, Abduction gives a node N_{i+1} such that

$$N_{i+1} \equiv \langle R_i \setminus \{L\}, CS_i, PSIC_i, \mathbf{EXP}_i \cup \{L\}, \mathbf{HAP}, \mathbf{FULF}_i, \emptyset \rangle$$

[41] Recall that literals $\neg$**NE** are mapped to new positive literals.

Unfolding gives a node N'_{j+1} such that

$$N'_{j+1} \equiv \langle R'_j \wedge \{true\} \setminus \{L\}, CS'_j \cup \{L = A\}, PSIC'_j, \mathbf{EXP}'_j, \mathbf{HAP}, \mathbf{FULF}'_j, \emptyset \rangle$$

We can apply the logical equivalence $Q \wedge true \leftrightarrow Q$ to R'_{j+1}. We can then apply constraint solving steps to deal with the constraint $L = A$; since $A = L/\theta$, the constraint rewrites to *true* and gives the substitution $\eta = \theta|_{vars(L)}$ to the (existentially quantified) variables in L. We reach the node:

$$N'_{j+2} \equiv \langle R'_j \setminus \{L\}, CS'_j, PSIC'_j, \mathbf{EXP}'_j, \mathbf{HAP}, \mathbf{FULF}'_j, \emptyset \rangle \eta$$

for which the thesis trivially holds (given that it holds for N'_j).

- If L does not contain universally quantified variables, $L/\theta \in N_n/\sigma$, so it will be in $\Delta_n^{\forall}$, and there will be a corresponding clause C in $SOKB'$. The selected clause is C, and the proof follows the scheme in the previous bullet.

If the selected literal L is not of type $[\neg]\mathbf{NE}$, then Tr' is abduction.

Propagation. Propagation is applied to a literal L and an implication. If L is not of type $[\neg]\mathbf{NE}$, transition Tr' is Propagation applied to the same elements.

Otherwise, Propagation is applied in the node

$$N_i \equiv \langle R_i, CS_i, PSIC_i, \mathbf{EXP}_i, \mathbf{HAP}, \mathbf{FULF}_i, \emptyset \rangle$$

to an implication $A, L_1, \ldots, L_n \rightarrow Q \in PSIC_i$ and a literal L of type $[\neg]\mathbf{NE} \in \mathbf{EXP}_i \cup \mathbf{FULF}_i$. Let us suppose that $L \in \mathbf{EXP}_i$, being the proof for the case $L \in \mathbf{FULF}_i$ very similar. The resulting node is

$$N_{i+1} \equiv \langle R_i, CS_i, PSIC_i \cup \{A = L, L_1, \ldots, L_n \rightarrow Q\}, \mathbf{EXP}_i, \mathbf{HAP}, \mathbf{FULF}_i, \emptyset \rangle.$$

Transition Tr' is *Unfolding**, applied to the literal A occurring in the body of an implication, and the clause C selected as follows. Since $L \in \mathbf{EXP}_i$ in derivation D, then $L/\theta \in N_n/\sigma$, and there will be a corresponding clause C in $SOKB'$. By definition of $SOKB'$ and θ, C is $H \leftarrow true$, where $H = L/\theta$. The selected clause is C. *Unfolding** generates a node

$$N'_{j+1} \equiv \langle R'_j, CS'_j, PSIC'_j \cup \{A = L, true, L_1, \ldots, L_n \rightarrow Q\}, \mathbf{EXP}'_j, \mathbf{HAP}, \mathbf{FULF}'_j, \emptyset \rangle$$

to which the logical equivalence $F \wedge true \leftrightarrow F$ is applicable, leading to node N'_{j+2} for which the thesis holds.

Splitting. Tr' is splitting, applied to the same disjunction.

Case Analysis. Tr' is case analysis, applied to the same elements.

Factoring. If applied to atoms different from $[\neg]\mathbf{NE}$, Tr' is factoring applied to the same abducibles. Otherwise, factoring generates two nodes; only one of the two children will be in the derivation D.

