PIANIFICAZIONE BASATA SU GRAFI

- Nel 1995 viene proposto da Blum e Furst CMU un nuovo pianificatore – GRAPHPLAN basato sui grafi

- Durante la pianificazione viene creato un particolare grafo detto Planning Graph
  - A ogni passo dell'albero di ricerca viene estesa questa struttura dati

- Inserisce il fattore TEMPO nella costruzione del piano.

- Pianificatore corretto e completo tra i più efficienti che siano stati costruiti

GRAPH PLAN: caratteristiche

- Usa la CLOSED WORLD ASSUMPTION quindi rientra nella categoria dei pianificatori off-line
- Graph-Plan restituisce il piano più corto possibile oppure restituisce una inconsistenza.
- Eredita dai pianificatori LINEARI il fatto di fare EARLY COMMITMENT: esempio l'azione A si svolge al time step 2
- Eredita dai pianificatori NON LINEARI, PARTIAL ORDER il fatto che i piani sono insiemi parzialmente ordinati di azioni: esempio nel time step 3 ci sono due azioni. Vengono quindi generati dei *piani paralleli*

GRAPH PLAN

- Le azioni si rappresentano come quelle di STRIPS con
  - PRECONDIZIONI
  - ADD LIST
  - DELETE LIST
- Gli oggetti hanno un tipo
- Esiste una azione **no-op** che non modifica lo stato
- Gli stati sono costituiti da predicati veri in quello stato.

PLANNING GRAPH

- Il planning graph è un **grafo diretto a livelli**
  - i nodi appartengono a livelli diversi
  - gli archi connettono nodi a livelli adiacenti.

- Il livello 0 corrisponde allo stato iniziale e alterna livelli proposizione e livelli azione corrispondenti a time step crescenti

- Nel planning graph possono esistere azioni e proposizioni in un time step t che interferiscono tra loro.

