Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture[*]

Alessandro Armando
Mechanized Reasoning Group
DIST, University of Genoa, Italy
armando@dist.unige.it

Daniele Zini
Mechanized Reasoning Group
DIST, University of Genoa, Italy
danielez@dist.unige.it

Abstract

There is a growing interest in the integration of mechanized reasoning systems such as automated theorem provers, computer algebra systems, and model checkers. State-of-the-art reasoning systems are the result of many man-years of careful development and engineering, and usually they provide a high degree of sophistication in their respective domain. Yet they often perform poorly when applied outside the domain they have been designed for. The problem of integrating mechanized reasoning systems is therefore being perceived as an important issue in automated reasoning. In this paper we present the Logic Broker Architecture, a framework which provides the needed infrastructure for making mechanized reasoning systems interoperate. The architecture provides location transparency, a way to forward requests for logical services to appropriate reasoning systems via a simple registration/subscription mechanism, and a translation mechanism which ensures the transparent and provably sound exchange of logical services.

1 Introduction

There is a growing interest in the integration of mechanized reasoning systems such as automated theorem provers, computer algebra systems, and model checkers. State-of-the-art reasoning systems are the result of many man-years of careful development and engineering, and usually they provide a high degree of sophistication in their respective domain. Yet they often perform poorly when applied outside the domain they have been designed for. For instance, computer algebra systems are usually good in performing heavy computation whereas theorem provers are not suited for such tasks; on the other hand theorem provers usually provide the user with more expressive languages than computer algebra systems' and this is a crucial feature in many practical applications.

The problem of integrating mechanized reasoning systems is therefore being perceived as an important issue in automated reasoning. Unfortunately this is not an easy task. The main difficulty is that most of the existing reasoning systems are conceived and built as stand-alone systems to be used by human users. Moreover, if the logical services provided by the component reasoning systems are not interfaced in a proper way, then the logical services provided by the compound systems may be unsound. This makes the composite systems unusable in all the application domains where the correctness is of paramount importance as, e.g., in the formal verification of safety or security critical systems.

In the last few years a number of prototype integrations have been proposed [2, 4, 5, 6, 7, 11, 16, 17, 18]. However all the existing attempts are ad-hoc solutions, geared to the characteristics of the specific systems considered. Moreover, in most cases solutions and techniques available in mature or emerging areas such as Software Engineering or Artificial Intelligence (as, for instance, CORBA or the Agent-Oriented Programming paradigm [13]) have been neglected.

In this paper we present the *Logic Broker Architecture*, a framework which provides the needed infrastructure for making mechanized reasoning systems interoperate. The architecture provides location transparency, a way to forward requests for logical services to appropriate reasoning systems via a simple registration/subscription mechanism, and a translation mechanism which ensures the transparent and provably sound exchange of logical services.

The paper is organized in the following way. In the next section we discuss the problem of interfacing reasoning systems in a sound way and outline a general solution to the problem via a worked out example. In Section 3 we introduce the Logic Broker Architecture and illustrate its basic functionalities. In Section 4 we present our running pro-

[*] We are grateful to Silvio Ranise for useful feedback on an earlier draft of this paper.

totype of the Logic Broker Architecture based on CORBA and the OPENMATH standard [9]. In Section 5 we discuss the related work. Finally in Section 6 we draw some concluding remarks.

2 Interfacing Mechanized Reasoning Systems

By *mechanized reasoning systems* (reasoning systems for short) we mean software systems capable of providing a set of deductive reasoning capabilities such as the ability of simplifying expressions, (dis)proving formulae, and solving sets of constraints. All such activities—which we call *logical services*—are relative to a given logic and the critical part of interfacing logical services is to ensure the 'compatibility' of the associated logics. Indeed if a reasoning system C (short for *client*) uses logical services of a reasoning system S (short for *server*) and if the logic of S is not 'compatible' with that of C, then C may end up with claiming the validity of invalid facts thereby compromising the soundness of the logical services it provides.

