Pianificazione (Planning)

Capitolo 10, Russell & Norvig

Questa presentazione include lucidi creati da: B.J. Dorr, L. Getoor, A. Lazaric, Russel & Norvig, M. Simi, S. Sceffer

Problema di Planning

- Trovare una **sequenza di azioni (piano)** che raggiunge un dato **goal** quando eseguita a partire da un dato **stato iniziale del mondo**. Cioè dati
  - un insieme di descrizioni di operatori (azioni primitive dell'agente),
  - una descrizione dello stato iniziale, e
  - una descrizione dello stato goal,

  calcolare un piano, che è

  - una sequenza di istanze di operatori, tale che eseguita a partire dallo stato iniziale cambia il mondo in modo da portarlo in uno stato che soddisfa la descrizione dello stato goal.

- I goal sono usualmente specificati come una congiunzione di (sotto)goal da raggiungere

Come “produrre” un Piano

- Generative Planning
  - utilizza **principi primi** (conoscenza delle azioni) per generare un piano
  - richiede **modelli formali delle azioni**

- Case-Based Planning
  - **recupera** un piano già prodotto per una situazione simile
  - **revisiona** il piano recuperato per adattarlo al problema in oggetto

- Reinforcement Learning
  - esegue azioni a caso, registrando gli effetti
  - apprende ricompense, modelli di azioni, politiche

Assunzioni Tipiche

- Tempo atomico: ogni azione è indivisibile
- Azioni concorrenti non sono ammesse (anche se le azioni non hanno bisogno di essere ordinate fra loro nel piano)
- Azioni deterministiche: il risultato delle azioni è completamente determinato, non c'è incertezza nel loro effetto
- L'agente è la sola causa di cambiamento del mondo
- L'agente è omniscente: ha conoscenza completa dello stato del mondo
- Closed World Assumption: tutto quello che si sa vero è incluso nella descrizione dello stato. Ciò che non è descritto è falso

Planning vs. problem solving

- Planning e problem solving possono spesso risolvere lo stesso tipo di problemi
- Planning è più potente per le rappresentazioni e i metodi usati
- Stati, goal, e azioni sono decomposte in insiemi di sentenze (usualmente in FOL)
- La ricerca spesso procede attraverso lo **spazio dei piani** invece dello spazio degli stati (anche se esistono pianificatori basati sugli stati)
- Subgoal possono essere pianificati indipendentemente, riducendo la complessità del problema di pianificazione

Goal del Planning

- Supponiamo che il goal sia HAVE(MILK).
- Da qualche stato iniziale dove HAVE(MILK) non è soddisfatto, la funzione successore deve essere applicata ripetutamente per generare eventualmente uno stato dove HAVE(MILK) è soddisfatto.
- Una rappresentazione esplicita delle azioni possibili e i loro effetti aiuterebbe il problem solver a selezionare le azioni rilevanti

deve … r
cono … ello
stato e quale è l'effetto di ognuna

Altrimenti, nel mondo reale un agente sarebbe sopraffatto da azioni irrilevanti

Goal del Planning

Supponiamo che il goal sia
HAVE(MILK) ∧ HAVE(BOOK)

Senza una rappresentazione esplicita del goal, il problem solver non può sapere che uno stato dove HAVE(MILK) è già raggiunto è più promettente di uno stato dove né HAVE(MILK) né HAVE(BOOK) è raggiunto

gli stati sono strutture dati specializzate sul dominio, e le euristiche devono essere fornite per ogni nuovo problema

Goal del Planning

- Scegliere le azioni per raggiungere un o…

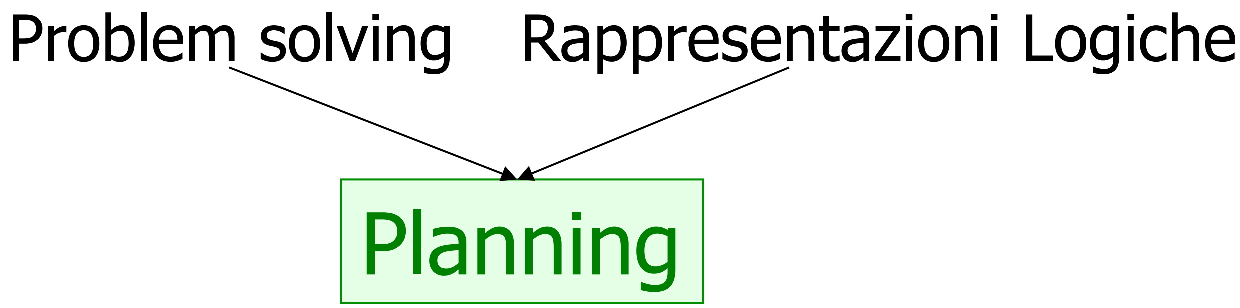
HAVE(MILK) e HAVE(BOOK) possono essere raggiunti da due sequenze di azioni quasi indipendenti

- M… problem solving?
- Alcune difficoltà con il problem solving:
  - Il goal può consistere di tanti sottogoal indipendenti, ma non c'è modo che il problem solver lo sappia

Planning: rappresentazioni

Il planning apre le black-box usando la logica per rappresentare:

- Azioni
- Stati
- Goal



Approcci Principali

- **Calcolo delle situazioni**
- **Planning nello spazio degli stati**
- **Partial order planning**
- **Grafi di Planning**
- **Decomposizione Gerarchica (HTN planning)**
- **Planning Reattivo (Reactive planning)**

Planning con Calcolo delle Situazioni

- Idea base: rappresentare il problema di planning in FOL
  - Il calcolo delle situazioni ci permette di ragionare sui cambiamenti del mondo
  - Usa inferenza (theorem proving) per "provare" che una particolare sequenza di azioni, quando applicata alla situazione che caratterizza lo stato del mondo, condurrà al risultato desiderato (piano = prova)

Calcolo delle Situazioni

- **Stato Iniziale**: una sentenza logica su (situazione) S_0

  $At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \neg Have(Bananas, S_0) \wedge \neg Have(Drill, S_0)$

- **Stato Goal**:

  $(\exists s)\ At(Home,s) \wedge Have(Milk,s) \wedge Have(Bananas,s) \wedge Have(Drill,s)$

- **Operatori** sono descrizioni di come il mondo cambia a causa delle azioni dell' agente:

  $\forall(a,s)\ Have(Milk,Result(a,s)) \Leftrightarrow ((a=Buy(Milk) \wedge At(Grocery,s)) \vee (Have(Milk, s) \wedge a \neq Drop(Milk)))$

- Result(a,s) già visto (la situazione risultante dalla esecuzione della azione a nella situazione s).
- Sequenze di azioni: Result'(l,s) è il risultato della esecuzione della lista di azioni (l) a partire da s:

  $(\forall s)\ Result'([\ ],s) = s$

  $(\forall a,p,s)\ Result'([a|p],s) = Result'(p,Result(a,s))$

Calcolo delle Situazioni

- Una soluzione è un piano che quando applicato allo stato iniziale conduce ad una situazione che soddisfa la query di goal:

  $At(Home,Result'(p,S_0))$
  $\wedge$ $Have(Milk,Result'(p,S_0))$
  $\wedge$ $Have(Bananas,Result'(p,S_0))$
  $\wedge$ $Have(Drill,Result'(p,S_0))$

- Quindi ci si aspetta un piano (cioè un assegnamento di variabile tramite unificazione) tale che:

  p = [Go(Grocery), Buy(Milk), Buy(Bananas), Go(HardwareStore), Buy(Drill), Go(Home)]

Calcolo delle Situazioni: Analisi

- In teoria va bene, ma il problem solving (ricerca) è esponenziale nel caso pessimo
- Inoltre, la risoluzione trova ***una*** prova (=piano), non necessariamente un buon piano !
- Ricordiamoci anche del Problema del Frame, della Qualifica e della Ramificazione …
- Quindi è meglio usare un **linguaggio ristretto** e un **algoritmo specializzato (planner)** piuttosto che un dimostratore generale di teoremi

Rappresentazioni base per il planning

- Approccio classico usato negli anni 70: **STRIPS**
- Stati rappresentati come una congiunzione di letterali ground
  - at(Home) ∧ ¬ have(Milk) ∧ ¬ have(bananas) ...
- I goal sono congiunzioni di letterali, ma possono avere variabili che sono assunte essere **quantificate esistenzialmente**
  - at(?x) ∧ have(Milk) ∧ have(bananas) ...
- Non c'è bisogno di specificare completamente lo stato
  - Non-specificato significa non rilevante o assunto falso
  - Rappresenta molti casi in poca memoria
  - Spesso rappresenta solo i cambiamenti nello stato piuttosto che l'intera situazione
- Al contrario di un dimostratore di teoremi, non cerca se il goal è vero, ma se c'è una sequenza di azioni che lo raggiunge

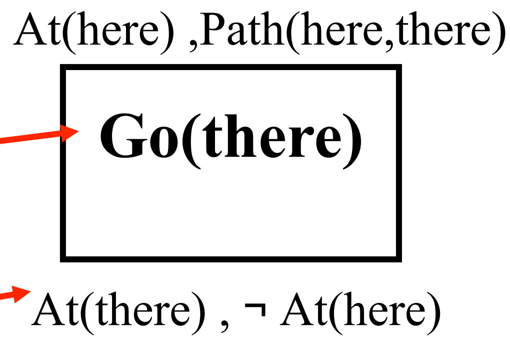
Rappresentazione Operatori/ azioni

- Gli operatori contengono tre componenti:
  - **Descrizione delle azioni**
  - **Precondizioni** – congiunzione di letterali positivi
  - **Effetto** – congiunzione di letterali positivi o negativi che descrivono come la situazione cambia quando si applica l'operatore
- Esempio:

  Op[Action: Go(there),

  Precondizioni: At(here) $\wedge$ Path(here,there),

  Effetto: At(there) $\wedge$ $\neg$ At(here)]



- Tutte le variabili sono quantificate universalmente
- Le variabili di situazione sono implicite
  - le precondizioni devono essere vere nello stato precedente all'applicazione dell'operatore; gli effetti sono veri immediatamente dopo

Mondo dei blocchi

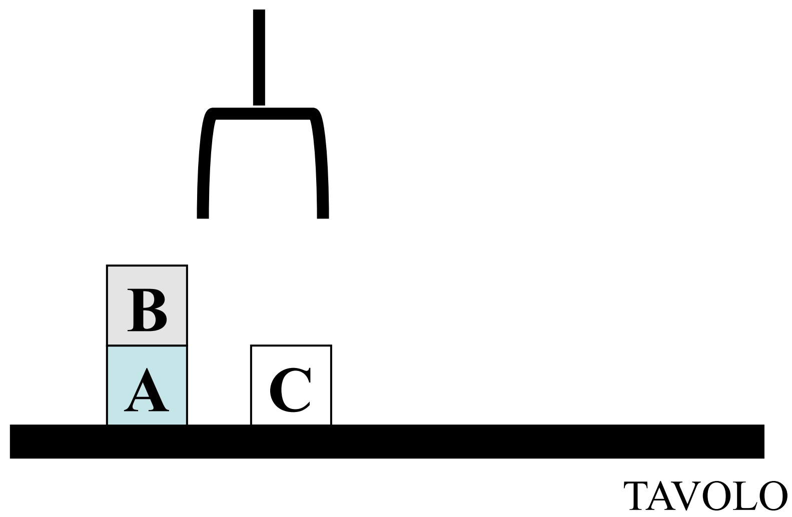
Il mondo dei blocchi è un micro-mondo che consiste di un tavolo, un insieme di blocchi e un manipolatore robotico

Alcuni vincoli del dominio:

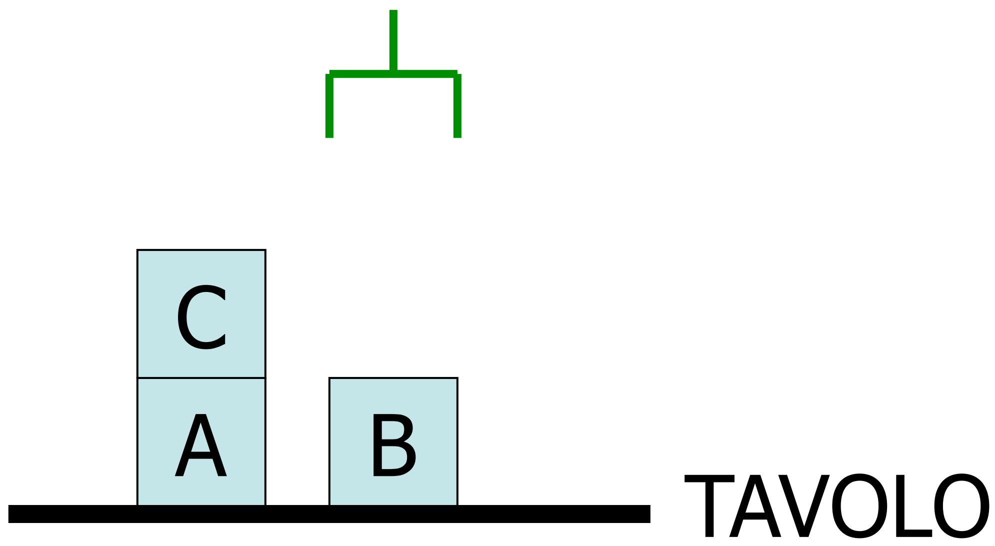
- Un solo blocco può essere immediatamente sopra un altro
- Un qualsiasi numero di blocchi sul tavolo
- Il manipolatore può mantenere un solo blocco

Rappresentazione tipica:

on(a,tavolo)
on(c,tavolo)
on(b,a)
handempty
clear(b)
clear(c)

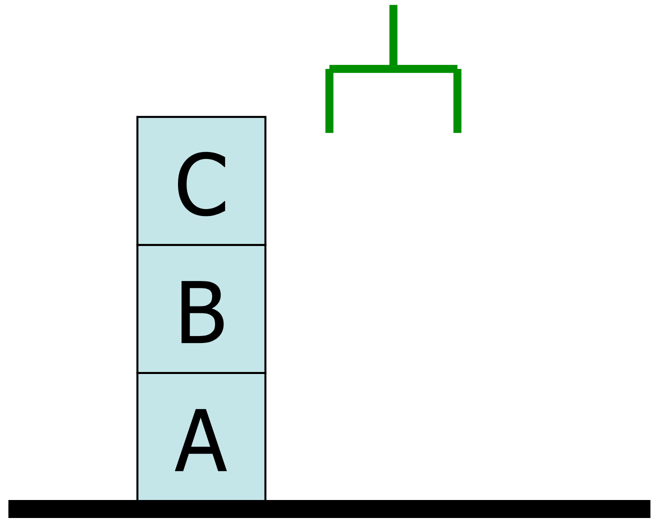


Rappresentazione dello Stato



Congiunzione di proposizioni:
BLOCK(A), BLOCK(B), BLOCK(C),
ON(A,TAVOLO), ON(B,TAVOLO), ON(C,A),
CLEAR(B), CLEAR(C), HANDEMPTY

Rappresentazione del Goal



Congiunzione di proposizioni:
ON(A,TAVOLO), ON(B,A), ON(C,B)

Il goal G è raggiunto in uno stato S se tutte le proposizioni in G sono anche in S

Rappresentazione delle Azioni

Unstack(x,y)

- P = HANDEMPTY, BLOCK(x), BLOCK(y),
  CLEAR(x), ON(x,y)
- E = ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x),
  ¬ ON(x,y), CLEAR(y)