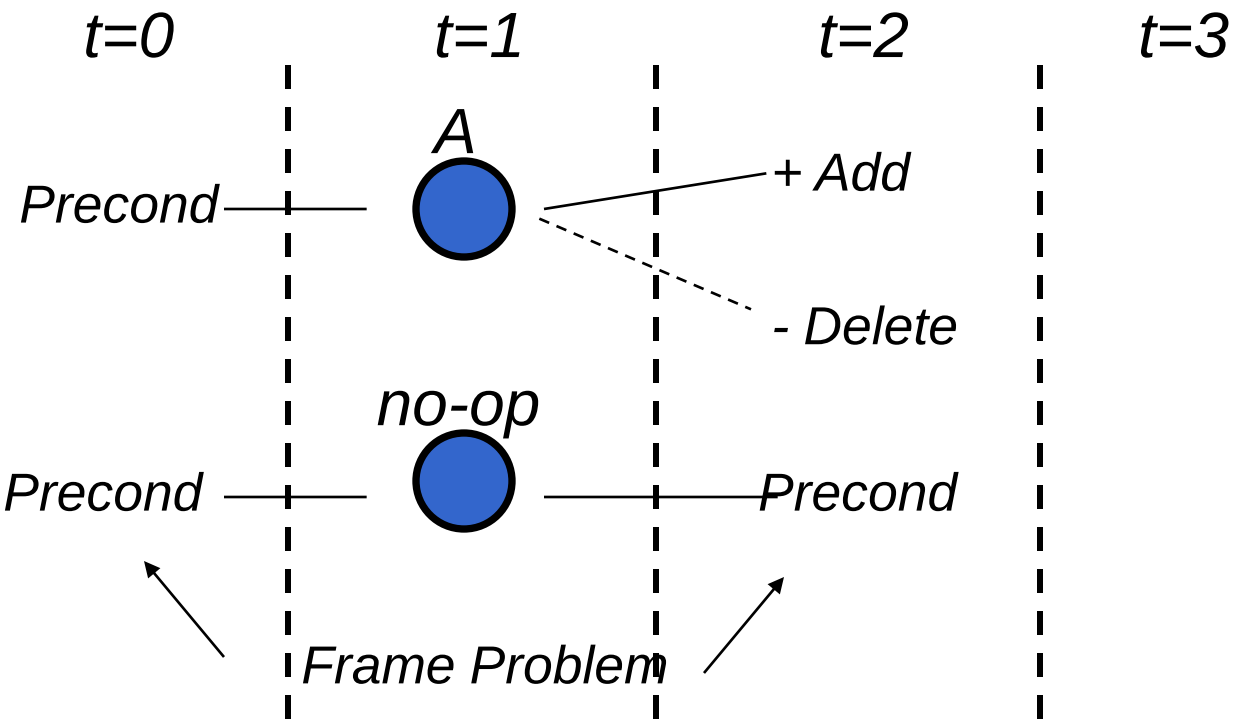
PLANNING GRAPH

- Nel **planning-graph** ci sono livelli diversi
  - proposition level: contenente proposition nodes
  - action level: contenente action nodes

- Livello 0 corrisponde allo stato iniziale ed è un proposition level

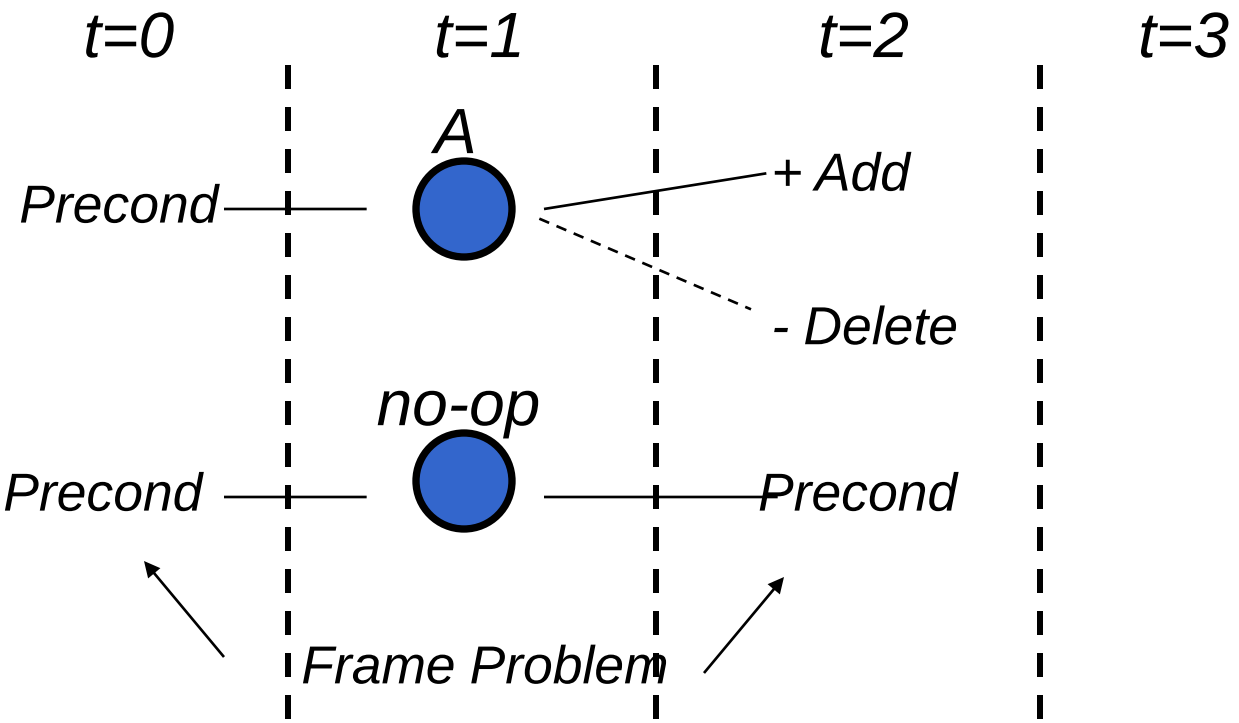
- Gli archi si dividono in:
  - archi precondizione (proposition → action)
  - archi add (action → proposition)
  - archi delete (action → proposition)

PLANNING GRAPH



A un certo time step si può inserire una azione se al time step precedente sono presenti le sue precondizioni

PLANNING GRAPH



Inoltre ci sono azioni fittizie che rappresentano inattività **no-op** che traslano le proposizioni del time step i al successivo

PLANNING GRAPH

- Ogni action level contiene
  - tutte le azioni che sono applicabili in quel time step
  - I vincoli che specificano quali coppie di azioni non possono essere eseguite contemporaneamente

- Ogni proposition level contiene tutti i letterali che potrebbero risultare da qualunque scelta di azioni nel time step precedente incluse le no-op

- **NOTA**: il processo di costruzione del planning graph non prevede che si faccia alcuna scelta per selezionare le azioni che costruiscono il piano.