We model the logic mechanized by a reasoning system by means of a *consequence relation* [3], i.e. a pair of the form $(L, \vdash)$ where L is a set of sentences and $\vdash \subseteq \mathcal{P}(L) \times L$[1] is a binary relation enjoying the following properties:

1. (*Inclusion*) if $\alpha \in \Gamma$, then $\Gamma \vdash \alpha$;
2. (*Monotonicity*) if $\Gamma \vdash \alpha$, then $\Gamma \cup \Delta \vdash \alpha$;
3. (*Cut*) if $\Gamma \vdash \alpha$ and $\Delta \cup \{\alpha\} \vdash \beta$, then $\Gamma \cup \Delta \vdash \beta$;

for all $\Gamma, \Delta \in \mathcal{P}(L)$ and $\alpha, \beta \in L$. Most of the logical services provided by reasoning systems can be specified in terms of the associated consequence relation, say $(L, \vdash)$. For instance, the logical service $\texttt{prove}(\Gamma, \alpha)$ which establishes whether a set of formulae Γ entails a formula α can be specified to return a positive answer if and only if $\Gamma \vdash \alpha$ holds. Similarly, the fact that the activity of simplifying α w.r.t. the information stored in Γ returns a formula β, in symbols $\texttt{simplify}(\Gamma, \alpha) = \beta$, can be specified by $\Gamma \vdash (\alpha \leftrightarrow \beta)$ where $\leftrightarrow$ is the logical connective for equivalence.

Given two consequence relations $(L_1, \vdash_1)$ and $(L_2, \vdash_2)$, a *morphism* from $(L_1, \vdash_1)$ into $(L_2, \vdash_2)$ is a function $\phi : L_1 \to L_2$ such that $\Gamma \vdash_1 \alpha$ implies $\phi(\Gamma) \vdash_2 \phi(\alpha)$ for all $\Gamma \in \mathcal{P}(L)$ and $\alpha \in L$.[2]

Let $(L_c, \vdash_c)$ and $(L_s, \vdash_s)$ be the consequence relations modeling the logics of two reasoning systems (the *client* and the *server*, respectively). It readily follows from the definition that if there exists a morphism ϕ from $(L_s, \vdash_s)$ into $(L_c, \vdash_c)$ then a problem of the form $\Gamma_c \vdash_c \alpha_c$ for some

[1]$\mathcal{P}(L)$ denotes the power set of L.
[2]$\phi(\Gamma)$ abbreviates $\{\phi(\gamma) : \gamma \in \Gamma\}$.

(Reflexivity)	$x' \preceq x'$
(Antisymmetry)	$(x' \preceq y' \wedge y' \preceq x') \Rightarrow x' = y'$
(Transitivity)	$(x' \preceq y' \wedge y' \preceq z') \Rightarrow x' \preceq z'$

Figure 1. An axiomatization for partial orderings

$\Gamma_c \in \mathcal{P}(L_c)$ and $\alpha_c \in L_c$ arising at the *client* side can always be reduced to asking the server whether $\Gamma_s \vdash_s \alpha_s$ for any $\Gamma_s \in \mathcal{P}(L_s)$ and $\alpha_s \in L_s$ such that $\phi(\Gamma_s) = \Gamma_c$ and $\phi(\alpha_s) = \alpha_c$. The notion of morphism between consequence relations therefore gives us a rigorous account of the notion of compatibility between logics we mentioned above.

Example. To illustrate, let us consider a *client* reasoning system with consequence relation $(L_c, \vdash_c)$ where L_c is a quantifier free first order language built out of an enumerable set of individual constants $a, b, c, \ldots$, an enumerable set of variables $x, y, z, \ldots$, the binary function symbols $\cap$ and $\cup$ denoting set union and intersection respectively, and the binary predicate symbol $\subseteq$ denoting set inclusion; $\vdash_c$ is such that $\Gamma \vdash_c \alpha$ if and only if $T_c \cup \Gamma \models \alpha$ where $\models$ denotes entailment in classical first order logic, and T_c is any suitable axiomatization for set theory (see, e.g., in [20]).