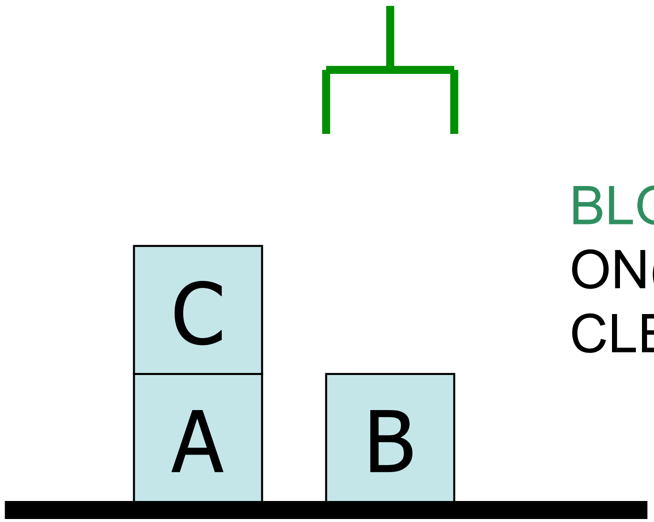
Effetto: lista di letterali

Precondizioni: congiunzione di proposizioni

Significa: Rimuove HANDEMPTY dallo stato

Significa: Aggiungi HOLDING(x) allo stato

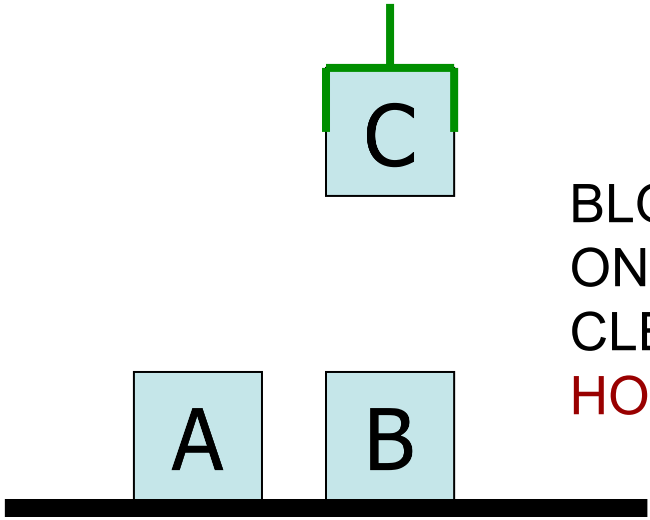
Esempio



BLOCK(A), BLOCK(B), BLOCK(C),
ON(A,TAVOLO), ON(B,TAVOLO), ON(C,A),
CLEAR(B), CLEAR(C), HANDEMPTY

Unstack(C,A)
- P = HANDEMPTY, BLOCK(C), BLOCK(A), CLEAR(C), ON(C,A)
- E = ¬HANDEMPTY, ¬CLEAR(C), HOLDING(C), ¬ ON(C,A), CLEAR(A)

Esempio



BLOCK(A), BLOCK(B), BLOCK(C),
ON(A,TAVOLO), ON(B,TAVOLO), ~~ON(C,A)~~,
CLEAR(B), ~~CLEAR(C)~~, ~~HANDEMPTY~~,
HOLDING(C), CLEAR(A)

Unstack(C,A)

- P = HANDEMPTY, BLOCK(C), BLOCK(A),
  CLEAR(C), ON(C,A)
- E = ¬HANDEMPTY, ¬CLEAR(C), HOLDING(C),
  ¬ ON(C,A), CLEAR(A)

Rappresentazione delle Azioni

Unstack(x,y)

- P = HANDEMPTY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- E = ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ ON(x,y), CLEAR(y)

Stack(x,y)

- P = HOLDING(x), BLOCK(x), BLOCK(y), CLEAR(y)
- E = ON(x,y), ¬CLEAR(y), ¬HOLDING(x), CLEAR(x), HANDEMPTY

Pickup(x)

- P = HANDEMPTY, BLOCK(x), CLEAR(x), ON(x,TAVOLO)
- E = ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ON(x,TAVOLO)

PutDown(x)

- P = HOLDING(x)
- E = ON(x,TAVOLO), ¬HOLDING(x), CLEAR(x), HANDEMPTY

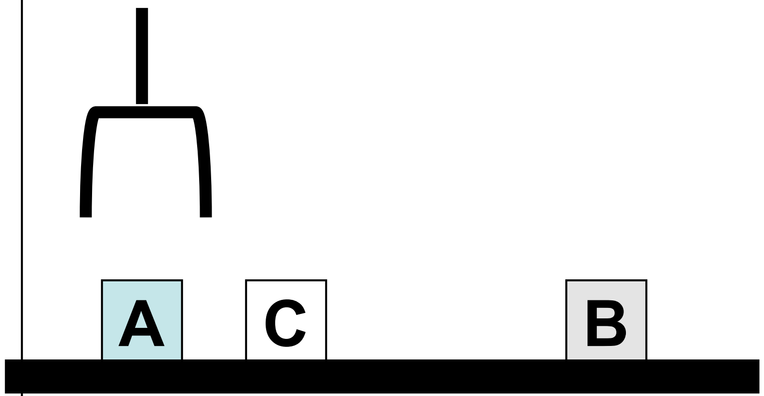
Problema di planning

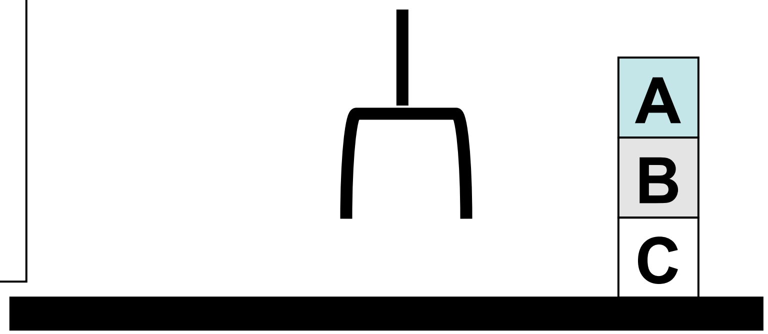
Stato iniziale:

- clear(a)
- clear(b)
- clear(c)
- on(a,tavolo)
- on(b,tavolo)
- on(c,tavolo)
- handempty

Goal:

- on(b,c)
- on(a,b)
- on(c,tavolo)





Un piano:

- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

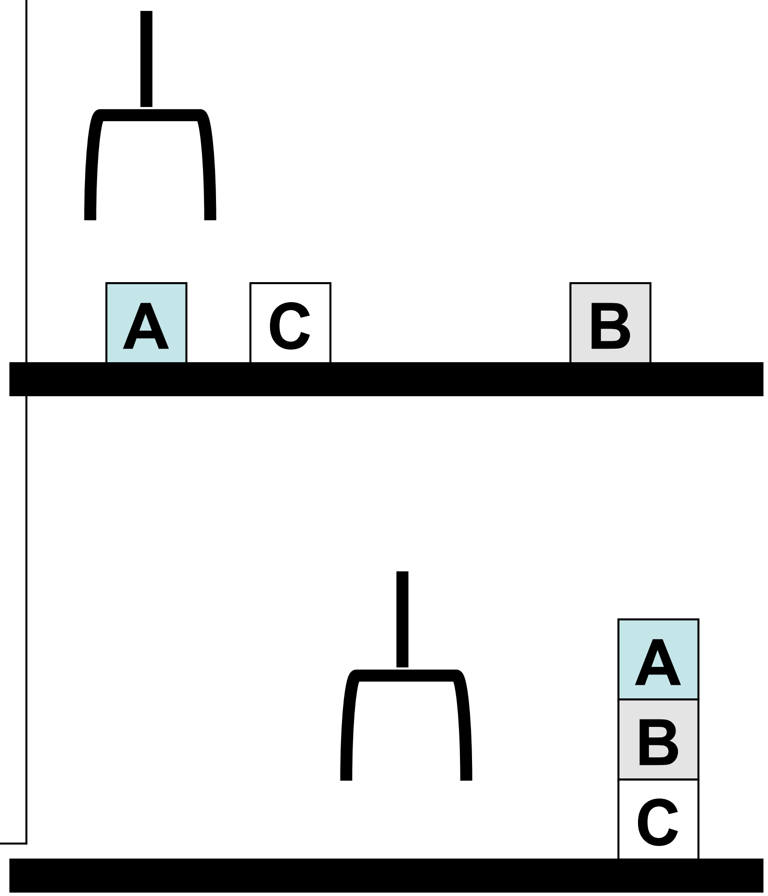
Problema di planning

Stato iniziale:

- clear(a)
- clear(b)
- clear(c)
- on(a,tavolo)
- on(b,tavolo)
- on(c,tavolo)
- handempty

Goal:

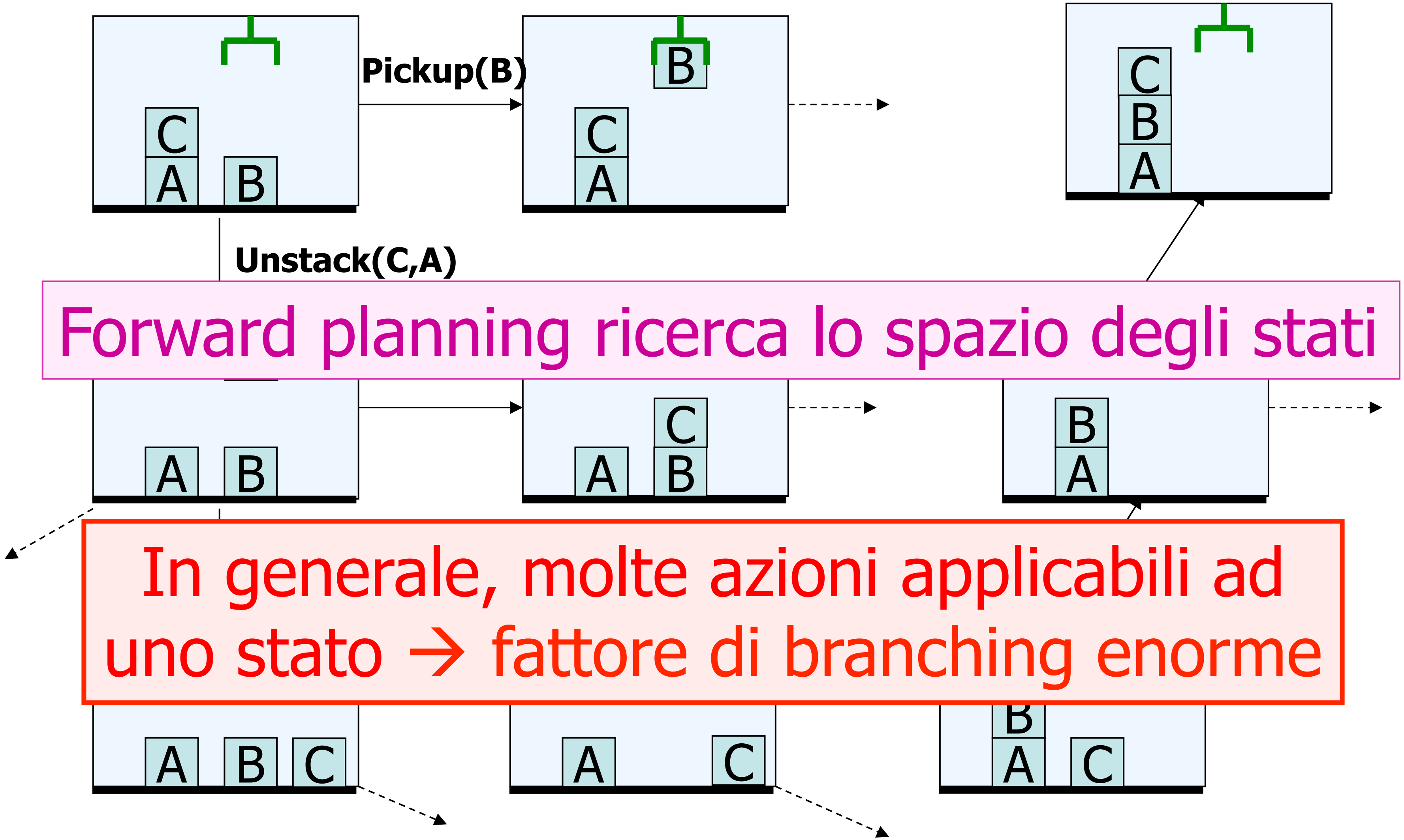
- on(b,c)
- on(a,b)
- on(c,tavolo)



Un piano:

- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Forward Planning



Forward planning ricerca lo spazio degli stati

In generale, molte azioni applicabili ad uno stato → fattore di branching enorme

Azioni Rilevanti

- Una azione è rilevante per un goal se uno dei suoi effetti combacia con una proposizione del goal
- Stack(B,A)
  - P = HOLDING(B), BLOCK(A), CLEAR(A)
  - E = ON(B,A), ¬CLEAR(A), ¬HOLDING(B), CLEAR(B), HANDEMPTY

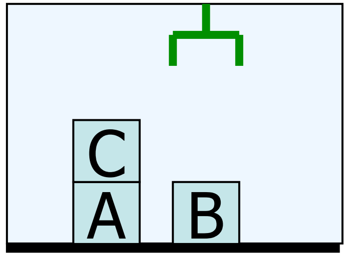
  è rilevante per ON(B,A), ON(C,B)

Backward Chaining

ON(B,A), ON(C,B)

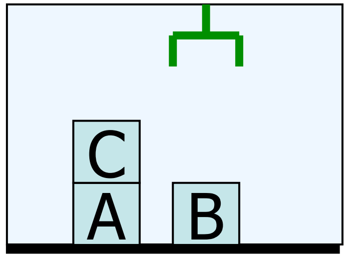
Stack(C,B)

ON(B,A), HOLDING(C), CLEAR(B)



In generale, ci sono molte meno azioni rilevanti per un goal che azioni applicabili → fattore di branching più piccolo che con il forward planning

Backward Chaining



ON(B,A), ON(C,B)

Stack(C,B)

ON(B,A), HOLDING(C), CLEAR(B)

Pickup(C)

ON(B,A), CLEAR(B), HANDEMPTY, CLEAR(C), ON(C,TAVOLO)

Backward planning ricerca lo spazio dei goal

CLEAR(C), ON(C,TAVOLO), HOLDING(B), CLEAR(A)

Pickup(B)

CLEAR(C), ON(C,TAVOLO), CLEAR(A), HANDEMPTY, CLEAR(B), ON(B,TAVOLO)

Putdown(C)

CLEAR(A), CLEAR(B), ON(B,TAVOLO), HOLDING(C)

Unstack(C,A)

CLEAR(B), ON(B,TAVOLO), CLEAR(C), HANDEMPTY, ON(C,A)

Planning in STRIPS

- STRIPS mantiene due strutture dati:
  - **Lista di Stati** – tutti i predicati correntemente veri.
  - **Pila di Goal** – una pila di goal da risolvere, con il goal corrente in testa alla pila.
- Se il goal corrente non è soddisfatto dallo stato presente, esamina gli effetti positivi degli operatori, e inserisce l'operatore e la lista delle precondizioni sulla pila. (Subgoal)
- Quando il goal corrente è soddisfatto, lo rimuove dalla pila.
- Quando un operatore è in testa alla pila, registra l'applicazione dell'operatore sulla sequenza del piano e usa gli effetti per aggiornare lo stato corrente.