INCONSISTENZE

Durante la costruzione del planning graph vengono individuate eventuali inconsistenze, in particolare

- Due **azioni** possono essere inconsistenti nello stesso time step
- Due **proposizioni** possono essere incosistenti nello stesso time step

In questo caso le azioni/proposizioni sono **mutuamente esclusive**

- Non possono comparire insieme in un piano
- **Ma possono comparire** nello stesso livello **nel planning graph**

INCONSISTENZE TRA AZIONI

- Effetti inconsistenti: una azione nega l'effetto di un'altra
  - L'azione move(part,dest) ha come effetto not at(part) mentre l'azione no-op su at(part) ha questo come effetto

- Interferenza: una azione cancella come effetto una precondizione dell'altra.
  - L'azione move(part,dest) ha come effetto not at(part) mentre l'azione no-op su at(part) ha questo come precondizione

- Competing needs: le azioni a e b hanno precondizioni mutuamente esclusive.
  - L'azione load(carico,mezzo) ha come precondizione in(carico, mezzo) mentre l'azione unload(carico, mezzo) ha come precondizione not in(carico, mezzo)

INCONSISTENZE TRA PROPOSIZIONI

- Due proposizioni sono inconsistenti se
  - una è la negazione dell'altra
  - Se tutti i modi per raggiungerle sono mutuamente esclusivi
  - Inoltre esistono regole che definiscono inconsistenze sul dominio
    - esistono regole quali ad esempio il fatto che un oggetto non può trovarsi in due luoghi contemporaneamente in uno stesso time step.

PLANNING GRAPH: ALGORITMO

- Il planning graph si costruisce nel modo seguente:
  - Tutte le proposizioni vere nello stato iniziale sono inserite nel primo *proposition level*

  CREAZIONE DELL'ACTION LEVEL

  - Per ogni operatore e ogni modo di unificare le sue precondizioni a proposizioni nel proposition level precedente, inserisci un action node SE due proposizioni non sono etichettate come **mutuamente esclusive**
  - Inoltre, per ogni proposizione nel proposition level precedente, inserisci un operatore no-op
  - Controlla gli action nodes cosi creati in modo tale che non interferiscano altrimenti marcali come esclusivi

PLANNING GRAPH: ALGORITMO

CREAZIONE DEL PROPOSITION LEVEL

- Per ogni action node nel action level precedente, aggiungi le proposizioni nella sua add list tramite archi non tratteggiati e inserisci archi tratteggiati per le proposizioni nella delete list
- Si faccia la stessa cosa per i no-op
- Marca come esclusive due proposizioni tali per cui tutti i modi per raggiungere la prima siano incompatibili con tutti i modi per raggiungere la seconda.

PLANNING GRAPH: ALGORITMO

Algoritmo per la costruzione passo a passo del planning graph:

1. **INIZIALIZZAZIONE:**
   Tutte le proposizioni vere nello stato iniziale sono inserite nel primo proposition level
2. **CREAZIONE DI UN ACTION LEVEL:**
   1. ∀operatore ∀ogni modo di unificare le sue precondizioni a proposizioni non mutuamente esclusive nel livello precedente, inserisci un action node
   2. Per ogni proposizione nel proposition level precedente, inserisci un operatore no-op
   3. Identifica le relazioni di mutua esclusione tra gli operatori appena costruiti

PLANNING GRAPH: ALGORITMO

Algoritmo per la costruzione passo a passo del planning graph:

3. **CREAZIONE DI UN PROPOSITION LEVEL:**
    1. Per ogni action node nel action level precedente, aggiungi le proposizioni nella sua add list tramite archi non tratteggiati
    2. Per ogni "no-op" nel livello precedente, aggiungi la proposizione corrispondente
    3. Per ogni action node nel action level precedente, collega proposizioni nella sua delete list tramite archi tratteggiati
    4. Marca come esclusive due proposizioni tali per cui tutti i modi per raggiungere la prima siano incompatibili con tutti i modi per raggiungere la seconda.