Let us consider also a *server* reasoning system with consequence relation $(L_s, \vdash_s)$ where L_s is a quantifier free first order language built out of an enumerable set of individual constants $a', b', c', \ldots$, an enumerable set of variables $x', y', z', \ldots$, and the binary predicate symbol $\preceq$ denoting a partial ordering; $\vdash_s$ is such that $\Gamma \vdash_s \alpha$ if and only if $T_s \cup \Gamma \models \alpha$ where T_s is a suitable axiomatization for partial orderings as that in Figure 1. Notice that this theory is decidable and thus we assume that the *server* can decide whether $\Gamma \vdash_s \alpha$ for for any $\Gamma \in \mathcal{P}(L_s)$ and $\alpha \in L_s$.

Let us consider the situation in which the *client* must establish whether

$$\{a \subseteq b, b \subseteq c\} \vdash_c (a \cap c) \cup c = c \tag{1}$$

using the following facts as conditional rewrite rules

$$x \subseteq y \;\Rightarrow\; x \cap y = x \tag{2}$$
$$x \subseteq y \;\Rightarrow\; x \cup y = y \tag{3}$$

The conclusion of (2) allows us to reduce (1) to

$$\{a \subseteq b, b \subseteq c\} \vdash_c a \cup c = c \tag{4}$$

but in order to enable this rewriting step the condition $a \subseteq c$ must be shown to be a consequence of $\{a \subseteq b,\ b \subseteq c\}$ i.e. that:

$$\{a \subseteq b,\ b \subseteq c\} \vdash_c a \subseteq c \tag{5}$$

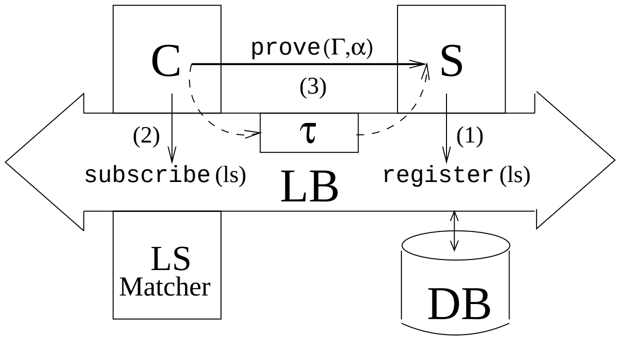


Figure 2. The Logic Broker Architecture

Let us now assume that the *client* fails to establish (5). Here is where the *server* comes into play. Let $(\cdot)^*$ be a bijective function from the terms of L_c to those of L_s and let $\tau : L_c \to L_s$ be such that $\tau(a \subseteq b) = a^* \preceq b^*$. It is easy to show that τ^{-1} is a morphism. Therefore by applying τ (5) can be reduced to

$$\{a^* \preceq b^*,\ b^* \preceq c^*\} \vdash_s a^* \preceq c^* \tag{6}$$

Since the *server* is a decision procedure for T_s, then (6) can be easily solved by the *server* side. By the defining properties of morphisms of consequence relations, (5) must necessarily hold. This enables the application of (2) at the *client* side. The application of (3) to the (4) goes along the same lines and concludes the problem. Thus, the original problem (1) is easily solved at the *client* side by this *client-server* combination.

Remark. Notice that if we regard the reasoning systems as agents knowledgeable in their respective domain, then the compatibility problem becomes the ontology problem for the agents; furthermore, the morphism becomes a function which translates the ontology of the server into that of the *client* thereby making the agents capable of meaningful interaction. We will come back on this issue later in Section 4.

3 The Logic Broker Architecture

The basic schema of the Logic Broker Architecture is depicted in Figure 2, where a *client* reasoning system C gets access to the services provided by a *server* S via the Logic Broker LB. The rôle of the LB is threefold:

1. the LB provides location transparency for the reasoning systems by routing messages to systems without requiring senders to know the locations of the receivers,
2. the LB facilitates interoperation by providing a way for systems to discover which servers can handle which requests using a simple registration/subscription mechanism, and
3. the LB automatically translates the requests for logical services issued by the *client* into corresponding requests for the server.

Since location transparency is a domain-independent feature which can be readily obtained by using general purpose and well established frameworks such as CORBA, here we focus last two (domain specific) issues.