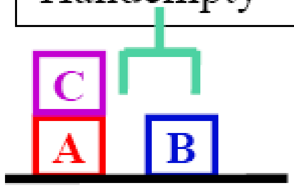
STRIPS Algorithm

- STRIPS (*initial-state, goals*)
  - *state* = *initial-state*; *plan* = []; *stack* = []
  - Push *goals* on *stack*
  - Repeat until *stack* is empty
    - If top of *stack* is **goal** that matches *state*, then pop *stack*
    - Else if top of stack is a **conjunctive goal** *g*, then
      - **Select** an ordering for the subgoals of *g*, and push them on *stack*
    - Else if top of *stack* is a **simple goal** *sg*, then
      - **Choose** an operator *o* whose add-list matches goal *sg*
      - Replace goal *sg* with operator *o*
      - Push the preconditions of *o* on the *stack*
    - Else if top of *stack* is an **operator** *o*, then
      - *state* = apply(*o*, *state*)
      - *plan* = [*plan*; *o*]

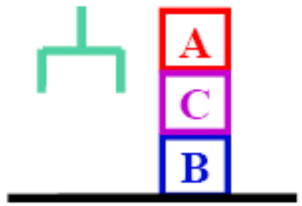
fonte: R. Simmons

Applicazione di STRIPS

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

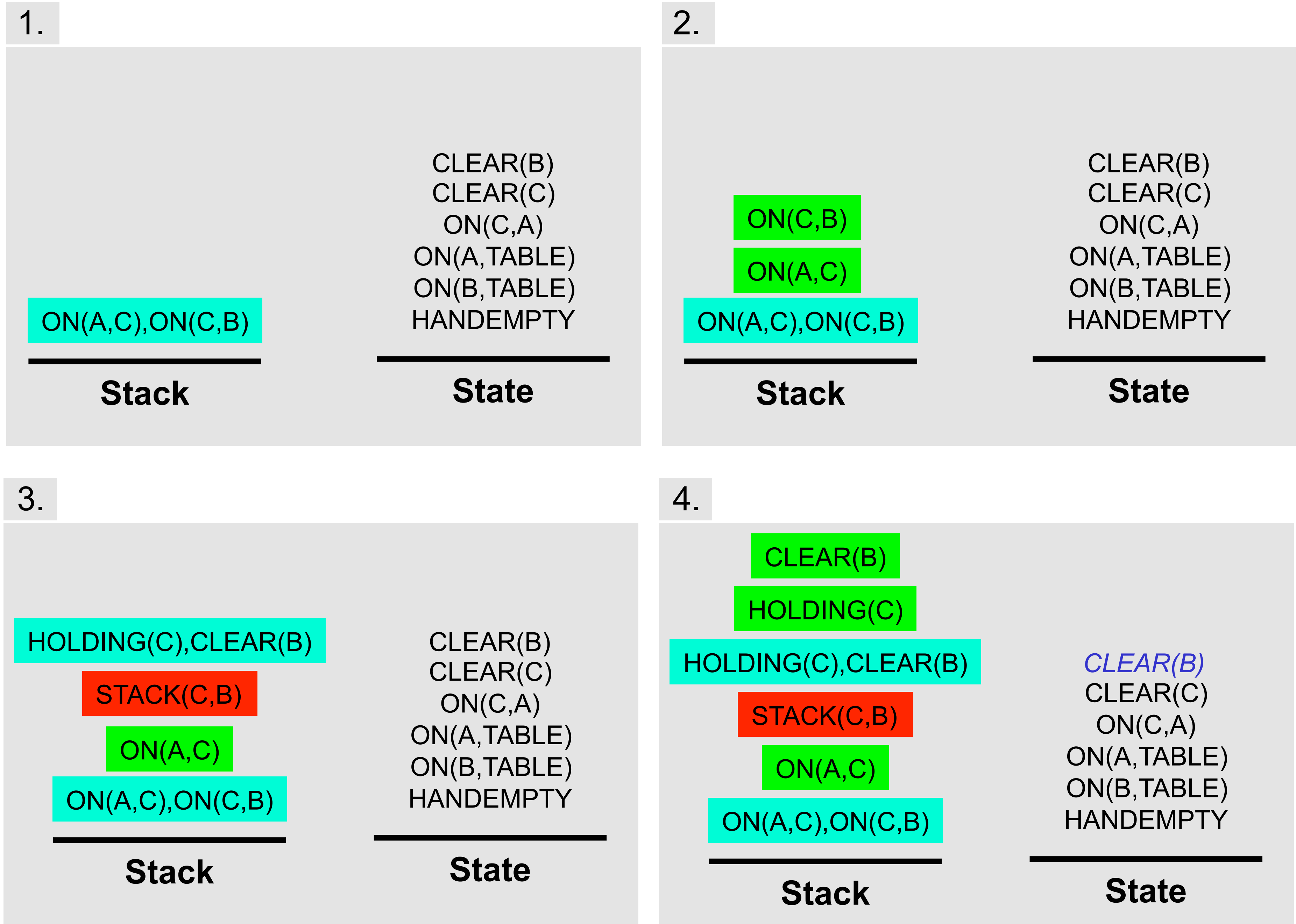


Initial State



Goal

fonte: R. Simmons


1.
ON(A,C),ON(C,B)
Stack
CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY
State
2.
ON(C,B)
ON(A,C)
ON(A,C),ON(C,B)
Stack
CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY
State
3.
HOLDING(C),CLEAR(B)
STACK(C,B)
ON(A,C)
ON(A,C),ON(C,B)
Stack
CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY
State
4.
CLEAR(B)
HOLDING(C)
HOLDING(C),CLEAR(B)
STACK(C,B)
ON(A,C)
ON(A,C),ON(C,B)
Stack
CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY
State

5.

HOLDING(C)
HOLDING(C),CLEAR(B)
STACK(C,B)
ON(A,C)
ON(A,C),ON(C,B)

Stack

CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY

State

6.

HANDEMPTY,CLEAR(C),ON(C,A)
UNSTACK(C,A)
HOLDING(C),CLEAR(B)
STACK(C,B)
ON(A,C)
ON(A,C),ON(C,B)

Stack

CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY

State

7.

UNSTACK(C,A)
HOLDING(C),CLEAR(B)
STACK(C,B)
ON(A,C)
ON(A,C),ON(C,B)

Stack

CLEAR(B)
CLEAR(C)
ON(C,A)
ON(A,TABLE)
ON(B,TABLE)
HANDEMPTY

State

8. [UNSTACK(C,A)]

HOLDING(C),CLEAR(B)
STACK(C,B)
ON(A,C)
ON(A,C),ON(C,B)

Stack

CLEAR(A)
HOLDING(C)
CLEAR(B)
ON(A,TABLE)
ON(B,TABLE)

State

9. [UNSTACK(C,A)]

Stack

- STACK(C,B)
- ON(A,C)
- ON(A,C),ON(C,B)

State

- CLEAR(A)
- *HOLDING(C)*
- *CLEAR(B)*
- ON(A,TABLE)
- ON(B,TABLE)

10. [UNSTACK(C,A); STACK(C,B)]

Stack

- ON(A,C)
- ON(A,C),ON(C,B)

State

- *HANDEMPTY*
- *CLEAR(C)*
- *ON(C,B)*
- CLEAR(A)
- ON(A,TABLE)
- ON(B,TABLE)

11. [UNSTACK(C,A); STACK(C,B)]

Stack

- HOLDING(A),CLEAR(C)
- STACK(A,C)
- ON(A,C),ON(C,B)

State

- HANDEMPTY
- CLEAR(C)
- ON(C,B)
- CLEAR(A)
- ON(A,TABLE)
- ON(B,TABLE)

12. [UNSTACK(C,A); STACK(C,B)]

Stack

- CLEAR(C)
- HOLDING(A)
- HOLDING(A),CLEAR(C)
- STACK(A,C)
- ON(A,C),ON(C,B)

State

- HANDEMPTY
- *CLEAR(C)*
- ON(C,B)
- CLEAR(A)
- ON(A,TABLE)
- ON(B,TABLE)

13. [UNSTACK(C,A); STACK(C,B)]

Stack	State
HOLDING(A)	HANDEMPTY
HOLDING(A),CLEAR(C)	CLEAR(C)
STACK(A,C)	ON(C,B)
ON(A,C)	CLEAR(A)
ON(A,C),ON(C,B)	ON(A,TABLE)
	ON(B,TABLE)

14. [UNSTACK(C,A); STACK(C,B)]

Stack	State
HANDEMPTY,CLEAR(A), ON(A,TABLE)	*HANDEMPTY*
PICKUP(A)	CLEAR(C)
HOLDING(A),CLEAR(C)	ON(C,B)
STACK(A,C)	*CLEAR(A)*
ON(A,C)	*ON(A,TABLE)*
ON(A,C),ON(C,B)	ON(B,TABLE)

15. [UNSTACK(C,A); STACK(C,B)]

Stack

PICKUP(A)
HOLDING(A),CLEAR(C)
STACK(A,C)
ON(A,C),ON(C,B)

State

HANDEMPTY
CLEAR(C)
ON(C,B)
CLEAR(A)
ON(A,TABLE)
ON(B,TABLE)

16. [UNSTACK(C,A); STACK(C,B); PICKUP(A)]

Stack

HOLDING(A),CLEAR(C)
STACK(A,C)
ON(A,C),ON(C,B)

State

HOLDING(A)
CLEAR(C)
ON(C,B)
ON(B,TABLE)

17. [UNSTACK(C,A); STACK(C,B); PICKUP(A)]

Stack

STACK(A,C)
ON(A,C),ON(C,B)

State

HOLDING(A)
CLEAR(C)
ON(C,B)
ON(B,TABLE)

18. [UNSTACK(C,A); STACK(C,B); PICKUP(A); STACK(A,C)]

Stack

ON(A,C),ON(C,B)

State

ON(A,C)
CLEAR(A)
HANDEMPTY
ON(C,B)
ON(B,TABLE)

Stack

Goal raggiunto!

State

ON(A,C)
CLEAR(A)
HANDEMPTY
ON(C,B)
ON(B,TABLE)

Piani subottimi

- Consideriamo le seguenti azioni:

Op[Action: Load(obj,plane,loc),
    Precondizioni: At(obj,loc) ∧ At(plane,loc),
    Effetto: Inside(obj,plane) ∧ ¬ At(obj,loc)]

Op[Action: Unload(obj,plane,loc),
    Precondizioni: Inside(obj,plane) ∧ At(plane,loc),
    Effetto: At(obj,loc) ∧ ¬ Inside(obj,plane)]

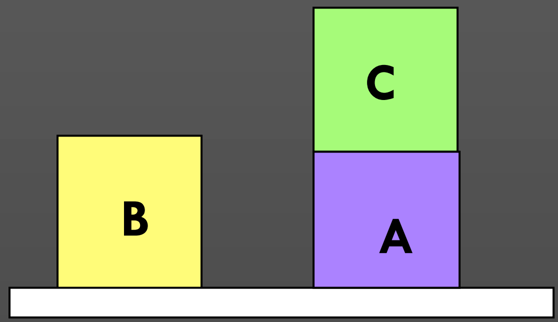
Op[Action: Fly(plane,from,to),
    Precondizioni: At(plane,from),
    Effetto: At(plane,to) ∧ ¬ At(plane,from)]

- Stato iniziale: **At(obj1,locA), At(obj2,locA), At(747,locA)**

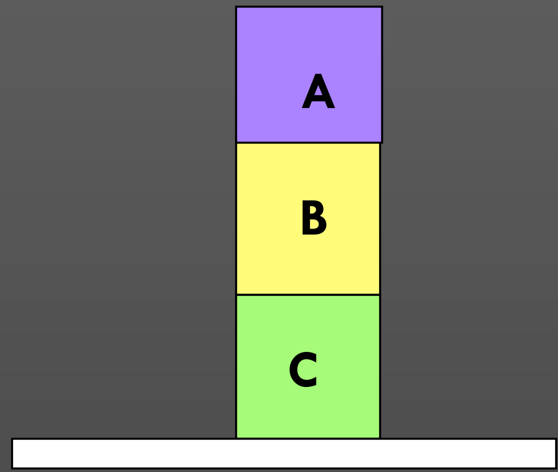
- Goal: **At(obj1,locB), At(obj2,locB)**

- Piano: [**Load(obj1,747,locA); Fly(747,locA,locB); Unload(obj1,747,locB); Fly(747,locB,locA); Load(obj2,747,locA); Fly(747,locA,locB); Unload(obj2,747,locB)**]

Esercizio



Stato iniziale



goal

Interazione fra sottogoal

- Algoritmi di planning semplici assumono che i sottogoal possano essere raggiunti indipendentemente
  - Ogni sottogoal può essere risolto separatamente e poi le relative soluzioni concatenate
- L'esercizio precedente mostra questo aspetto tramite la così detta "Anomalia di Sussman" :
  - La soluzione a on(a,b) (tramite unstack(c,a), stack(a,b)) è disfatta quando si risolve il secondo sottogoal on(b,c) (tramite unstack(a,b), stack(b,c)).
  - Se si risolve per primo on(b,c), la soluzione sarà disfatta da on(a,b)
- STRIPS classico non è in grado di superare questo problema

Problemi non risolvibili

- Dovuti alla linearità e ad azioni irreversibili:

  Op[Action: Load(obj,plane,loc),
  Precondizioni: At(obj,loc) ∧ At(plane,loc),
  Effetto: Inside(obj,plane) ∧ ¬ At(obj,loc)]

  Op[Action: Unload(obj,plane,loc),
  Precondizioni: Inside(obj,plane) ∧ At(plane,loc),
  Effetto: At(obj,loc) ∧ ¬ Inside(obj,plane)]

  Op[Action: Fly(plane,from,to),
  Precondizioni: At(plane,from) ∧ **Have-fuel(plane)**,
  Effetto: At(plane,to) ∧ ¬ At(plane,from)] ∧ ¬ **Have-fuel(plane)**

- Stato iniziale: **At(obj1,locA), At(obj2,locA), At(747,locA), Have-fuel(747)**

- Goal: **At(obj1,locB), At(obj2,locB)**

Problemi non risolvibili

- Tentiamo di risolvere prima il sottogoal **At(obj1,locB)**
  - [**Load(obj1,747,locA); Fly(747,locA,locB); Unload(obj1,747,locB)**]
  - ma non riusciamo a raggiungere **At(obj2,locB)** perché è finito il carburante!
- Tentiamo di risolvere prima il sottogoal **At(obj2,locB)**
  - [**Load(obj2,747,locA); Fly(747,locA,locB); Unload(obj2,747,locB)**]
  - ma non riusciamo a raggiungere **At(obj1,locB)** perché è finito il carburante!

In ogni caso STRIPS non è in grado di risolvere il problema !