ESEMPIO

- Abbiamo un carrello R e due carichi A e B che si trovano nella posizione di partenza L e devono essere spostati nella posizione di destinazione P.

- Tre azioni (la sintassi la vediamo dopo)
  - MOVE(R,PosA, PosB)
  - LOAD(Pos, Oggetto)
  - UNLOAD(Pos, Oggetto)

ESEMPIO

- MOVE(R,PosA, PosB)
  - PRECONDIZIONI: at(R,PosA), div(PosA, PosB), hasFuel(R)
  - ADD LIST: at(R,PosB)
  - DELETE LIST: at(R,PosA), hasFuel(R)

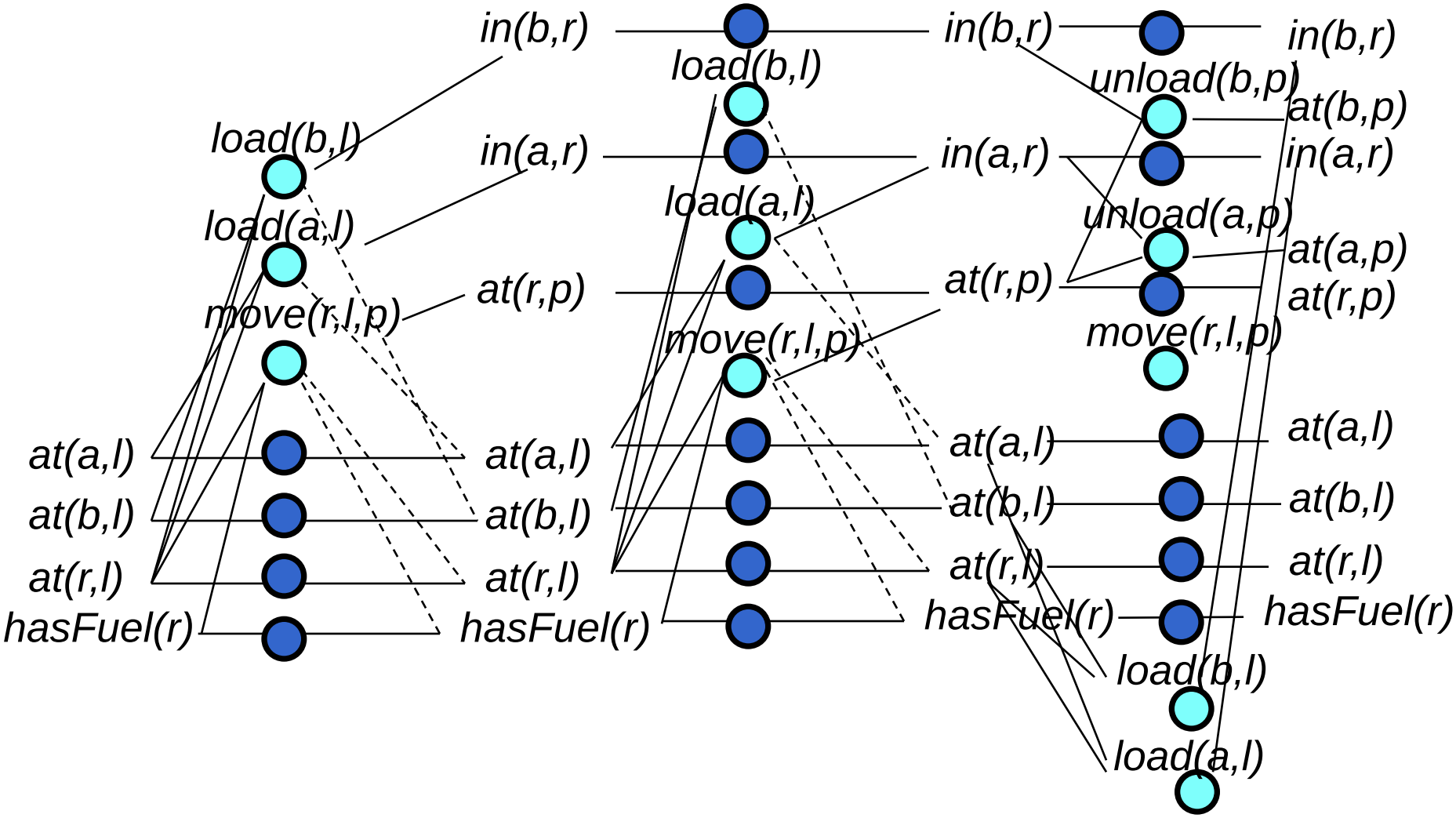
- LOAD(Oggetto, Pos)
  - PRECONDIZIONI: at(R,Pos), at(Oggetto,Pos)
  - ADD LIST: in(R,Oggetto)
  - DELETE LIST: at(Oggetto,Pos)