3.1 The registration/subscription mechanism

A reasoning system S can register to the LB as a *server* by sending the LB a message of the form `register`(LS_s), where LS_s is a specification of the logical service it is able and willing to provide (e.g. `prove`, `simplify`, ...); notice that the specification of a logical service comes equipped with the specification of the associated consequence relation. When LB receives such a message it stores the pair $\langle \text{S}, \text{LS}_s \rangle$ in a database of registered logical services for later use.

Dually, a reasoning system can subscribe to the LB as a *client* by issuing the LB a message of the form `subscribe`(LS_c), where LS_c is a specification of the requested service. Upon receipt of such a message, the LB searches the database of registered services for a pair $\langle \text{S}, \text{LS}_s \rangle$ such that the specification of the logical service LS_s has the same interface of the required one, and checks if there exists a morphism ϕ from $(L_s, \vdash_s)$ into $(L_c, \vdash_c)$. This last task is carried out by a specialized reasoning module called the *Logical Service Matcher* (*LS Matcher* for short).[3]

If both steps terminate successfully then the LB establishes a connection between the *client* and the *server*.

3.2 The translation mechanism

Let ϕ be the morphism determined by the LS Matcher as explained in Section 3.1. The LB defines a function $\tau : L_c \to L_s$ which translates a formula α_c of the *client* into a formula α_s of the *server* such that $\phi(\alpha_s) = \alpha_c$ if α_c is in the range of ϕ and it is undefined otherwise.

Whenever the *client* issues a message of the form `prove`(Γ, α), then the LB transparently carries out the following steps:

1. Γ and α are translated into $\tau(\Gamma)$ and $\tau(\alpha)$ respectively,
2. the message `prove`$(\tau(\Gamma),\tau(\alpha))$ is delivered to the selected *server*, and

[3]Notice that the problem of finding a morphism between two consequence relations is not decidable in the general case and therefore user intervention may be necessary here.

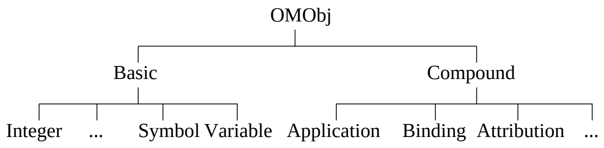


Figure 3. The hierarchy of the OpenMath Objects.

3. the answer returned by the *server* is delivered back to the *client*.

It readily follows from the definition of morphism that whenever the *client* gets a positive answer in reply to a message of the form `prove`(Γ, α), then $\Gamma \vdash_c \alpha$ holds. This is an important result which ensures the logical soundness of the interaction between the *client* and the *server*.

4 A Prototype Implementation of the Logic Broker Architecture

We have built a first running prototype of the Logic Broker Architecture using CORBA. CORBA is the candidate of choice for many reasons. First of all CORBA greatly simplifies the activity of combining reasoning systems since it is specifically designed to reduce to a minimum the burden associated with the activity of interfacing software systems. Secondly, CORBA gives us location transparency for free. Last but not least, CORBA is an industrial standard and many ORB implementations (both free and commercial) are publicly available.

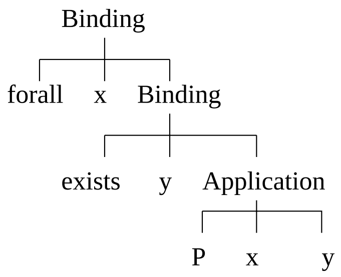
We recall from the previous sections that the main difficulty in matching logical services is that of finding a morphism between the associated consequence relations. Unfortunately this problem is undecidable and therefore a solution which is both general and fully automatic is not possible. The solution we adopted is based on the an emerging standard for representing mathematical knowledge called OpenMath [9]. In particular we exploited the following features of the OpenMath standard:

- OpenMath provides a common representation for mathematical expressions via OpenMath objects. An OpenMath *object* is a mathematical expression built out of primitive constructs for atomic entities (such as symbols, variables, and numbers) and constructs for compound objects such as applications, bindings, and attributions (i.e. annotations) as illustrated in Figure 3. The OpenMath representation for the formula $\forall x.\exists y.P(x,y)$ is depicted in Figure 4.

