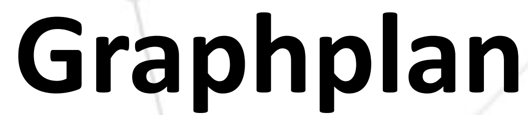
Pianificazione STRIPS basata su grafi

Michele Lombardi <michele.lombardi2@unibo.it>

L’esercitazione/lezione di oggi

Argomento: Pianificazione basata su grafi & derivati

- Graphplan
  - Richiami su pianification grafica
  - Graphplan in azione
- Fast Forward
  - Richiami + FF in azione
- Modellazione PDDL
  - Esercizio PDDL #1
- SATPlan & Blackbox
  - Pianificazione come problema di soddisfacibilità
  - Blackbox in azione
  - Esercizio PDDL #2



Pianificazione basata su grafi

Graphplan

Cos'è Graphplan?

- E' un pianificatore per problemi tipo STRIPS introdotto da Blum & Furst (CMU) nel 1995
- Basato su una struttura dati detta planning graph
- Pianificazione come ricerca sul planning graph
- E' un pianificatore off-line
- E' completo e produce sempre il piano più corto possibile
- Produce piani parzialmente ordinati

Pianificazione STRIPS like

- **Le azioni si rappresentano come quelle di STRIPS** con

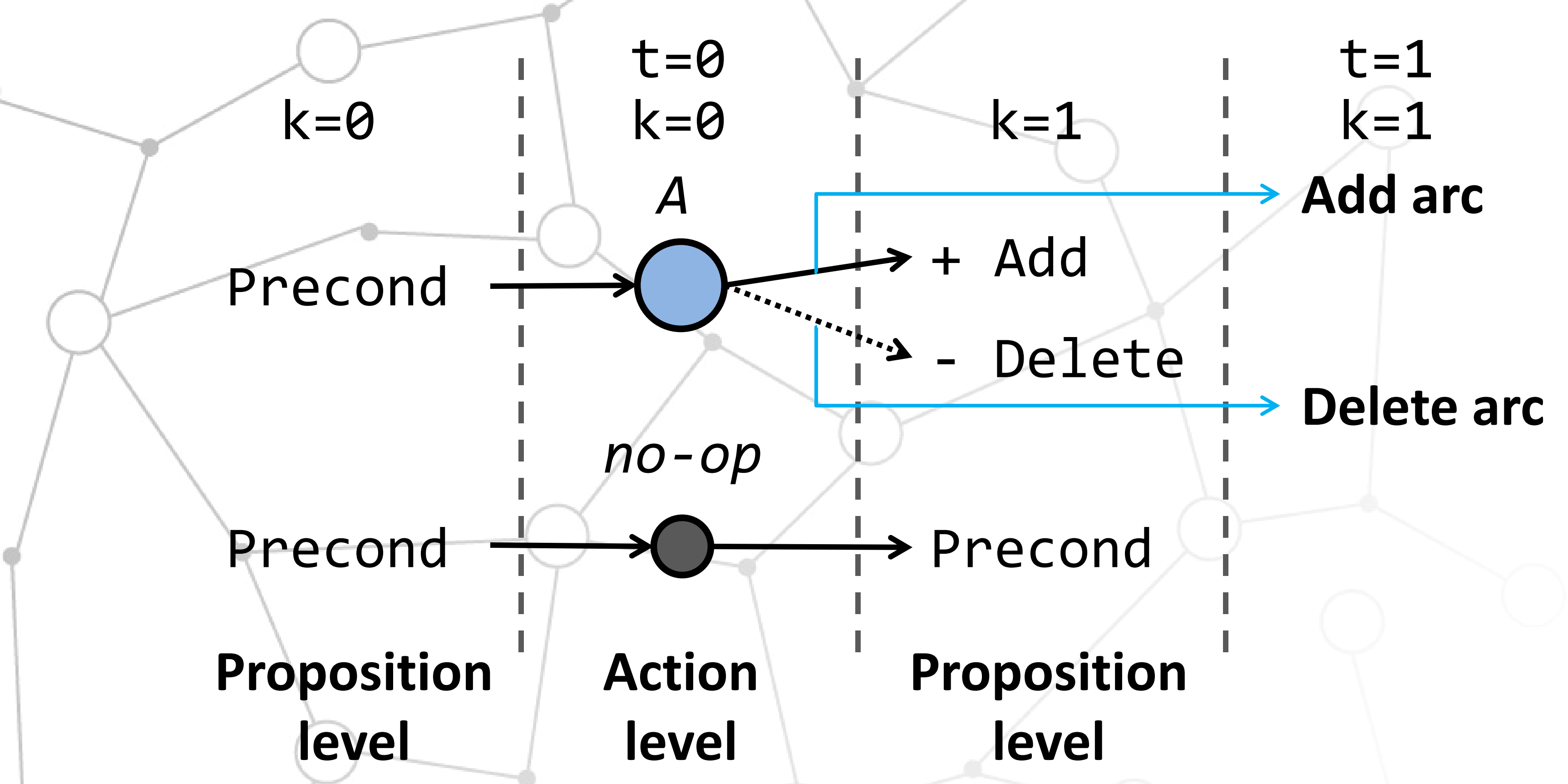
  PRECONDIZIONI
  ADD LIST
  DELETE LIST