- UNLOAD(Oggetto,Pos)
  - PRECONDIZIONI: in(R,Oggetto), at(R,Pos)
  - ADD LIST: at(Oggetto,Pos)
  - DELETE LIST: in(R,Oggetto)

ESEMPIO

- Oggetti
  - carrello r
  - oggetti a, b
  - locazioni l, p
- Stato iniziale:
  - at(a,l)
  - at(b,l)
  - at(r,l)
  - hasFuel(r)
- Goal:
  - at(a,p)
  - at(b,p)

PLANNING GRAPH



ESTRAZIONE DI UN PIANO

- Una volta costruito il planning graph è possibile estrarre un piano detto **valid-plan,** un sottografo connesso e consistente del planning graph

- Caratteristiche valid plan

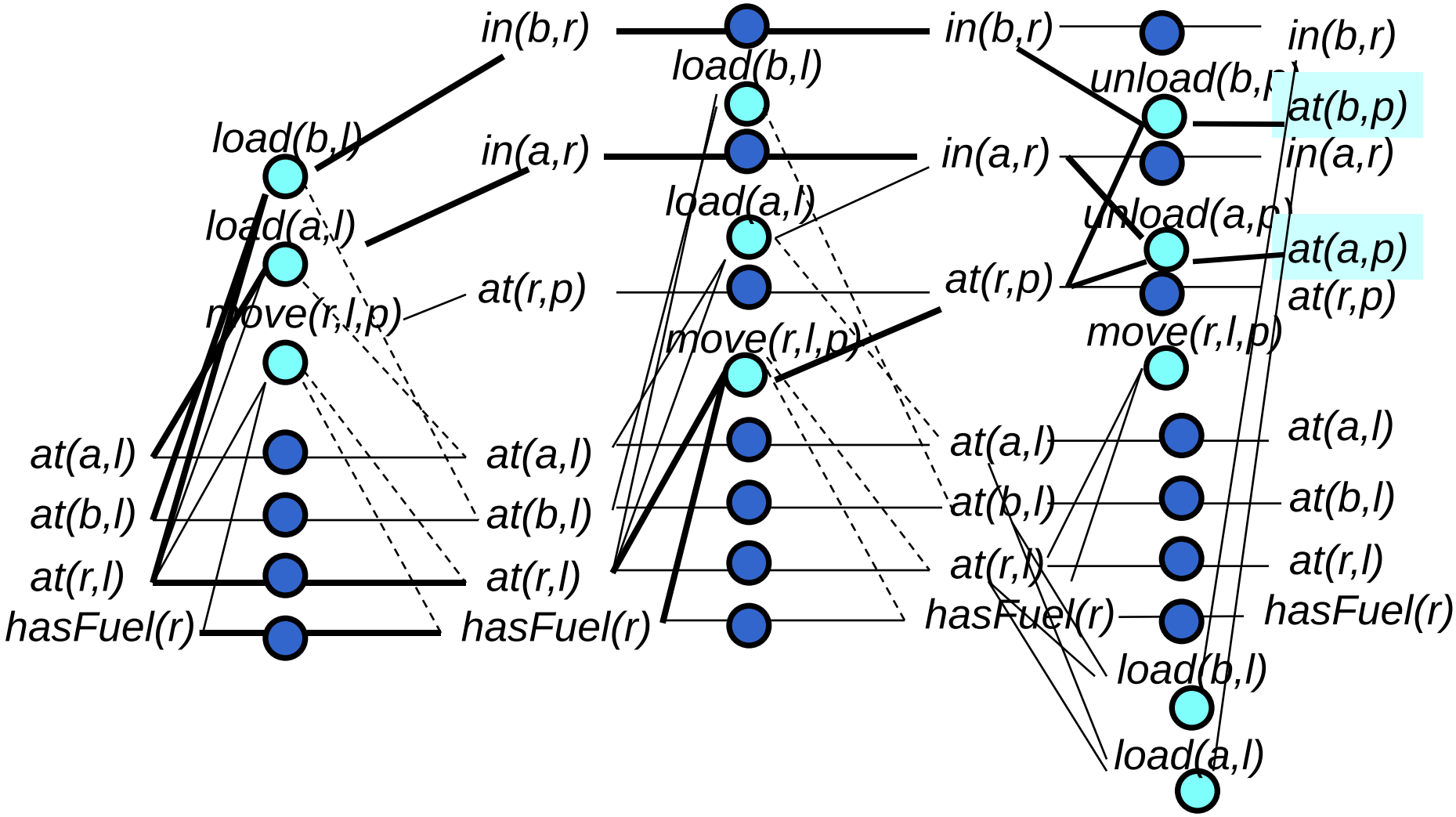
- Azioni allo stesso time step del valid plan possono essere eseguite in qualunque ordine (non interferiscono)
- Proposizioni allo stesso time step del valid plan sono non mutuamente esclusive
- L'ultimo time step contiene tutti i letterali del goal e questi non sono marcati come mutuamente esclusivi