- OpenMath provides a standard ontology for mathematical domains based on the notion of *content dictionary* (CD for short). CDs specify the semantic of OpenMath symbols by providing the following information:

  – the *name* of the symbol
  – the *description* in natural language
  – an *example*
  – a *signature*
  – a *commented mathematical property* (CMP for short)
  – a *formal mathematical property* (FMP for short)

  The first two items are strings of characters whereas the last four are OpenMath objects. While both the *example* and the CMP specify the semantic at an informal level, the *signature* and the FMP provide the formal semantics. The entry of the OpenMath CD `arith1` for the OpenMath Symbol `plus` is given in Figure 5.



Tree representation of OpenMath object representing the formula $\forall x.\exists y.P(x,y)$. "forall" and "exists" are OpenMath Symbols defined in the Content Dictionary "quant1" while "x" and "y' are OpenMath variables. See [21] for the details.

Figure 4. Example of OpenMath Object

Name: `plus`

Description: `An nary commutative function plus.`

Commented Mathematical property (CMP): `a + b = b + a`

Formal Mathematical property (FMP):

```
forall [ a b ] . (eq (plus (a, b), plus (b, a)))
```

Figure 5. The definition of the OpenMath Symbol `plus` in the Content Dictionary `arith1`

The choice of the OPENMATH standard[4] simplified considerably the development of our prototype. The main implementation effort amounted to the definition of the interfaces for OPENMATH objects, the Logical Services, the Reasoning Systems (both *client* and *server*), the Logic Broker and the LS Matcher using the OMG Interface Definition Language (IDL for short).

In order to connect to reasoning systems via our prototype implementation of the Logic Broker Architecture, the reasoning systems are required to translate mathematical statements expressed in their respective logical language into/from equivalent OPENMATH objects so to preserve the semantics encoded in the content dictionaries. In practice this means that the reasoning system must be encapsulated by a *wrapper* in charge of the translation. As in the general case, the big advantage of using a standard communication language is that a single translation for each private language to the standard is necessary as opposed to the general case in which a translation for each pair of private languages is needed.

5 Related Work

A number of approaches aiming at the smooth and/or effective combination of mechanized reasoning systems is available in the literature. TeamWork [10] is an approach for the knowledge-based distribution of search which has been successfully applied to solve equational deduction problems. ILF [8] is a framework for which supports the cooperation of provers. For the lack of space here we focus on the approach which is more related to ours, namely the MathWeb architecture.

MathWeb [12] is a distributed network architecture for automated and interactive theorem proving aiming at supporting modularization, interoperability, robustness, and scalability of mathematical software systems. The main features of MathWeb are largely complementary to those of the Logic Broker Architecture. In MathWeb the activity of combining logical services is simplified considerably thanks to the adoption of the Agent-Orient Programming paradigm. On the other hand, MathWeb makes no provision to ensure the soundness of the combination of the logical services, whereas this is one of the main features of the Logic Broker Architecture. Furthermore, MathWeb agents are required to employ a MathWeb specific protocol based on an XML encoding of (a subset of) KQML performatives. This is usually achieved by encapsulating the reasoning systems into wrappers implementing the protocol. In our prototype implementation of the Logic Broker Architecture the addition of new reasoning systems is a considerably simpler activity thanks to the use of CORBA. In the future work we plan to lift the Logic Broker Architecture to an agent-based architecture [14].

The Open Mechanized Reasoning Systems (OMRS for short) project [15, 1] aims at the definition of a specification framework for specifying logical services. Specifications play a fundamental rôle when automatic and/or provably sound integrations are at stake. The incorporation of the OMRS specification framework into the Logic Broker Architecture is part of our future work.

[4]We deviate from the OPENMATH standard in that we do not commit to specialized encodings for OPENMATH objects and content dictionaries. In the OPENMATH ESPRIT Project a big effort is put in building language-specific libraries (both in C and JAVA) for two specific encodings (Binary and XML). This is not necessary in our framework since the CORBA frees the programmer from the burden of dealing with the actual encoding of the information exchanged.