In the first, factoring unifies two abducibles L_1 and L_2. If this first node is in D, by definition of θ we know that $L_1/\theta = L_2/\theta$. Thus, in $\Delta_n^{\forall}$ they correspond to the same

clause C, so the thesis already holds for the nodes N_{i+1} and N'_j (i.e., we do not introduce a transition in the derivation D' at this step).

In the second node, factoring imposes that $L_1 \neq L_2$. Thus, in $\Delta_n^{\forall}$ they correspond to different clauses C, so, again, the thesis already holds for the nodes N_{i+1} and N'_j and we do not introduce a transition in the derivation D'.

Equivalence Rewriting. Tr' is the same equivalence rewriting rule, applied to the same elements.

Happening. Is not considered.

non-Happening. Tr' is non-Happening, applied to the same elements.

Closure. Tr' is closure.

Violation NE. Applies to a node N_i in which $\mathbf{NE}(X) \in \mathbf{EXP}_i$ and $\mathbf{H}(Y) \in \mathbf{HAP}_i$ such that X and Y are unifiable. Violation **NE** generates two nodes: in the first, X is unified with Y and a violation is raised (thus, this node cannot be in a successful derivation). In the second, $X \neq Y$ is imposed. Thus, X and Y will be bound to non unifiable terms by the substitution θ. transition; the thesis already holds for the pair of nodes N_{i+1} and N_j, as well as for N_i and N_j. Intuitively, Fulfillment **NE** simply moves an expectation from **EXP** to **FULF**. In the current lemma, this distinction is blurred, as all expectations of the type $[\neg]$**NE** are in the $SOKB'$, wether they were fulfilled or pending.

Fulfillment E, Violation E. Tr' is Fulfillment **E** (resp. Violation **E**), applied to the same elements.

We still have to show that D' is a successful derivation, i.e.,

- its final node N'_m satisfies conditions of IC_S-, $\neg$-, and **E**-consistency, plus fulfillment;
- N'_m is a node of quiescence.

N'_m satisfies conditions of IC_S-, $\neg$-, and **E**-consistency, plus fulfillment because the corresponding node N_n does; in fact $[\mathbf{EXP}'_m \cup \mathbf{EXP}_n^{\forall}]\sigma = [\mathbf{EXP}_n]\sigma$ and $[\mathbf{FULF}'_m \cup \mathbf{FULF}_n^{\forall}]\sigma = [\mathbf{FULF}_n]\sigma$.

Concerning quiescence, we know that N_n is a node of quiescence for the society with $SOKB$. By construction, we know that $[\mathbf{EXP}'_m]\sigma \subseteq [\mathbf{EXP}_n]\sigma$ (and $[\mathbf{FULF}'_m]\sigma \subseteq [\mathbf{FULF}_n]\sigma$), while the other elements of the tuple are the same. Thus, if a transition that only involves the elements in the tuple is applicable to N'_m, then it is also applicable in N_n. The only difference is the $SOKB$; the only transition applied to clauses in $SOKB$ is *Unfolding*.

Unfolding can be applied to an atom in a conjunct of R, or in the body of an implication. But, since N_n is a final node, $R_n = \emptyset$, thus $R'_m = \emptyset$, so the first case cannot be. In the second case, an atom of a predicate defined in the $SOKB$ occurs in the body of an implication. Since the only difference stands in $[\neg]$**NE** atoms, and R_n is a node of quiescence, the only possibility is that a literal L of type $[\neg]$**NE** occurs in the body of a PSIC. But, in order to apply Unfolding, there must be a clause C matching with L. In this case, by definition of $SOKB'$, we would have a corresponding atom $A \in \mathbf{EXP}_n \cup \mathbf{FULF}_n$ matching L, and this would make transition Propagation applicable to node N_n, which is not. Thus, also the second case is impossible.

□