TEOREMI

1. Se esiste un piano valido allora questo è un sottografo del planning graph.
2. In un planning graph due azioni sono mutuamente esclusive in un time step se non esiste un valid plan che le contiene entrambe.
3. In un il planning graph due proposizioni sono mutualmente esclusive in un time step se sono inconsistenti ossia una nega carsi dell'altra.

Conseguenza importante:
Le inconsistenza trovate dall'algoritmo permettono di eliminare strade nell'albero di ricerca

VALID PLAN


in(b,r)
load(b,l)
in(a,r)
load(a,l)
at(r,p)
move(r,l,p)
at(a,l)
at(b,l)
at(r,l)
hasFuel(r)
in(b,r)
load(b,l)
in(a,r)
load(a,l)
at(r,p)
move(r,l,p)
at(a,l)
at(b,l)
at(r,l)
hasFuel(r)
in(b,r)
unload(b,p)
in(a,r)
unload(a,p)
at(r,p)
move(r,l,p)
at(a,l)
at(b,l)
at(r,l)
hasFuel(r)
load(b,l)
load(a,l)
in(b,r)
at(b,p)
in(a,r)
at(a,p)
at(r,p)
at(a,l)
at(b,l)
at(r,l)
hasFuel(r)

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 Sol ≠ fail: return Sol
               else if LEVEL_OFF(grafo): return fail
            grafo = ESPANDI_GRAFO(grafo, problema)
```

- Il primo nodo contiene il planning graph iniziale. Questo contiene solo un time step (proposition level) con le proposizioni vere nello stato iniziale.
  Il grafo iniziale è estratto da `GRAFO_INIZIALE(problema)`

ALGORITMO

- Il goal da raggiungere è estratto dalla funzione `GOAL(problema)`
- Se gli obiettivi sono non mutuamente esclusivi nell'ultimo livello il planning graph potrebbe contenere un piano, ossia un valid plan. Il valid plan è estratto tramite ricerca BACKWARD da `ESTRAI_SOLUZIONE(grafo, obiettivi)` che fornisce una soluzione o un fallimento
  - Si procede livello dopo livello per meglio sfruttare i vincoli di mutua esclusione
  - Metodo ricorsivo: dato un insieme di goals a tempo t si cerca un insieme di azioni a tempo t-1 che abbiano tali goals come add effects. **Le azioni devono essere non mutuamente esclusive.**
  - La ricerca è ad albero, ibrida breadth/depth first e completa

ALGORITMO

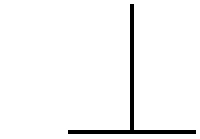
- **Memoization** (non è un errore tipografico!)
  Se ad un certo step della ricerca si determina che un sottoinsieme di goals non è soddisfacibile, graphplan salva questo risultato in una hash table. Ogni volta che lo stesso sottoinsieme di goals verrà selezionato in futuro quel ramo di ricerca fallirà automaticamento

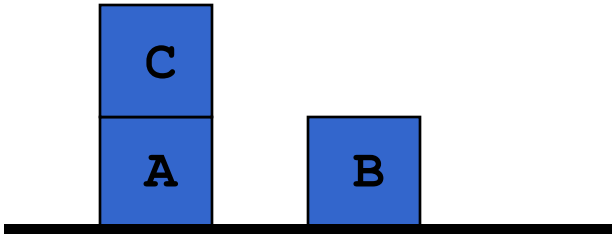
PRIMO ESEMPIO: Mondo a Blocchi

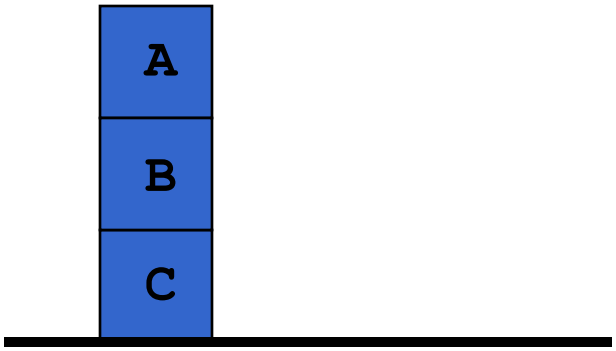
```
(blockA OBJECT)
(blockB OBJECT)
(blockC OBJECT)

(preconds
     (on-table blockA)
     (on-table blockB)
     (on blockC blockA)
     (clear blockB)
     (clear blockC)
     (arm-empty))

(effects
     (on blockA blockB)
     (on blockB blockC))
```







Inserirli nel file block_facts

PRIMO ESEMPIO: Mondo a Blocchi

Inserirli nel file block_ops

```
(operator
  PICK-UP
  (params  (<ob1> OBJECT))
  (preconds
    (clear <ob1>)  (on-table <ob1>)  (arm-empty))
  (effects
    (holding <ob1>)))

(operator
  PUT-DOWN
  (params  (<ob> OBJECT))
  (preconds
    (holding <ob>))
  (effects
     (clear <ob>)  (arm-empty)  (on-table <ob>)))
```

PRIMO ESEMPIO: Mondo a Blocchi

```
(operator
 STACK
 (params (<ob> OBJECT) (<underob> OBJECT))
 (preconds (clear <underob>) (holding <ob>))
 (effects   (arm-empty) (clear <ob>)
            (on <ob> <underob>)))