6 Conclusions

We have presented the *Logic Broker Architecture*, a framework which provides the middle-ware for making mechanized reasoning systems interoperable. We have shown that the architecture provides location transparency, a common syntax for representing mathematical information, and a way to forward requests for logical services to appropriate reasoning systems via a simple registration/subscription mechanism.

References

[1] A. Armando, A. Coglio, and F. Giunchiglia. The Control Component of Open Mechanized Reasoning Systems. *Electronic Notes in Theoretical Computer Science*, 23(3):3–20, 1999.

[2] Alessandro Armando and Silvio Ranise. From integrated reasoning specialists to ”plug-and-play” reasoning components. In Jaques Calmet and Jan Plaza, editors, *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)*, volume 1476 of *LNAI*, pages 42–54, Berlin, September 16–18 1998. Springer.

[3] A. Avron. Simple consequence relations. LFCS Report Series, Laboratory for the Foundations of Computer Science, Computer Science Department, University of Edinburgh, 1987.

[4] C. Ballarin, K. Homann, and J. Calmet. Theorems and Algorithms: An Interface between Isabelle and Maple. In A. H. M. Levelt, editor, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, pages 150–157, Berkeley, CA, USA, 1995. ACM Press.

[5] B. Buchberger and T. Jebelean, editors. *Proceedings of the Second International Theorema Workshop*, RISC-Reports series No. 98-10. RISC-Hagenberg, Austria, 29-30 June, 1998.

[6] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuţa, and D. Văsaru. An Overview of the *Theorema* Project. In Küchlin [19], pages 384–391.

[7] B. Buchberger, D. Văsaru, and T. Ida, editors. *Proceedings of the First International Theorema Workshop*, RISC-Reports series No. 97-20. RISC-Hagenberg, Austria, 9-10 June, 1997.

[8] B. I. Dahn, J. Gehne, T. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ILF. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 57–60, Berlin, July 13–17 1997. Springer.

[9] S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 implementation. In Küchlin [19], pages 241–248.

[10] J. Denzinger and I. Dahn. Cooperating theorem provers. In Wolfgang Bibel and Peter H. Schmidt, editors, *Automated Deduction: A Basis for Applications. Volume II, Systems and Implementation Techniques*. Kluwer Academic Publishers, Dordrecht, 1998.

[11] A. Franke, S.M. Hess, C.G. Jung, M. Kohlhase, and V. Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 5:156–187, 1999.

[12] Andreas Franke and Michael Kohlhase. System description: MATHWEB, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *LNAI*, pages 217–221, Berlin, July 7–10, 1999. Springer-Verlag.

[13] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the Association for Computing Machinery*, 37(7), July 1994.

[14] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[15] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories: Towards an Architecture for Open Mechanized Reasoning Systems. Technical Report 9409-15, IRST, Trento, Italy, 1994. Also published as Stanford Computer Science Department Technical note number STAN-CS-TN-94-15, Stanford University. Short version published in Proc. of the First International Workshop on Frontiers of Combining Systems (FroCoS'96), Munich, Germany, March 1996.

[16] J. Harrison and L. Théry. Extending the HOL Theorem Prover with a Computer Algebra System to Reason about the Reals. pages 174–184.

[17] M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra with Proof Planning. In Jaques Calmet and Carla Limongelli, editors, *Design and Implementation of Symbolic Computation Systems; International Symposium, DISCO '96; Proceedings*, volume 1128 of *LNCS*, Karlsruhe, Germany, september 18-20 1996. Springer Verlag, Berlin, Germany.

[18] M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[19] W. Küchlin, editor. *Symbolic and algebraic computation (ISSAC-97) : international symposium, Kihei, Maui, HI, USA, July 21-23, 1997*, New York, NY, 1997. ACM Press.

[20] E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand Reinhold, Amsterdam, 2 edition, 1985.

[21] D. P. Carlisle O. Caprotti and A. M. Cohen. The OpenMath Standard. Technical Report D1.3.3a (Draft) (Public), The OpenMath Esprit Consortium, August 1999.