- E' implicitamente definita una azione **no-op** che non modifica lo stato

- **Stato = oggetti + insieme di predicati**
  Gli oggetti hanno un tipo (typed)!

Planning graph

- Il planning graph è un **grafo diretto a livelli:**
  - Proposition levels, contenenti proposition nodes
  - Action levels, contenti action nodes
- Il livello 0 corrisponde allo stato iniziale (è un proposition level)
- Gli archi si dividono in:
  - Archi *precodizione* (proposition → action)
  - Archi *add* (action→ proposition)
  - Archi *delete* (action→ proposition)



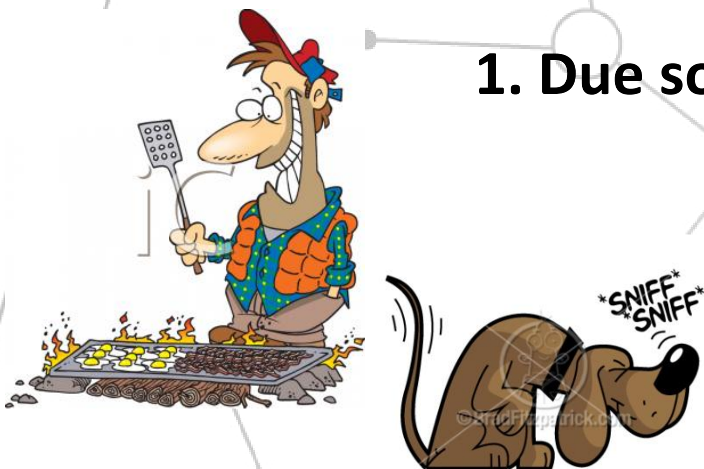
- Si può inserire una azione all'action level *k* se le sue precondizioni sono presenti al proposition level *k*
- Ogni action level contiene tutte le azioni applicabili
- Azioni fittizie "no-op" traslano le proposizioni di un time step al successivo

Costruzione del planning graph

Algoritmo per la costruzione passo a passo del planning graph:

1. **INIZIALIZZAZIONE:**
   Aggiungi le proposizioni vere nello stato iniziale
2. **CREAZIONE DELL'ACTION LEVEL *k*:**
   1. Inserisci i no-op (uno per proposizione) del prop. level *k*
   2. Aggiungi le azioni applicabili
      (precondizioni non mutuamente esclusive)
3. **CREAZIONE DEL PROPOSITION LEVEL *k*:**
   1. Aggiungi gli effetti dei no-op (proposizioni a livello *k-1*)
   2. Aggiungi gli add effects del resto dell'action level *k*
   3. Aggiungi i delete arcs

Un esempio classico



1. Due soggetti

2. Un luogo
di partenza

3. Un mezzo
di trasporto

4. Un luogo di
destinazione

Un esempio classico

Azioni:

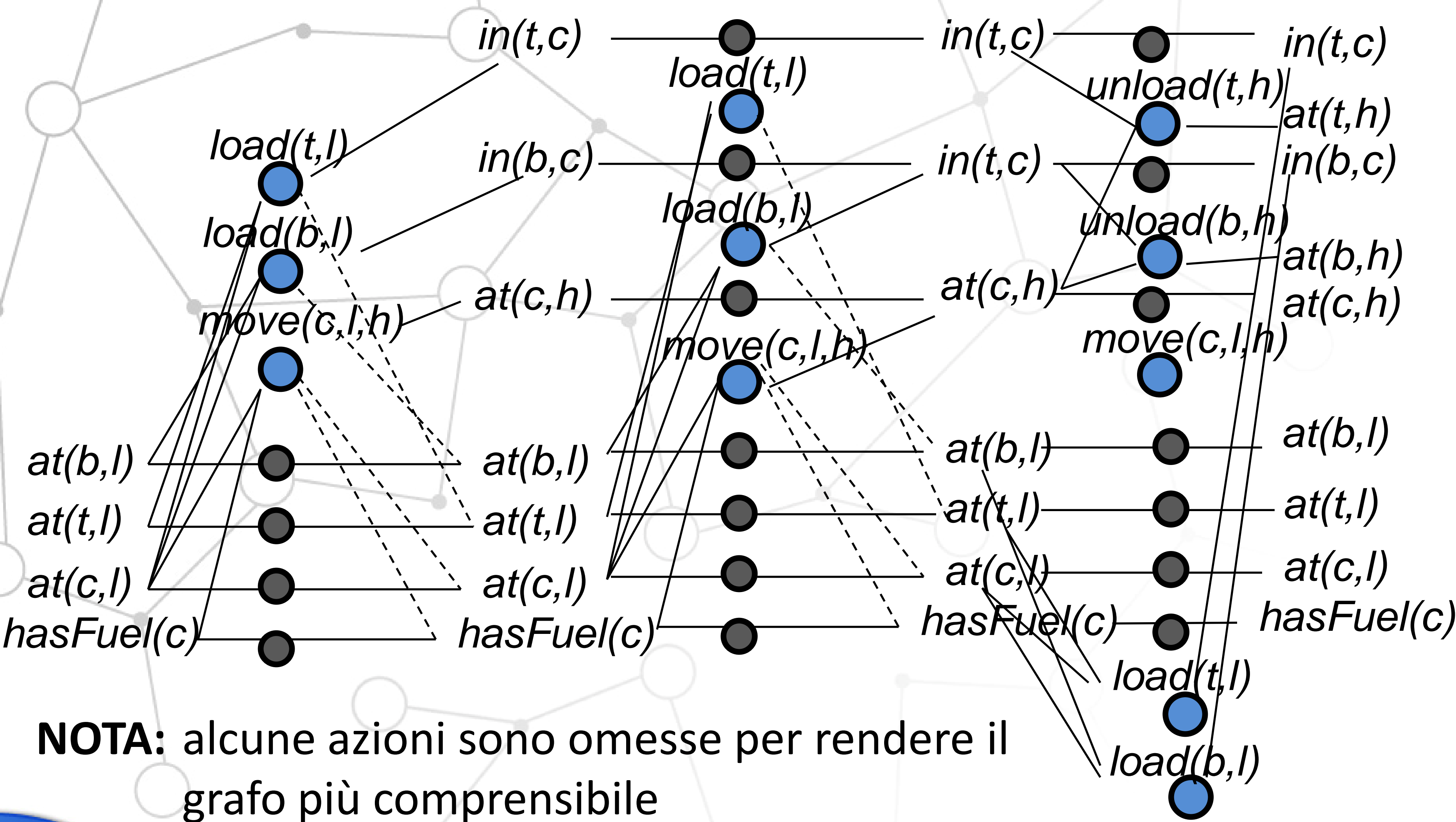
- **MOVE(R, PosA, PosB)**
  PRECONDIZIONI: at(R,PosA), hasFuel(R)
  ADD LIST: at(R,PosB)
  DELETE LIST: at(R,PosA), hasFuel(R)
- **LOAD(Oggetto, R, Pos)**
  PRECONDIZIONI: at(R,Pos), at(Oggetto,Pos)
  ADD LIST: in(R,Oggetto)
  DELETE LIST: at(Oggetto,Pos)
- **UNLOAD(Oggetto, Pos)**
  PRECONDIZIONI: in(R,Oggetto), at(R,Pos)
  ADD LIST: at(Oggetto,Pos)
  DELETE LIST: in(R,Oggetto)