(operator
 UNSTACK
 (params (<ob> OBJECT) (<underob> OBJECT))
 (preconds (on <ob> <underob>) (clear <ob>)
           (arm-empty))
 (effects (holding <ob>) (clear <underob>)))
```

FAST FORWARD

Fast Forward

FF è un pianificatore euristico estremamente efficiente introdotto da Hoffmann nel 2000

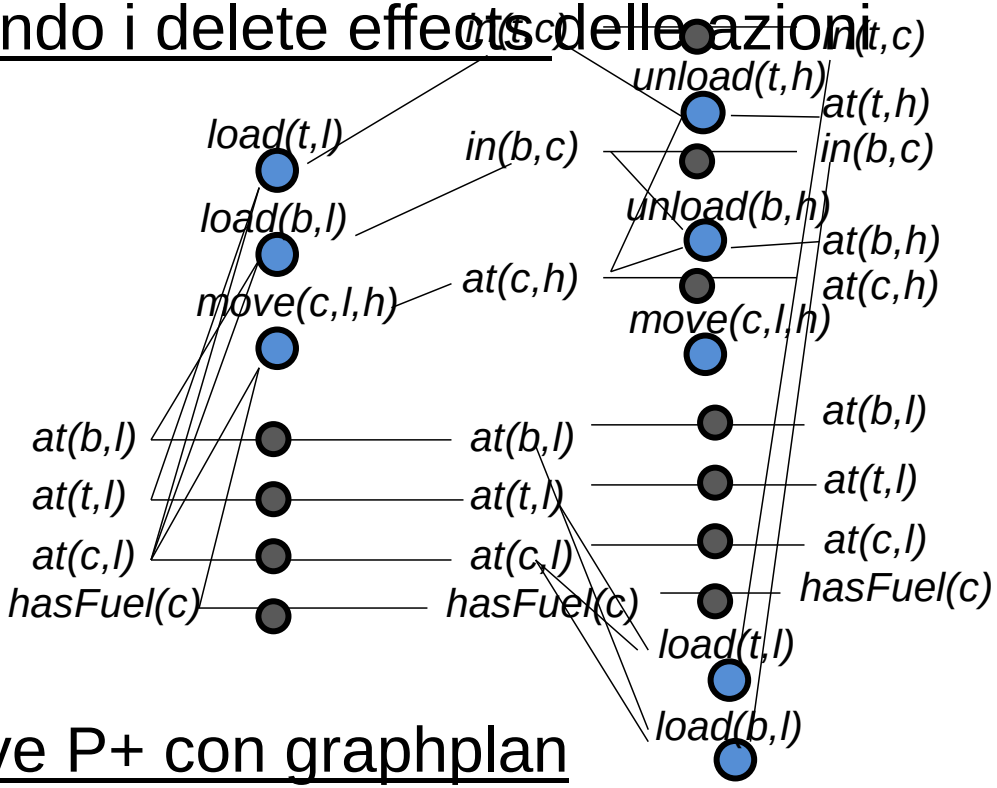
- Euristico = ad ogni stato S è una valutazione della distanza dal goal mediante una funzione euristica

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

FUNZIONE EURISTICA

- Dato un problema P, uno stato S ed un goal G, FF considera il problema rilassato P+ che si ottiene da P trascurando i delete effects delle azioni



- FF risolve P+ con graphplan
- Il numero di azioni nel piano risultante è utilizzato come euristica

FUNZIONE EURISTICA

FF utilizza in realtà un cosiddetto “enforced hill climbing”

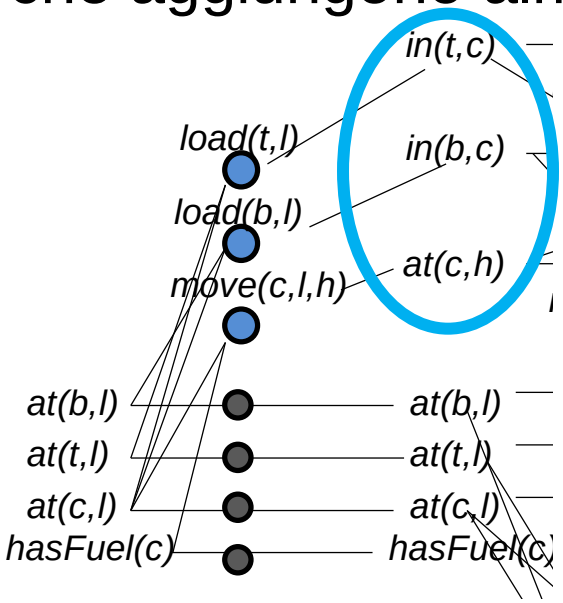
```
function FF(problema): return soluzione or fails
         S = STATO_INIZIALE(problema)
         k = 1
         loop do:
            explore all states S’ at k steps
            if a better state S* is found: S = S*
            else if k can be increased: k = k+1
            else perform complete A* search
```

- In pratica è una ricerca completa breadth first
- Una soluzione viene sempre trovata, a meno che lo stato corrente non sia un vicolo cieco

HELPFUL ACTIONS

Per muoversi da uno stato S ad uno adiancente non si considerano tutte le azioni, ma solo le cosiddette **azioni utili (helpful actions)**

- Sia G1 l'insieme delle proposizioni al time step 1 della soluzione del problema rilassato P+: $H(S) = \{pre(o) \subseteq S, add(o) \cap G1 \neq \emptyset\}$
- Ossia H(S) contiene le azioni applicabili nello stato corrente (S), che aggiungono almeno una delle proposizioni in G1




```
H(S0) = {load(t,l),
         load(b,l),
         move(c,l,h)}
```

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).
- 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
- Blackbox PER PROGETTI
  - 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.