Planning nello spazio degli stati: riassunto

- Spazio delle situazioni (localizzazione, possedimenti, etc.)
- Il piano è una soluzione trovata “cercando” tra le situazioni il goal
- Un **planner progressivo** cerca il goal in avanti (forward) a partire dallo stato iniziale
- Un **planner regressivo** cerca all’indietro (backward) a partire dal goal
- Attenzione: problema della Anomalia di Sussman

Planning nello spazio dei piani

- Una alternativa è la **ricerca attraverso lo spazio dei *piani***, piuttosto che delle situazioni.
- Si parte da un **piano parziale** che viene espanso e raffinato fino a raggiungere un piano completo che risolve il problema.
- **Operatori di raffinamento** aggiungono vincoli a piani parziali e operatori di modifica effettuano altri cambiamenti.
- Operatori alla STRIPS:

  Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

  Op(ACTION: RightSock, EFFECT: RightSockOn)

  Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

  Op(ACTION: LeftSock, EFFECT: leftSockOn)

possono risultare in un piano parziale del goal

  [RightShoe, LeftShoe]

Partial-order planning (POP)

- Un **planner lineare** costruisce un piano come una **sequenza totalmente ordinata** di passi
- Un **planner non-lineare (aka partial-order planner)** costruisce un piano come un insieme di passi con alcuni vincoli temporali
- Vincoli della forma S1<S2 se il passo S1 deve venire prima di S2.
- Si **raffina** un piano ordinato parzialmente (POP) per mezzo di:
  - **Aggiunta di un nuovo passo al piano**, o
  - **Aggiunta di un nuovo vincolo** ai passi già presenti nel piano.
- Un POP può essere **linearizzato** (convertito in un piano totalmente ordinato) attraverso un ordinamento topologico

Minimo Impegno

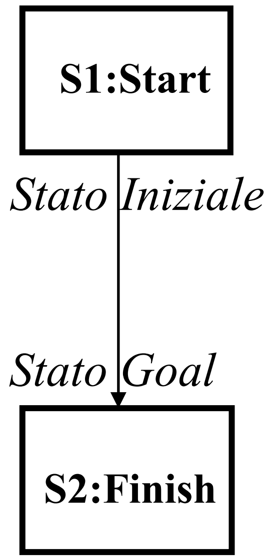
- I planner non-lineari incorporano il principio del **minimo impegno (least commitment)**
  - Si scelgono solo quelle azioni, ordinamenti, e assegnamenti di variabili che sono assolutamente necessari, lasciando le altre decisioni al futuro
  - Evita di prendere decisioni premature su aspetti che non contano
- Un planner lineare sceglie sempre di aggiungere un passo in un punto preciso della sequenza
- Un planner non-lineare sceglie di aggiungere un passo ed eventualmente qualche vincolo temporale fra passi

Piano Non-lineare

- Consiste di
  - (1) Un insieme di **passi** $\{S_1, S_2, S_3, S_4\ldots\}$
    - Ogni passo ha la descrizione di un operatore, precondizioni e post-condizioni
  - (2) Un insieme di **link causali** $\{ \ldots (S_i,C,S_j) \ldots\}$
    - Che significa che uno dei propositi del passo S_i è di raggiungere la precondizione C del passo S_j
  - (3) Un insieme di **vincoli di ordinamento** $\{ \ldots S_i<S_j \ldots \}$
    - Nel caso in cui il passo S_i deve venire prima del passo S_j
- Un piano non-lineare è **completo** sse
  - Ogni passo menzionato in (2) e (3) è in (1)
  - Se S_j ha prerequisito C, allora esiste un link causale in (2) nella forma (S_i,C,S_j) per qualche S_i
  - Se (S_i,C,S_j) è in (2) e il passo S_k è in (1), e S_k "minaccia" (S_i,C,S_j) (rende C falso), allora (3) contiene $S_k<S_i$ o $S_k>S_j$

Il Piano Iniziale

Ogni piano inizia nello stesso modo



Esempio Banale

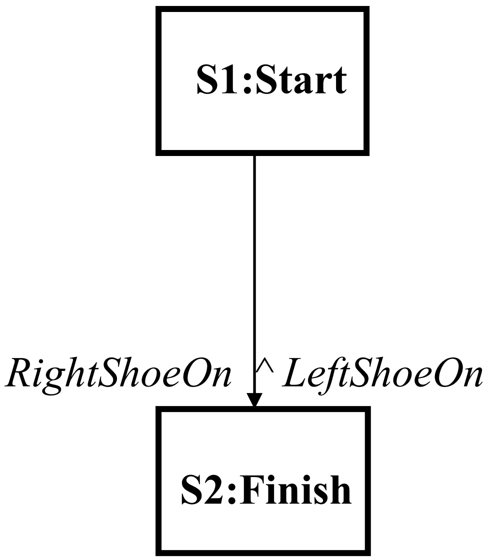
Operatori:

Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

Op(ACTION: RightSock, EFFECT: RightSockOn)

Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

Op(ACTION: LeftSock, EFFECT: leftSockOn)



Passi: {S1:[Op(Action:Start)],

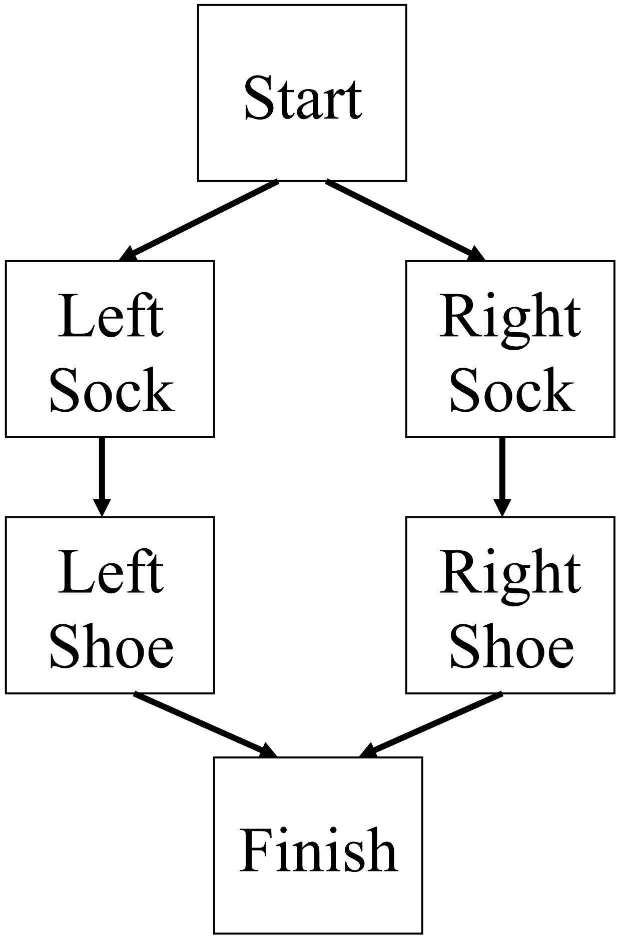
S2:[Op(Action:Finish,

Pre: RightShoeOn^LeftShoeOn)]}

Link: {}

Ordinamenti: {S1<S2}

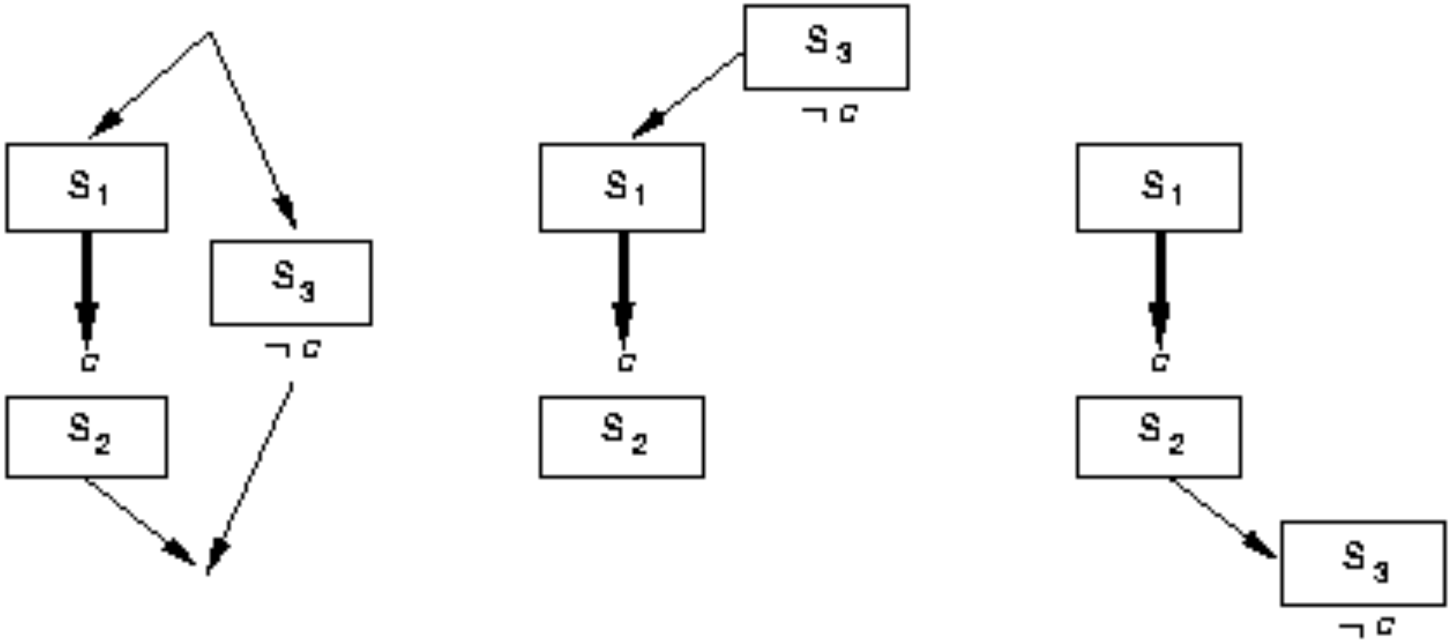
Soluzione



POP: vincoli ed euristiche

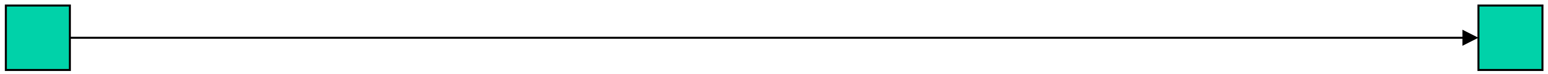
- Aggiungere solo passi che raggiungono una precondizione correntemente non raggiunta
- Usare un approccio a minimo-impegno:
  - Non ordinare passi a meno che non sia strettamente necessario
- Onorare link causali $S_1 \xrightarrow{c} S_2$ che **proteggono** una condizione *c*:
  - Non aggiungere mai un passo intermedio S_3 che viola *c*
  - Se una azione parallela minaccia (**threatens**) *c* (cioè, ha l'effetto di negare (in gergo, **clobbering**) *c*, risolvere la minaccia aggiungendo vincoli temporali:
    - Ordinare S_3 prima di S_1 (**demotion**), oppure
    - Ordinare S_3 dopo S_2 (**promotion**)

Risoluzione delle Minacce



(a) Minaccia (b) Demotion (c) Promotion

Planning Non-lineare



P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY

P: ON(B,A)
ON(C,B)
-:
+:

Precondizioni Aperte

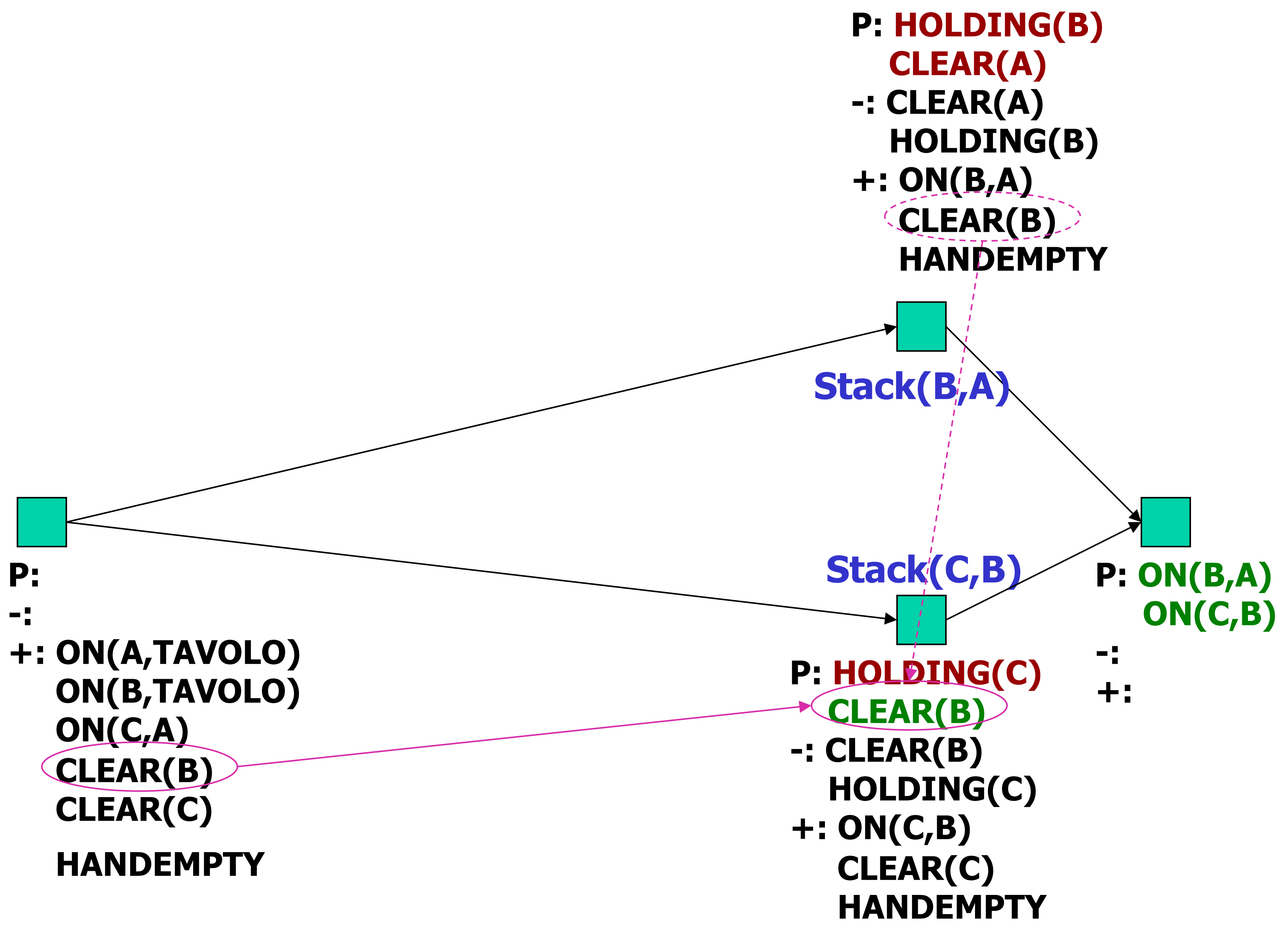
Il piano è incompleto

P: HOLDING(B)
CLEAR(A)
-: CLEAR(A)
HOLDING(B)
+: ON(B,A)
CLEAR(B)
HANDEMPTY

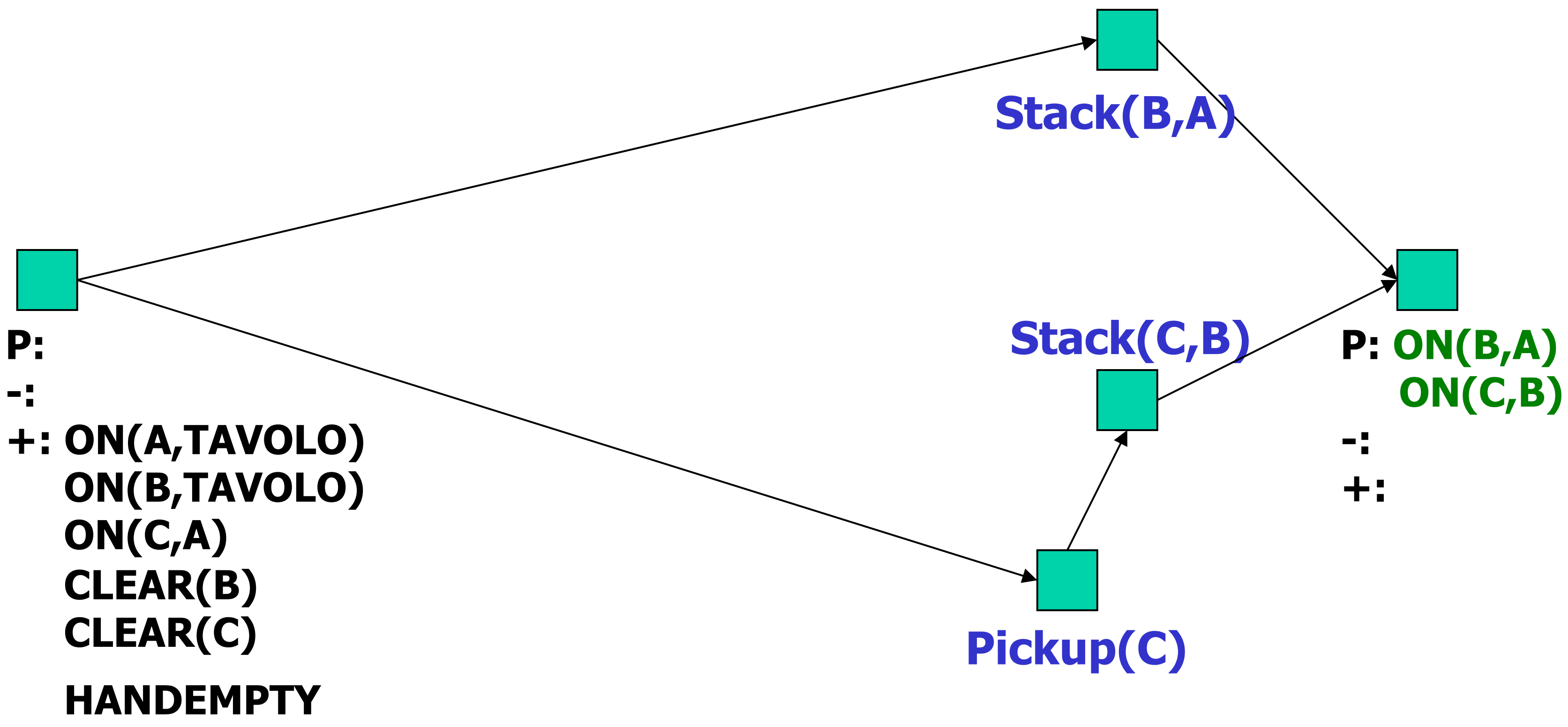
Stack(B,A)

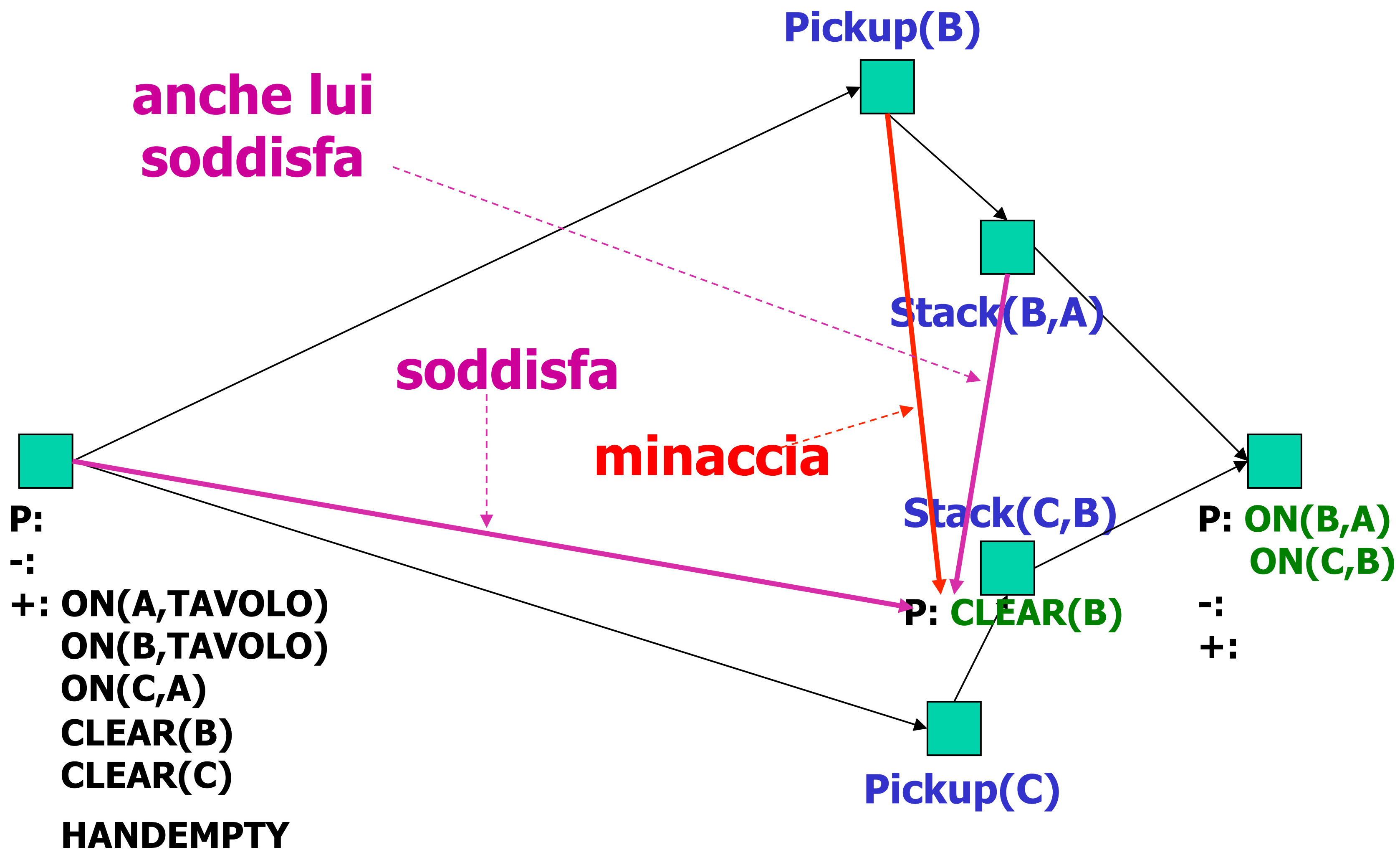
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY

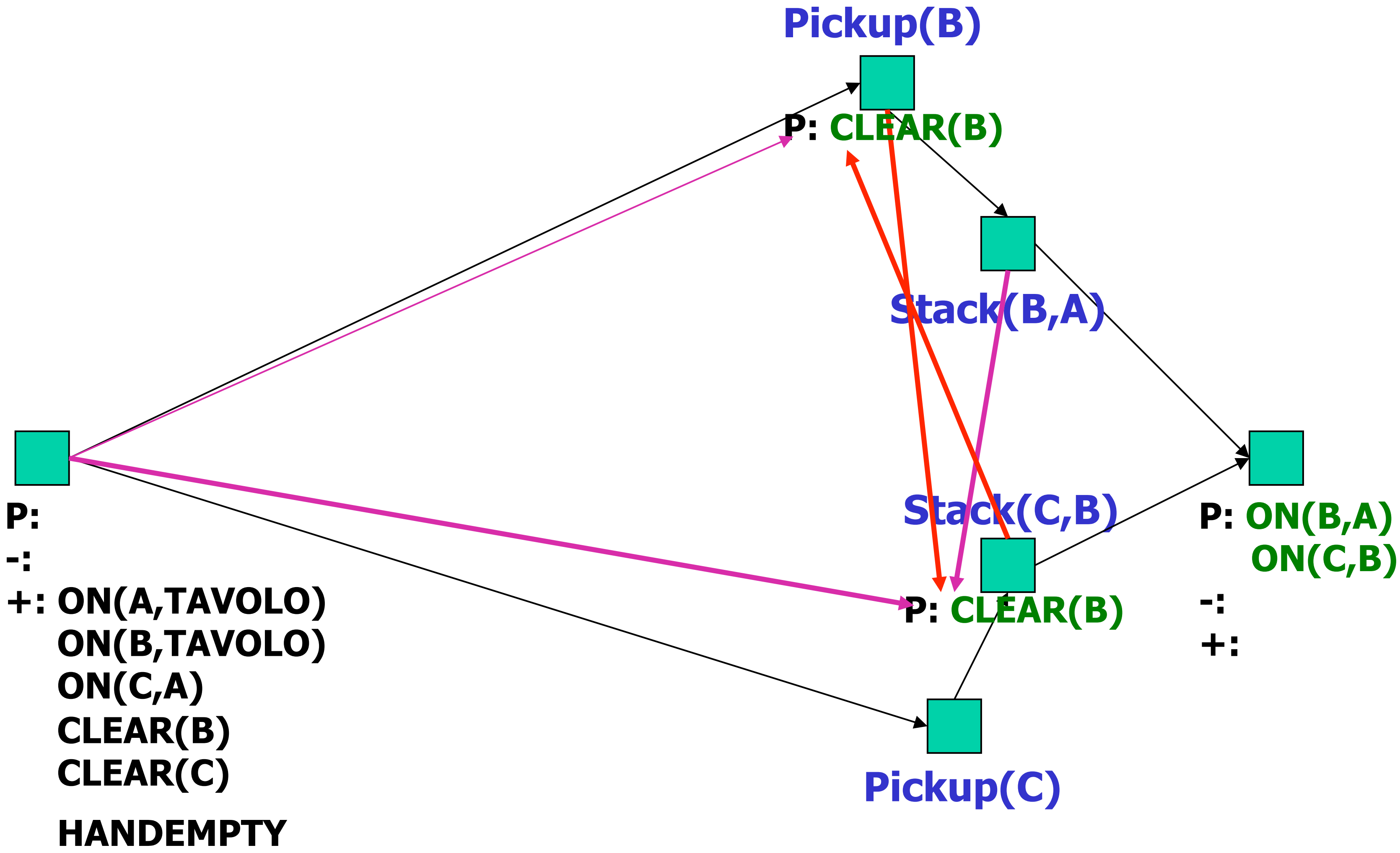
P: ON(B,A)
ON(C,B)
-:
+:



P: HOLDING(B)
CLEAR(A)
-: CLEAR(A)
HOLDING(B)
+: ON(B,A)
CLEAR(B)
HANDEMPTY
Stack(B,A)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Stack(C,B)
P: HOLDING(C)
CLEAR(B)
-: CLEAR(B)
HOLDING(C)
+: ON(C,B)
CLEAR(C)
HANDEMPTY
P: ON(B,A)
ON(C,B)
-:
+:

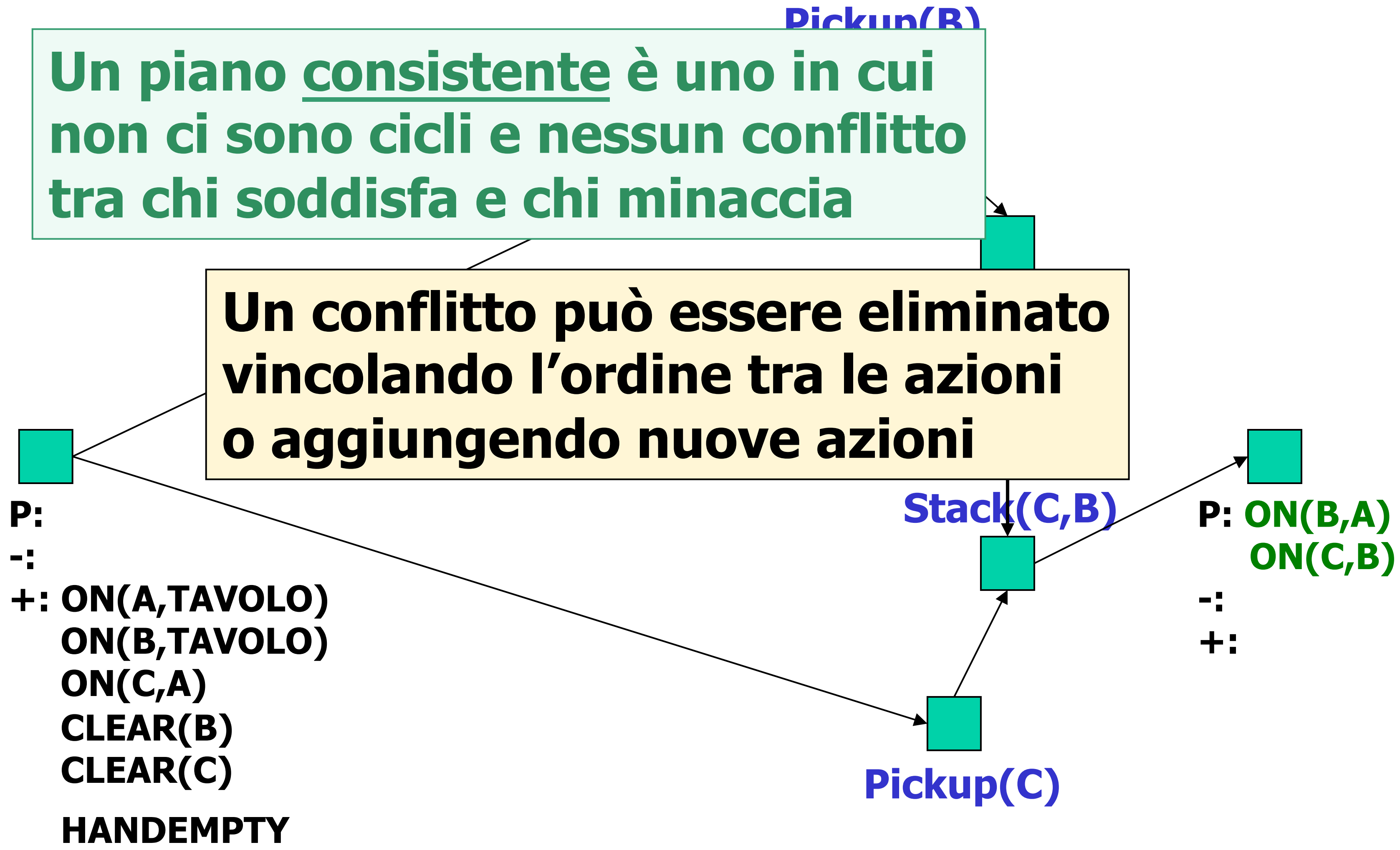

Stack(B,A)
Stack(C,B)
Pickup(C)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
P: ON(B,A)
ON(C,B)
-:
+:

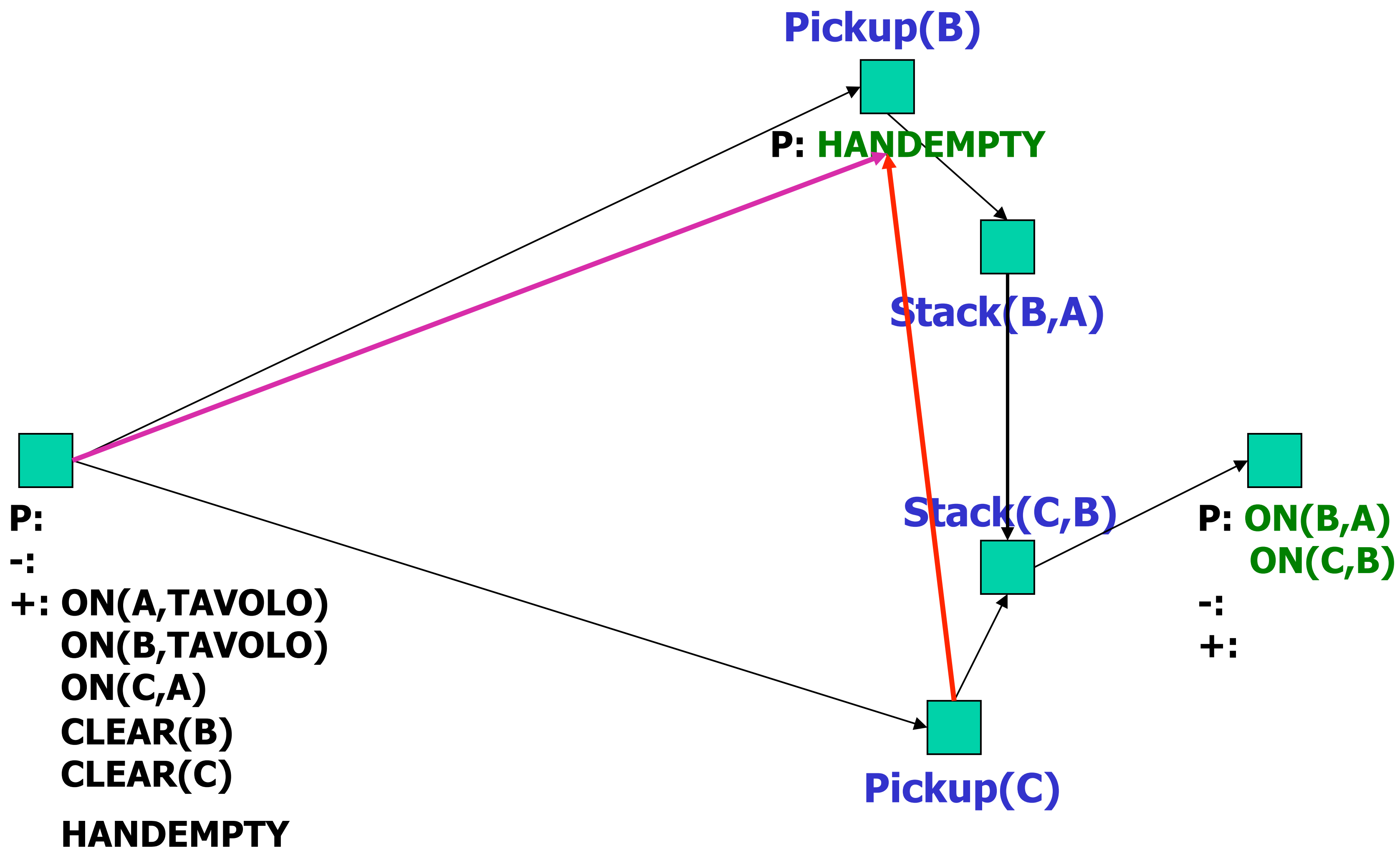

Pickup(B)
anche lui
soddisfa
Stack(B,A)
soddisfa
minaccia
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Stack(C,B)
P: CLEAR(B)
P: ON(B,A)
ON(C,B)
-:
+:
Pickup(C)

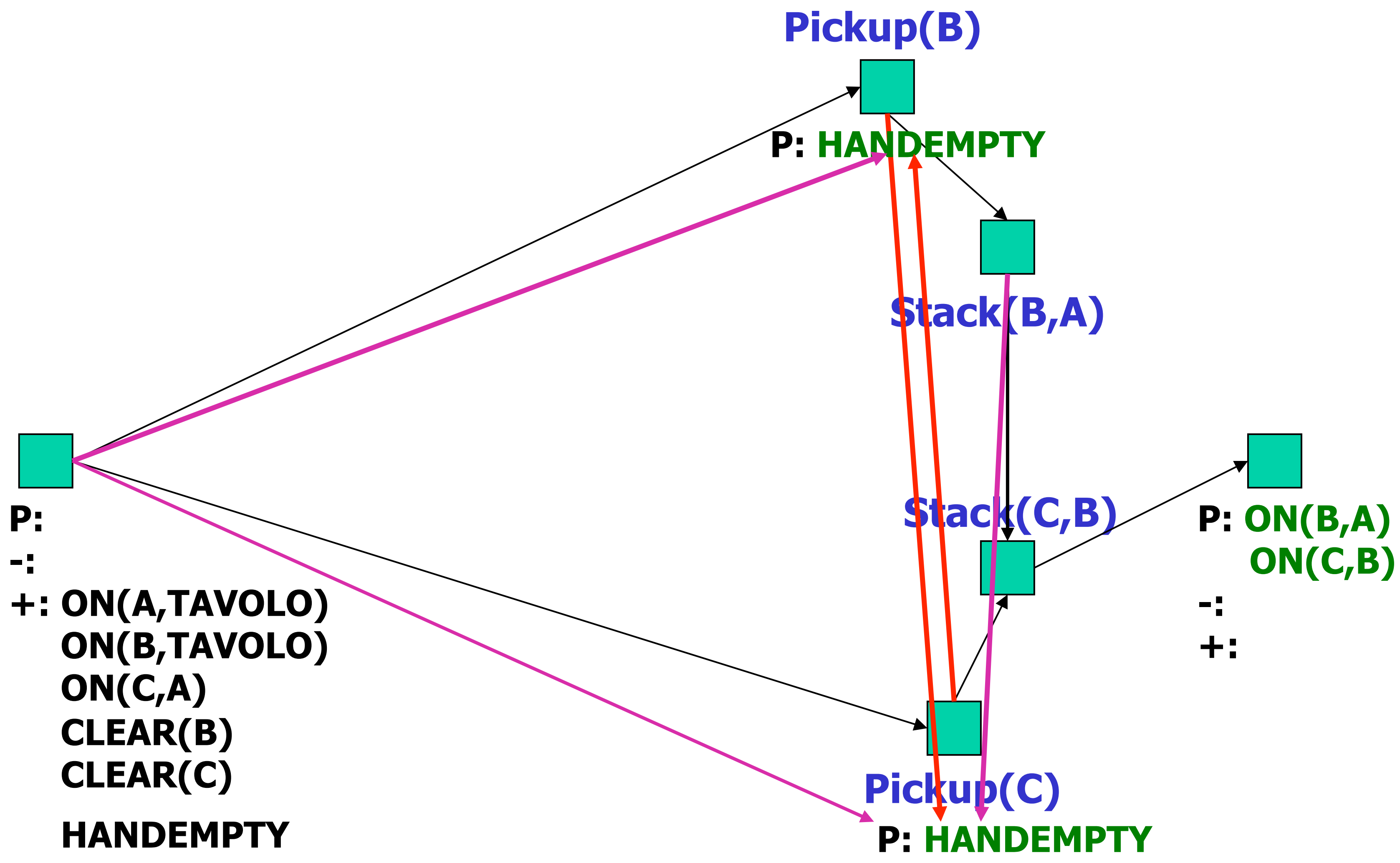

Pickup(B)
P: CLEAR(B)
Stack(B,A)
Stack(C,B)
P: CLEAR(B)
Pickup(C)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
P: ON(B,A)
ON(C,B)
-:
+:

Un piano <u>consistente</u> è uno in cui non ci sono cicli e nessun conflitto tra chi soddisfa e chi minaccia

Un conflitto può essere eliminato vincolando l’ordine tra le azioni o aggiungendo nuove azioni




Pickup(B)
P: HANDEMPTY
Stack(B,A)
Stack(C,B)
Pickup(C)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
P: ON(B,A)
ON(C,B)
-:
+:

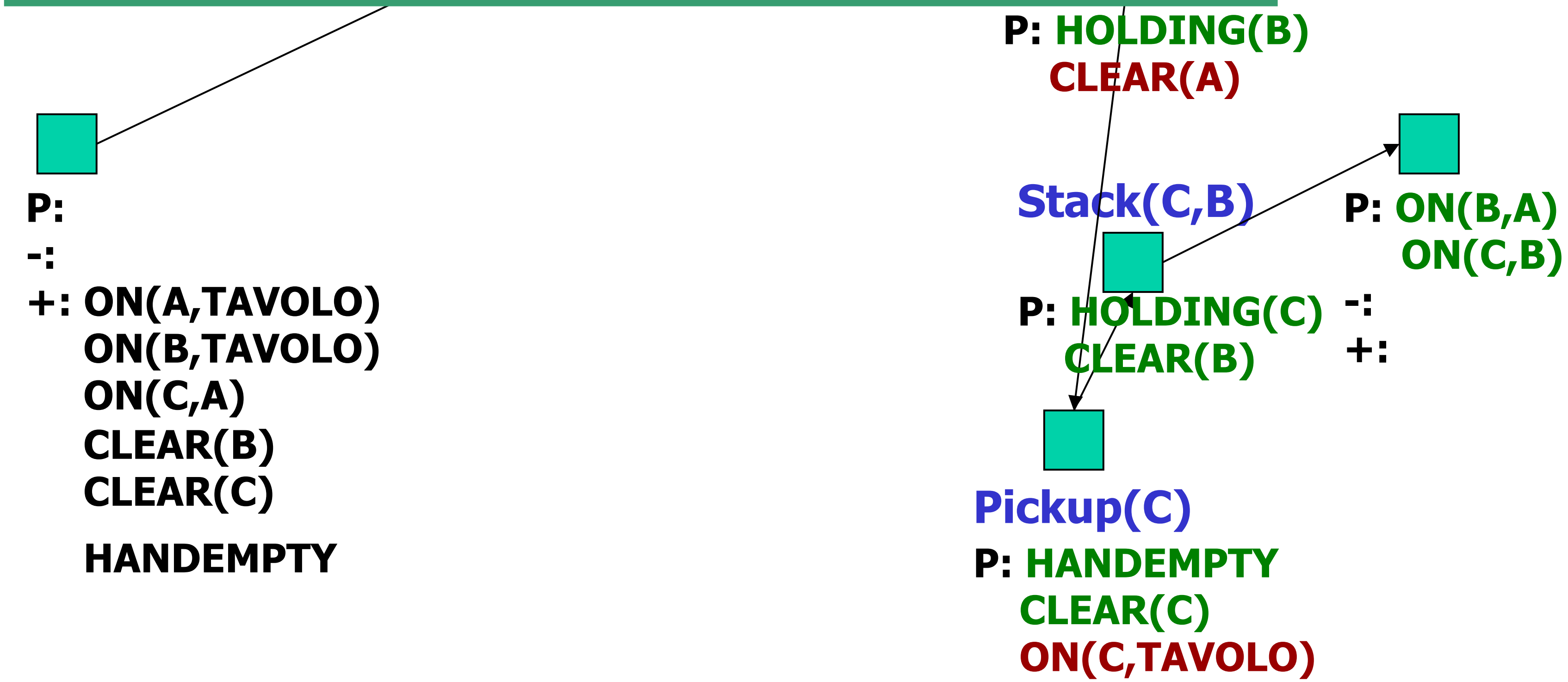


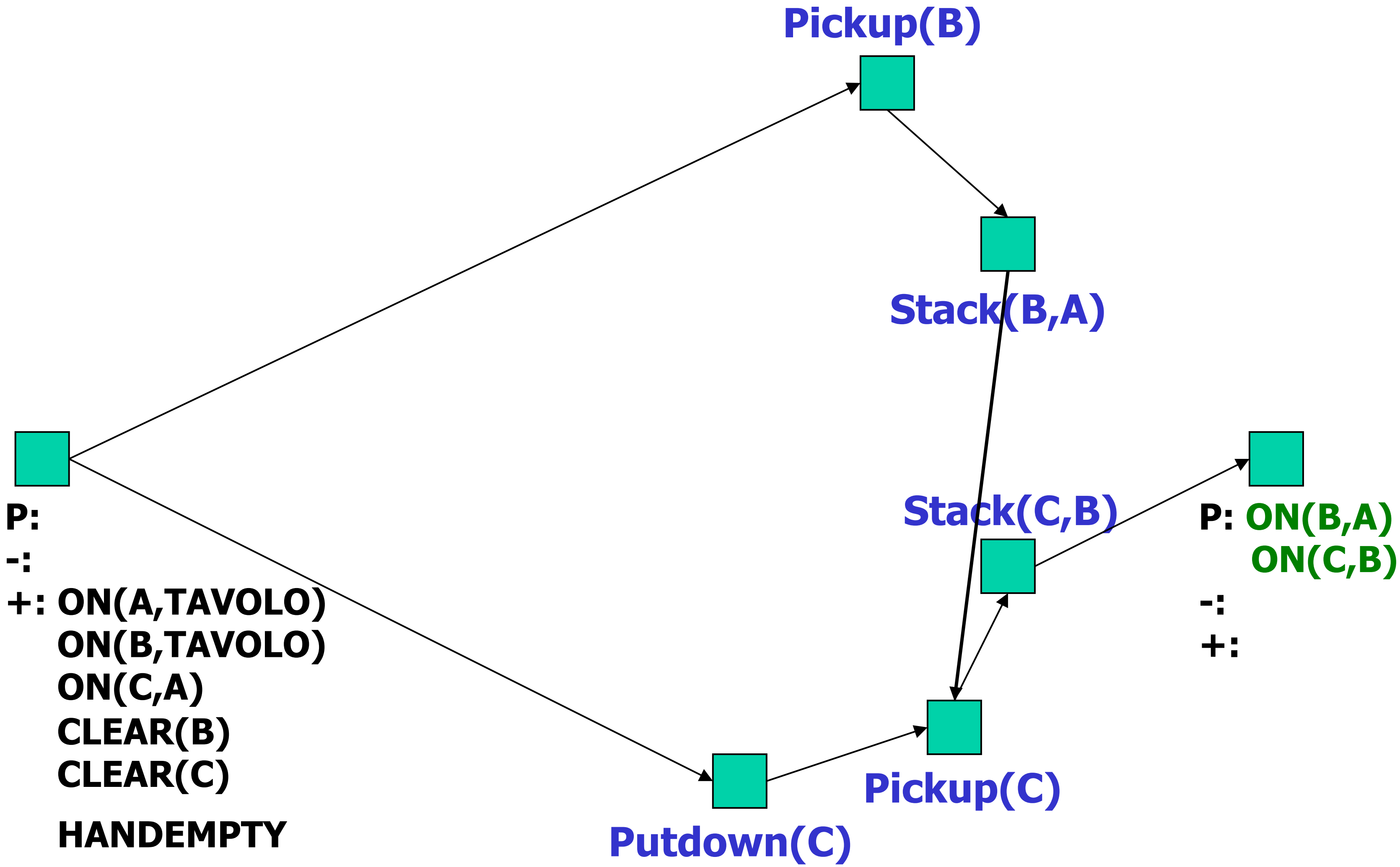
Pickup(B)
P: HANDEMPTY
Stack(B,A)
Stack(C,B)
P: ON(B,A)
ON(C,B)
-:
+:
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Pickup(C)
P: HANDEMPTY