Un esempio classico

Tipi, stato, goal:

- **OGGETTI:**
  ```
  carrello: car (c)
  oggetti: jack (j), bobby (t)
  locazioni: home (h), mushrooms (m)
  ```
- **STATO INIZIALE:**
  ```
  at(j,h)
  at(b,h)
  at(c,h)
  hasFuel(c)
  ```
- **GOAL:**
  ```
  at(j, m)
  at(b, m)
  ```

Per ogni operatore/proposizione ad ogni time step viene calcolata una **lista di mutua esclusione**



NOTA: alcune azioni sono omesse per rendere il grafo più comprensibile

Inconsistenze

Inconsistenza tra **azioni**:

- Interferenza (interference): una azione nega un effetto o una precondizione di un'altra
- Requisiti in competizioni (competing needs): due azioni hanno precondizioni mutuamente esclusive

Due **proposizioni** sono mutuamente esclusive:

- Se sono uno la negazione del'altro
- Inconsistenze sul dominio
- Se tutti i possibili modi per raggiungerli coinvolgono azioni mutuamente esclusive

Estrazione di un piano

Valid plan

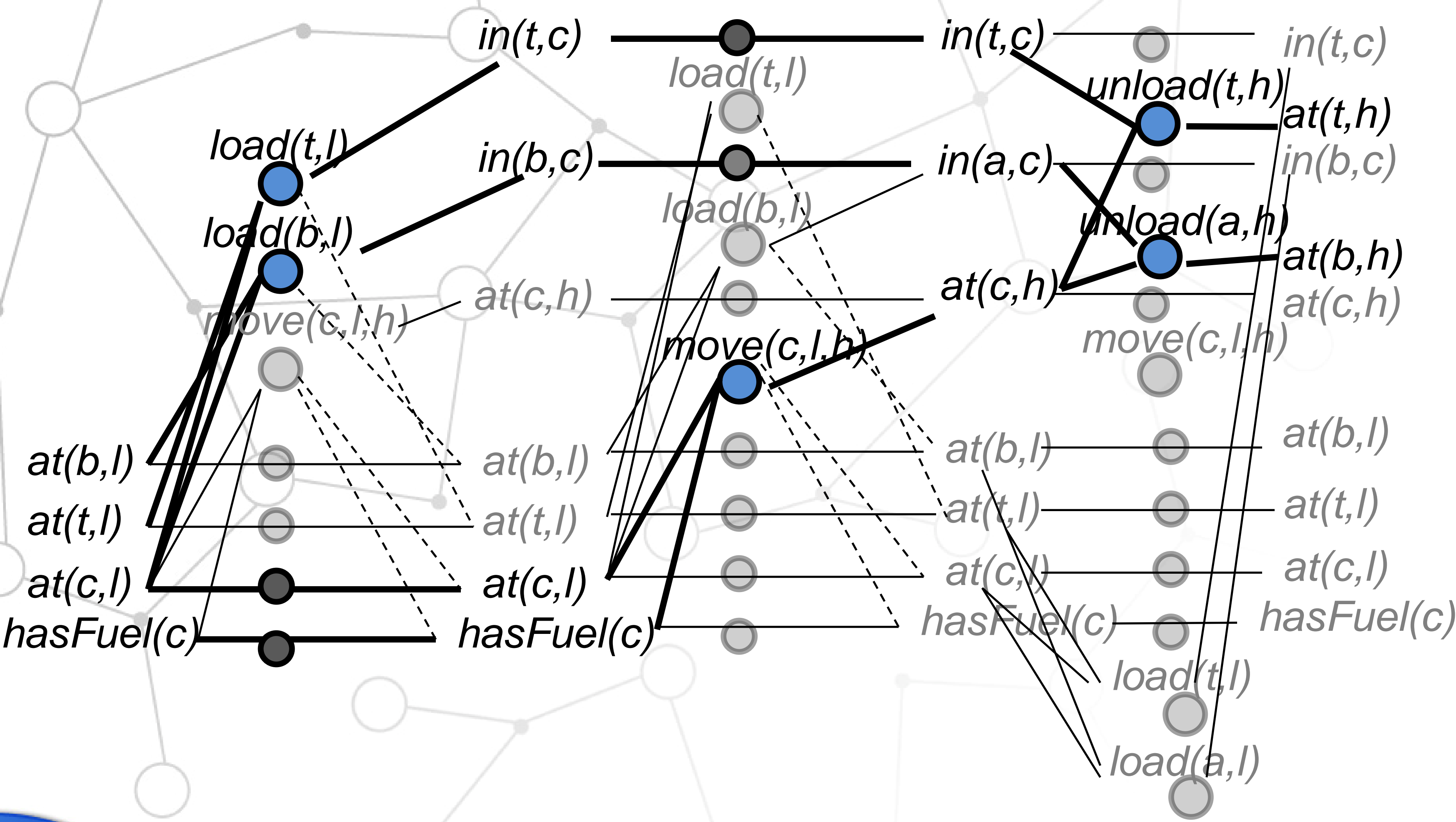
un sottografo connesso e consistente del planning graph

- Azioni allo stesso time step del valid plan possono essere eseguite in qualunque ordine (non interferiscono)
- Proposizioni allo stesso time step del valid plan sono non mutuamente esclusive
- L'ultimo time step contiene tutti i letterali del goal e questi non sono marcati come mutuamente esclusivi

La ricerca di un valid plan

- procede a ritroso a partire dai goals
- Ad ogni step viene selezionato un <u>insieme</u> di azioni

Un esempio di **valid plan**



Graphplan: algoritmo

```
function GRAPHPLAN(problema):
         grafo = GRAFO_INIZIALE(problema)
         obiettivi = GOAL(problema)
         loop do:
            if obiettivi non mutex nell’ultimo step:
               Sol = ESTRAI_ SOLUZIONE(grafo, obiettivi)
               if Sol ≠ fail: return Sol
               else if LEVEL_OFF(grafo): return fail
            grafo = ESPANDI_GRAFO(grafo, problema)
```

- **Memoization** (non è un errore tipografico!)
  Se si determina che un sottoinsieme di goals non è soddisfacibile, graphplan salva questo risultato in una hash table, per evitare di fare la stessa scelta in futuro.

Esercizio

Avviare Graphplan

- Potete trovare Graphplan su
  `/afs/reinbow.ing.unibo.it/software/linux/ia2010/`
- Ma vi conviene farvi un link in un posto comodo (la vostra home? Il desktop?)
  ln –s ... <percorso> <link name>
- Eseguire Graphplan
  ```
  cd <percorso>
  ./graphplan -h
  ```
- Scaricate dal sito del corso gli esercizi su Graphplan; una volta estratto l'archivio trovate:
  ```
  ops file: cart.gp.ops
  fact file: cart.gp.facts
  ```

Esercizio

Vediamo la costruzione del grafo in azione...

- Provate a risolvere il problema del carrello con Graphplan
- Aumentando mano a mano il livello di dettaglio dell'output, cercate di identificare:
  - Il grafo via via prodotto
  - Le liste di mutua esclusione

Esercizio

Vediamo la ricerca di un valid plan in azione...

- Provate a risolvere il problema del carrello con Graphplan
- Aumentando mano a mano il livello di dettaglio dell'output, cercate di identificare:
  - Il punto in cui avviene la ricerca
  - Se viene usata la memoization
  - Provate a disabilitare l'utilizzo delle mutue esclusioni; cosa vi aspettate che succeda? Cosa succede effettivamente?