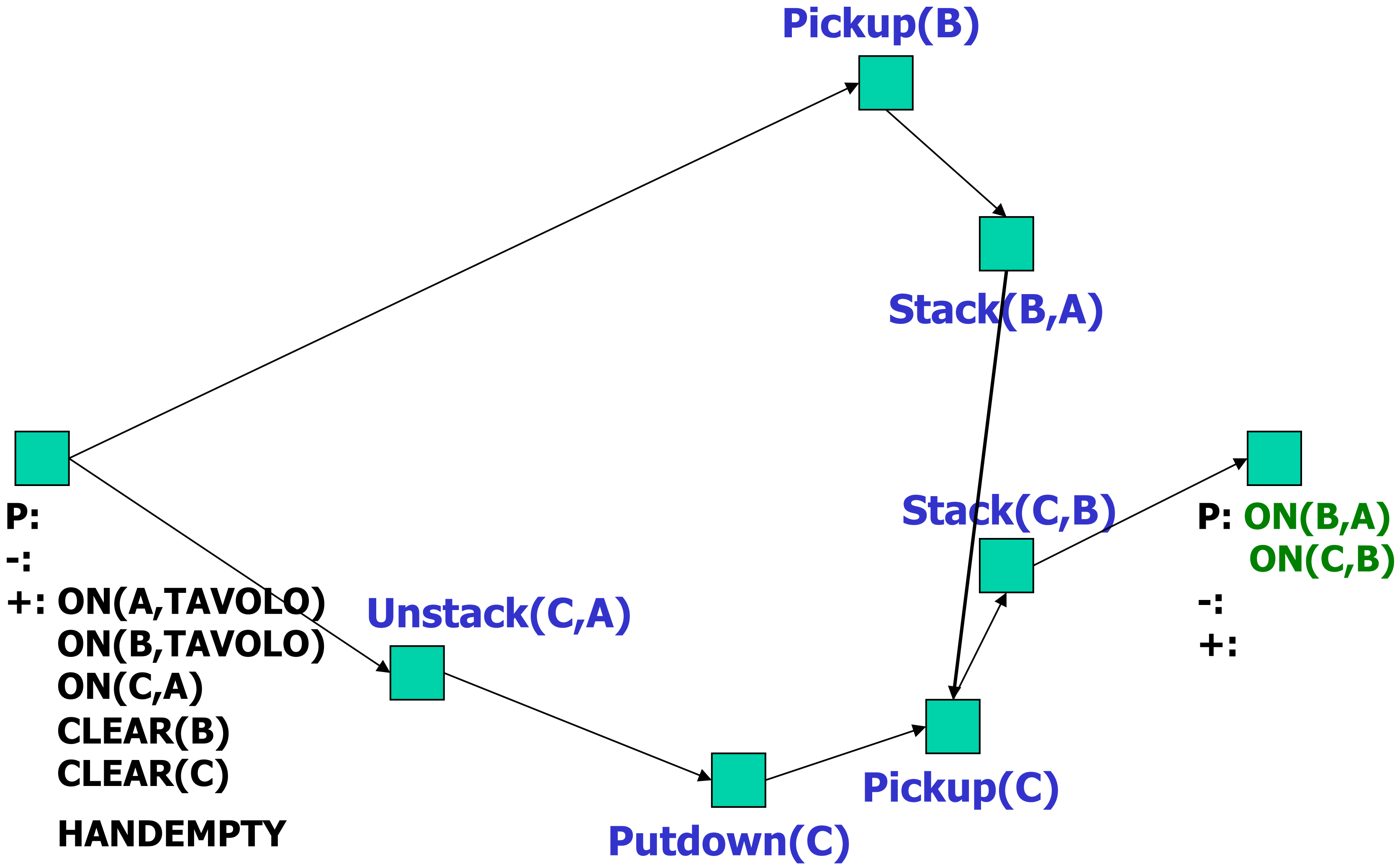
~ euristica Most-constrained-variable in CSP

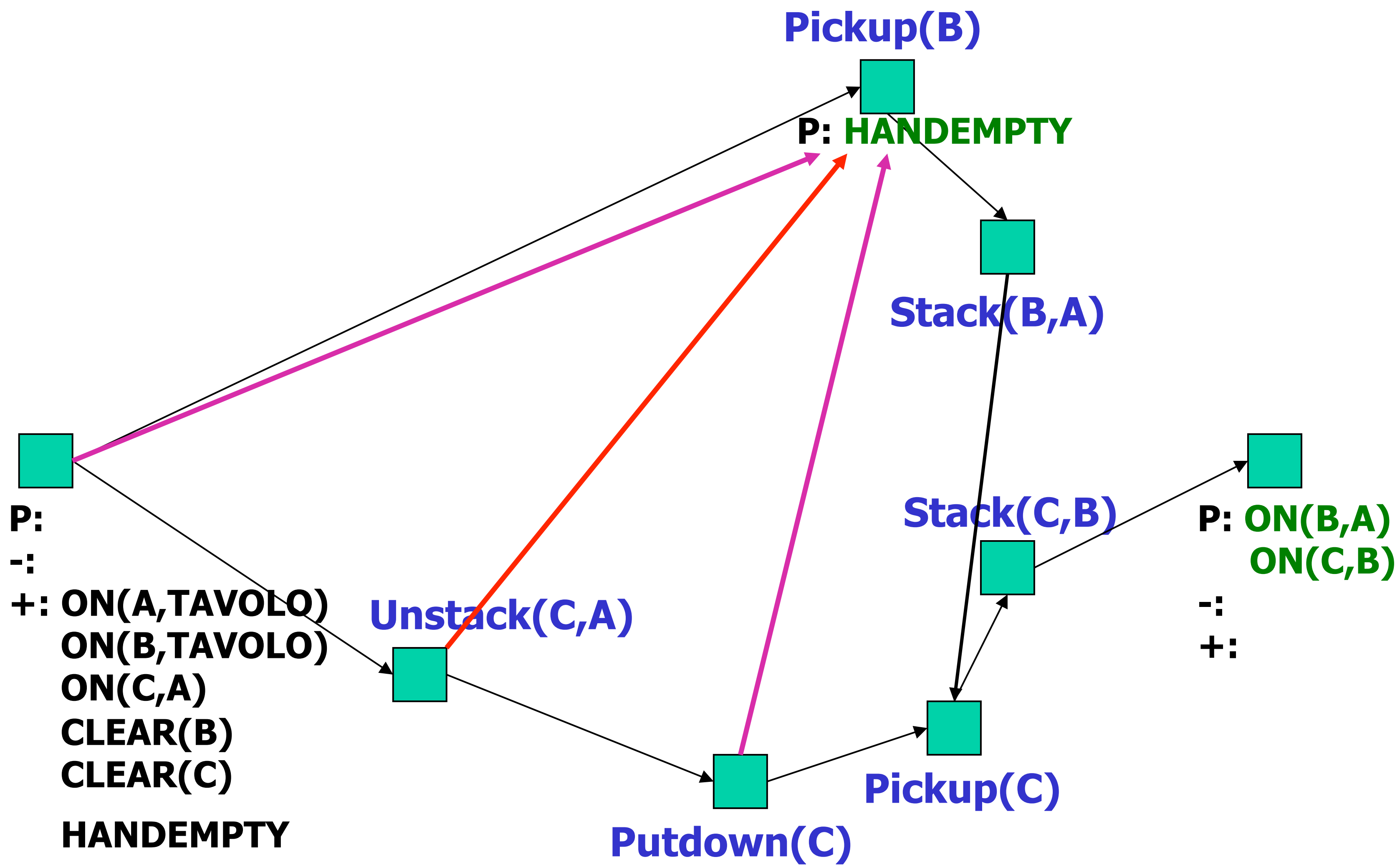
→ sceglie la precondizione non raggiunta che può essere soddisfatta nel <u>minor</u> numero di modi possibile

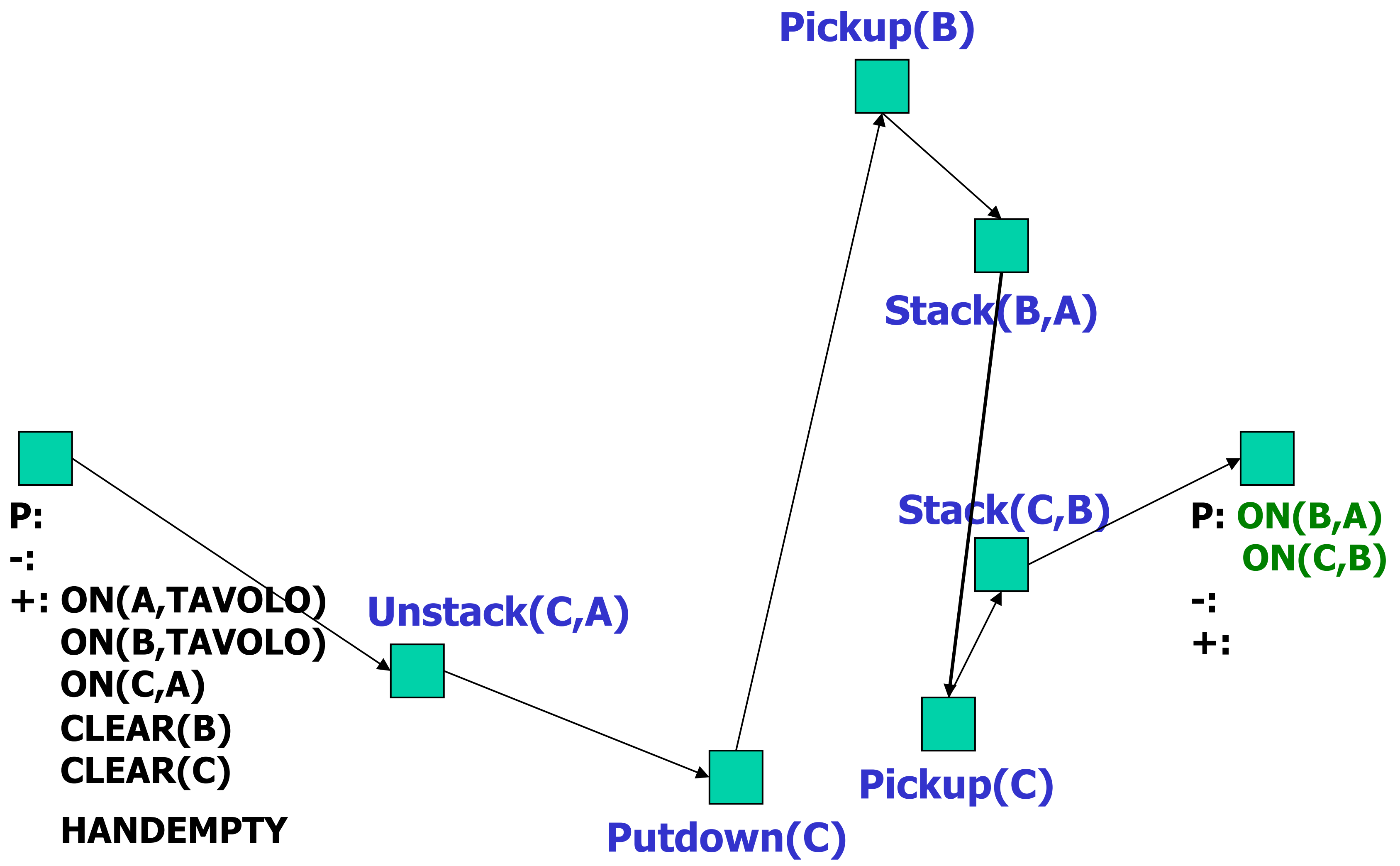
→ ON(C,TAVOLO)



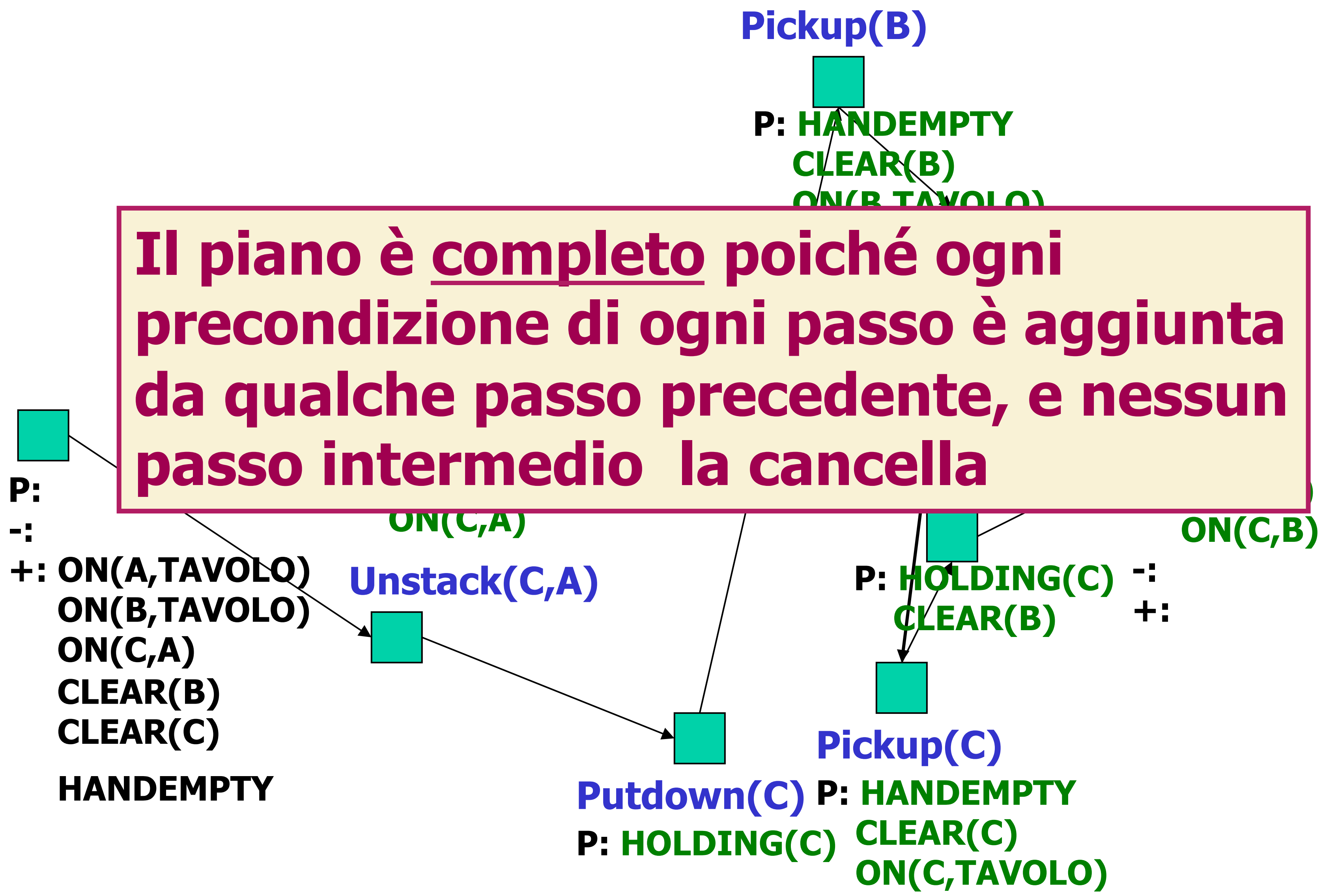

Pickup(B)
Stack(B,A)
Stack(C,B)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
P: ON(B,A)
ON(C,B)
-:
+:
Pickup(C)
Putdown(C)


Pickup(B)
Stack(B,A)
Stack(C,B)
Unstack(C,A)
Pickup(C)
Putdown(C)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
P: ON(B,A)
ON(C,B)
-:
+:


Pickup(B)
P: HANDEMPTY
Stack(B,A)
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Unstack(C,A)
Putdown(C)
Pickup(C)
Stack(C,B)
P: ON(B,A)
ON(C,B)
-:
+:



Pickup(B)
Stack(B,A)
Stack(C,B)
P: ON(B,A)
ON(C,B)
-:
+:
P:
-:
+: ON(A,TAVOLO)
ON(B,TAVOLO)
ON(C,A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Unstack(C,A)
Putdown(C)
Pickup(C)



Il piano è <u>completo</u> poiché ogni precondizione di ogni passo è aggiunta da qualche passo precedente, e nessun passo intermedio la cancella