FF: Fast Forward

Pianificazione basata su grafi come euristica

FF: Fast Forward

Fast Forward
FF è un pianificatore euristico <u>estremamente efficiente</u> introdotto da Hoffmann nel 2000

- Euristico = ad ogni stato S è una valutazione della distanza dal goal mediante una funzione euristica

Funzionamento base: hill climbing + A*

1. A partire da uno stato S, si esaminano tutti i successori S'
2. Se si individua uno stato successore S* migliore di S, ci si sposta su di esso e torna al punto 1
3. Se non si trova alcuno stato con valutazione migliore, viene eseguita una ricerca completa A*, usando la stessa euristica

Esercizi

Avviare FF

- Potete trovare FF su `/afs/reinbow.ing.unibo.it/software/linux/ia2010/`
- Decomprimere l’archivio, poi fate un link simbolico al programma come nel caso precedente
- Provare ad eseguire il programma utilizzando come input i due files .pddl e .facts che avete trovato nell’archivio

PDDL

Planning Domain Definition Language

Tentativo di standardizzazione di un linguaggio per modellare problemi di planning

- Sviluppato come supporto all'International Planning Competition

Una definizione PDDL consiste di due parti

- Un dominio (operatori + oggetti)
- Un problema

Tipicamente in file separati

PDDL - Domain

Sintassi (approssimativa) di una definizione di dominio

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

PDDL - Domain

Sintassi (approssimativa) di una definizione di dominio

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

caratteristiche del linguaggio effettivamente utilizzate

PDDL - Domain

Sintassi (approssimativa) di una definizione di dominio

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Tipi di oggetto che fanno parte del dominio

PDDL - Domain

Sintassi (approssimativa) di una definizione di dominio

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Tipi di predicato; e.g. at(home,car)

PDDL - Domain

Sintassi (approssimativa) di una definizione di dominio

```
(define (domain DOMAIN_NAME)
(:requirements [:strips] [:equality] [:typing] [:adl])
[(:types NAME_1 ... NAME_N)]
(:predicates
      (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
      ...
)

(:action ACTION_1_NAME
[:parameters (?P1 ?P2 ... ?PN)]
[:precondition PRECOND_FORMULA]
[:effect EFFECT_FORMULA] )
...
)
```

Descrizione degli operatori

PDDL – Predicati e azioni

Sintassi per le variabili con tipo:

```
?<var name> - <type>
```

Sintassi di una formula

Aggregazione di predicati mediante operatori logici:

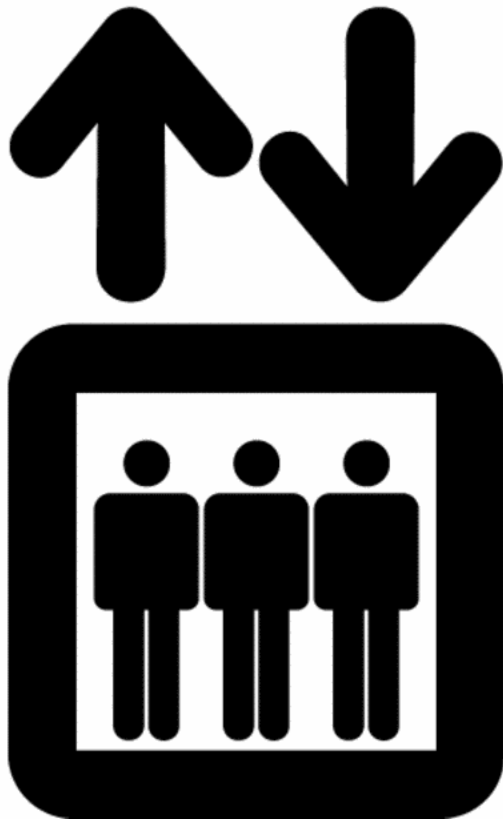
```
(and PREDICATE_1 ... PREDICATE_N)
(or PREDICATE_1 ... PREDICATE_N)
(not PREDICATE)
```

Effetti

Add e delete effect sono aggregati (i delete effect sono predicati in "not" nella formula:

```
:effect FORMULA
```

Esercizio



- Un palazzo ha 5 piani, 5 inquilini e due ascensori
- Ad un’ora X di un giorno Y ogni inquilino si trova ad un piano e deve arrivare ad un altro
- Occorre trovare un piano di spostamento che porti ogni inquilino alla sua destinazione

Si modelli il problema in PDDL e lo si risolva con FF. Occore fare in modo che il pianificatore produca mantenga basso tempo di spostamento degli ascensori.

Gli inquilini

Il giovane **Enrico** è al piano terra dopo una mattinata faticosa e torna a casa al 3° piano

L'avvenente **Elena** è al piano terra e riporta soddisfatta i suoi acquisti al 2° piano

Dopo un pranzo in famiglia al 3° piano, **Ettore** torn al al lavoro nell'officina al pianterreno

Elvira ed **Ennio** sono al 1° (a giocare a briscola) e al 2° (a comporre) e tornano al 4° piano

Blackbox

Unificare pianificazione basata su grafi e pianificazione come soddisfacibilità

SATPLAN

SATPLAN
Nel 92 Selman e Kautz introducono un approccio alla pianificazione basato sulla ricerca di un assegnamento di variabili logiche indicanti una selezione di azioni che soddisfi una serie di vincoli, piuttosto che sulla deduzione

APPROCCIO DEDUTTIVO
situation calculus – Formulazione di Green

- Data una serie di regole di deduzione in first order logic...
- ...e un insieme di fluent inizialmente validi...
- ...si cerca di dedurre i goals

Idea: modellarlo come un SATisfiability problem (SAT)

SATPLAN

SATisfiability problem

Problema di decidere se una formula logica sia soddisfacibile; è definito da:

- X = insieme delle variabili (0-1)
- F = formula logica

PRO: esistono risolutori **molto** efficienti per SAT che diventerebbero utilizzabili come pianificatori

CON: La codifica diretta della formulazione di Green, tuttavia, pone alcuni problemi...

SATPLAN

1. **Le regole di deduzione sono generali** (ex. sono parametriche, quantificate universalmente)

```
∀x, y: clear(x,s) ∧ clear(y,s)
            ⇒ on(x,y,move(x,y,s)) ∧ clear(x,move(x,y,s))
```

...ma si applicano ad un numero finito di oggetti! Potremmo introdurre una variabile logica per ogni possibile unificazione di una azione con le sue precondizioni

→ **VARIABILE**

```
clear(b1,s) ∧ clear(b2,s) ∧ move(b1,b2,s)
   ⇒ on(b1,b2,move(b1,b2,s)) ∧ clear(b1, move(b1,b2,s))
clear(b1,s) ∧ clear(b3,s) ∧ move(b1,b3,s)
   ⇒ on(b1,b3,move(b1,b3,s)) ∧ clear(b1, move(b1,b3,s))
...
```

SATPLAN

2. Le regole di deduzione sono ricorsive

```
clear(b1,s) ∧ clear(b2,s) ∧ move(b1,b2,s)
    ⇒ on(b1,b2,move(b1,b2,s)) ∧ clear(b1, move(b1,b2,s))
```

...ma a noi interessano piani con un numero finito di azioni! Potremmo introdurre una variabile logica per ogni possibile unificazione di una azione con le sue precondizioni ad ogni time step.

```
clear(b1,t0) ∧ clear(b2,t0) ∧ move(b1,b2,t0)
                              ⇒ on(b1,b2,t1) ∧ clear(b1, t1)
clear(b1,t1) ∧ clear(b3,t1) ∧ move(b1,b2,t1)
                              ⇒ on(b1,b3,t2) ∧ clear(b1, t2)
...
```

SATPLAN

3. **Le regole possono ammettere soluzioni spurie** perché “falso” implica qualunque cosa. Ex. Se “move” viene selezionata ed uno dei “clear” è falso, gli effetti possono essere ancora veri!

```
clear(b1,t0) ∧ clear(b2,t0) ∧ move(b1,b2,t0)
                         ⇒ on(b1,b2,t1) ∧ clear(b1, t1)
```

...ma precondizioni ed effetti possono essere trattati in modo simmetrico:

```
move(b1,b2,t0) ⇒ clear(b1,t0) ∧ clear(b2,t0) ∧
                 on(b1,b2,t1) ∧ clear(b1, t1)
```

SATPLAN → BLACKBOX

- Con questi accorgimenti, per un certo numero di time step k:
  1. `problema = variabili logiche + vincoli logici(`$\wedge\vee\neg\Rightarrow$`)`
  2. `piano = un assegnamento delle variabili logiche`

SATPLAN nella sua prima versione effettuava questa conversione con k crescente, e risolveva il problema mediante un SAT solver.

BLACKBOX (Selman, Kautz 1999) migliora l'approccio <u>costruendo mano a mano un planning graph</u>, e <u>convertendo il planning graph</u> in un problema SAT.

Blackbox: algoritmo

```
function BLACKBOX(problema):
         grafo = GRAFO_INIZIALE(problema)
         obiettivi = GOAL(problema)
         loop do:
            if obiettivi non mutex nell'ultimo step:
               converti il grafo in un problema SAT
               Sol = RICERCA(problema SAT, risolutore)
               if sol Sol ≠ fail: return Sol
               else if LEVEL_OFF(grafo): return fail
            grafo = ESPANDI_GRAFO(grafo, problema)
```

- Rispetto a graphplan cambia solo il modo di fare ricerca (e la conversione, chiaramente)
- Si può usare qualunque SAT solver, o anche graphplan stesso

Esercizio

Avviare blackbox

- Potete trovare blackbox su `/afs/reinbow.ing.unibo.it/software/linux/ia2010/`
- Decomprimere l'archivio, poi fate un link simbolico al programma come nel caso precedente
- Provare ad eseguire il programma utilizzando come input i due files .pddl e .facts che avete trovato nell'archivio

Esercizio

Vediamo blackbox in azione...

1. Ad eseguire blackbox sull'esempio del carrello
2. Ad eseguire blackbox usando come solver Graphplan
3. Ad eseguire blackbox usando come solver CHAFF
4. A vedere la WFF (Well Formed Formula) in cui viene convertito il planning graph all'ultimo stadio (per questo destreggiatevi un po' tra la varie opzioni disponibili)

Esercizio

Modellare in pddl il **nine puzzle**

Stato iniziale

7	3	4
2		1
8	6	5

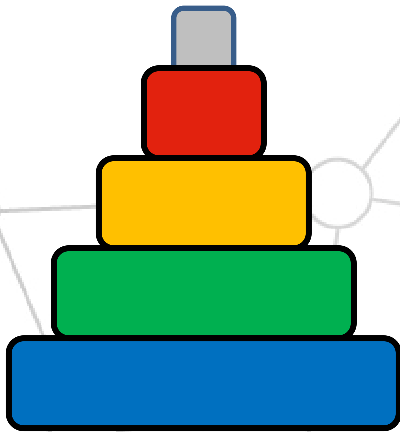
Goal

1	2	3
4	5	6
7	8	

- Provare a risolverlo con i vari pianificatori
- Come variano le performance “mescolando” un altro po’?

Esercizio

Modellare in pddl il puzzle della **torre di Hanoi**



- Si sposta un cerchio alla volta
- Un cerchio più piccolo non può mai andare su uno più grande
- Obiettivo: spostare i cerchi nello stesso ordina sull'ultima asta
- Provare a risolverlo con i vari pianificatori

Riferimenti

- GRAPHPLAN
  - http://www.cs.cmu.edu/~avrim/graphplan.html
  - A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis", Artificial Intelligence, 90:281--300 (1997).
- Blackbox
  - http://www.cs.rochester.edu/u/kautz/satplan/blackbox/index.html
  - SATPLAN: http://www.cs.rochester.edu/u/kautz/satplan/index.htm
  - Henry Kautz and Bart Selman, “Planning as Satisfiability”, Proceedings ECAI-92.
  - Henry Kautz and Bart Selman, “Unifying SAT-based and Graph-based Planning”, Proc. IJCAI-99, Stockholm, 1999.
- Fast Forward
  - http://members.deri.at/~joergh/ff.html
  - J. Hoffmann, “FF: The Fast-Forward Planning System”, in: AI Magazine, Volume 22, Number 3, 2001, Pages 57 - 62