Proprietà di POP

- Algoritmo nondeterministico: in caso di fallimento effettua backtracking sui punti di scelta (choice point)
  - scelta di passo per raggiungere un sotto-obiettivo
  - scelta di demotion o promotion in caso di minaccia
- POP è corretto, completo e sistematico (nessuna ripetizione)
- Esistono estensioni (es. azioni rappresentate con FOL)
- Efficiente se fornito di euristiche derivate dalla descrizione del problema

POP: variabili non istanziate

- azioni rappresentate con FOL

  Es. Mondo dei blocchi

  precondizione aperta: On(a,b)

  azione:

  OP(ACTION: MoveB(Z,X,Y),

  PRECOND: $On(Z,X) \land Clear(Z) \land Clear(Y)$,

  EFFECT: $On(Z,Y) \land Clear(X) \land \neg On(Z,X) \land \neg Clear(Y)$)

- notare che $On(a,b) = On(Z,Y)\theta$ con $\theta = \{Z/a, Y/b\}$ e quindi bisogna applicare $MoveB(Z,X,Y)\theta$

POP: variabili non istanziate

- M1: MoveB(Z,X,Y)θ

OP(ACTION: MoveB(a,X,b),
PRECOND: On(a,X) ∧ Clear(a) ∧ Clear(b),
EFFECT: On(a,b) ∧ Clear(X) ∧ ¬On(a,X) ∧ ¬Clear(b))

- Notare che X rimane non istanziata
- Supponiamo di aggiungere M1 al piano. Quindi aggiungiamo il link causale:

$$\text{MoveB(a,X,b)} \xrightarrow{\text{On(a,b)}} \text{Finish}$$

- Se nel piano c'è già una azione M2 con effetto ¬On(a,Q), c'è una minaccia solo se Q prende b
- Per rappresentare questa situazione si deve aggiungere il vincolo

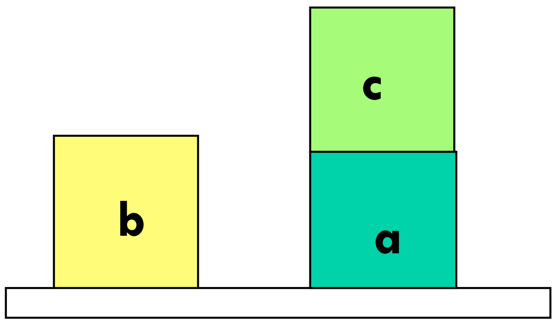
Q ≠ b (in generale var ≠ cost oppure var ≠ var)

POP con variabili

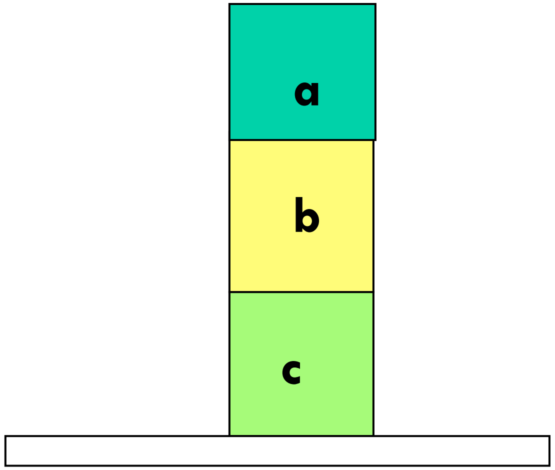
- altra azione

OP(ACTION: MoveT(W,X),
PRECOND: On(W,X) ∧ Clear(W),
EFFECT: On(W,t) ∧ Clear(X) ∧ ¬On(W,X))

t costante: significa TAVOLO



Stato iniziale



goal

POP: esempio con variabili

start

on(c,a) **on(a,t)** **on(b,t)** **clear(c)** **clear(b)**

POP: esempio con variabili

start

on(c,a) on(a,t) on(b,t) clear(c) clear(b)

on(a,b) on(b,c)

finish

POP: esempio con variabili

start

on(c,a) on(a,t) on(b,t) clear(c) clear(b)

MoveB (a,x_1,b)

on(a,b) on(b,c)

finish

POP: esempio con variabili

start

on(c,a) on(a,t) on(b,t) clear(c) clear(b)

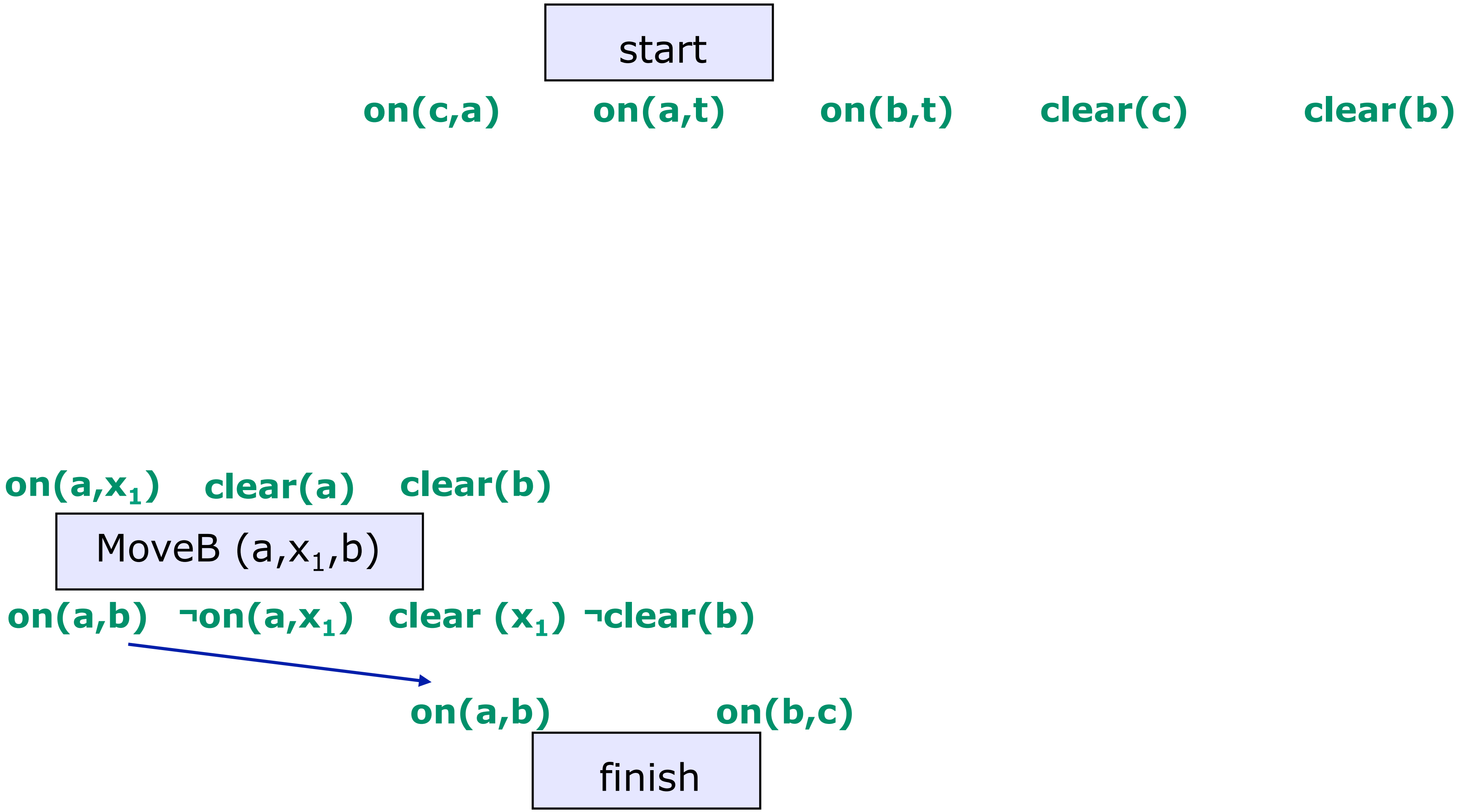
MoveB (a,x_1,b)

on(a,b) ¬on(a,x_1) clear (x_1) ¬clear(b)

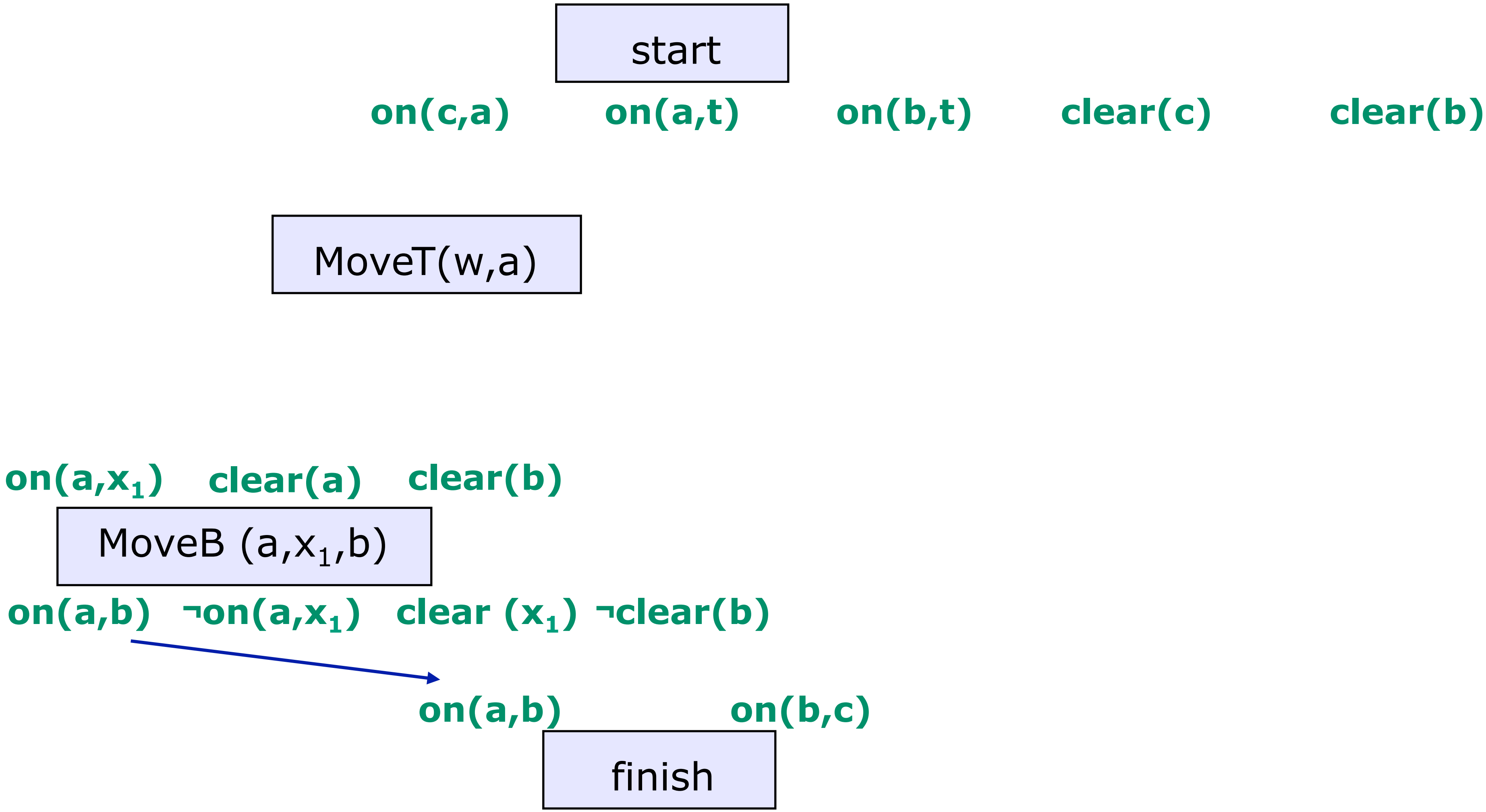
on(a,b) on(b,c)

finish

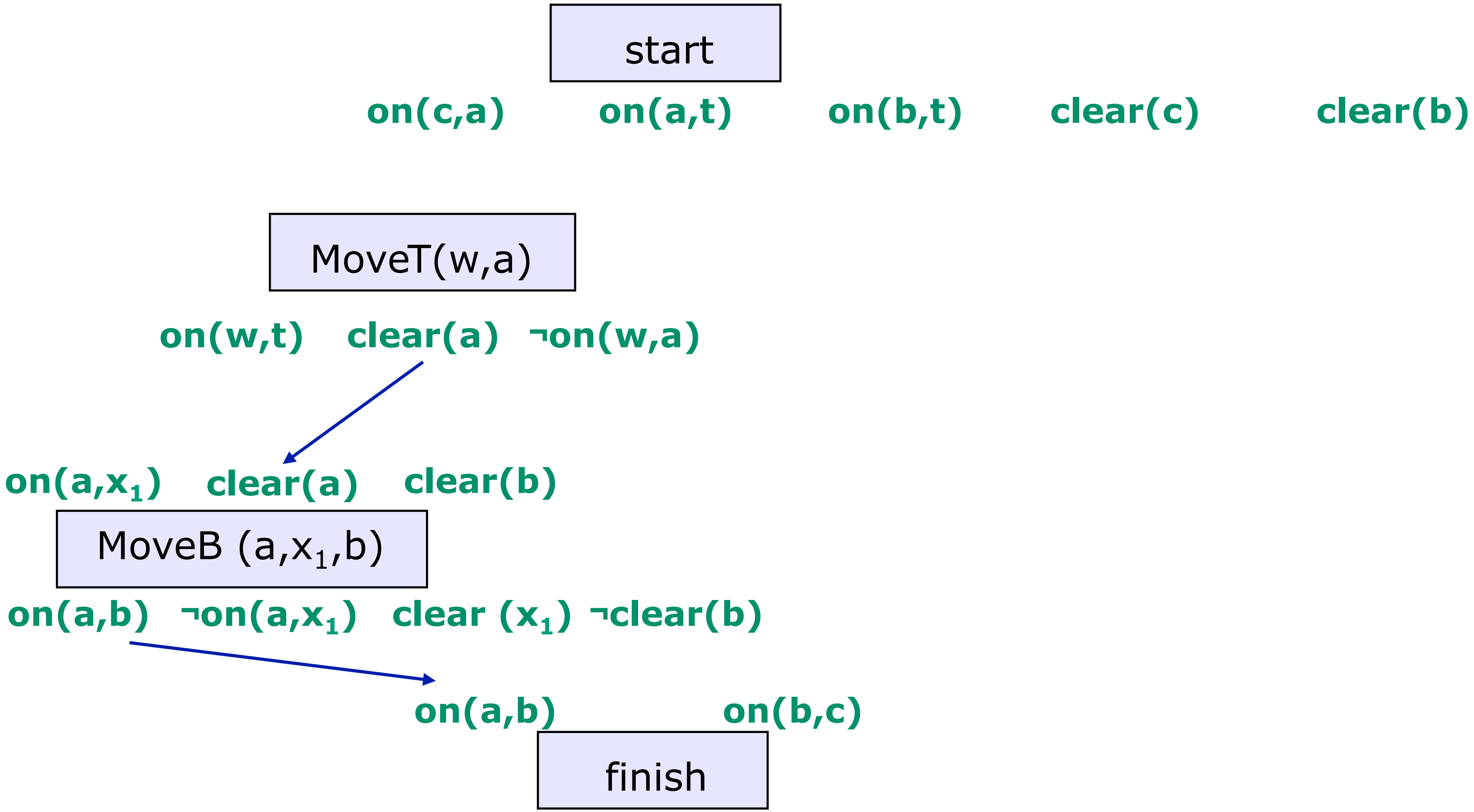
POP: esempio con variabili



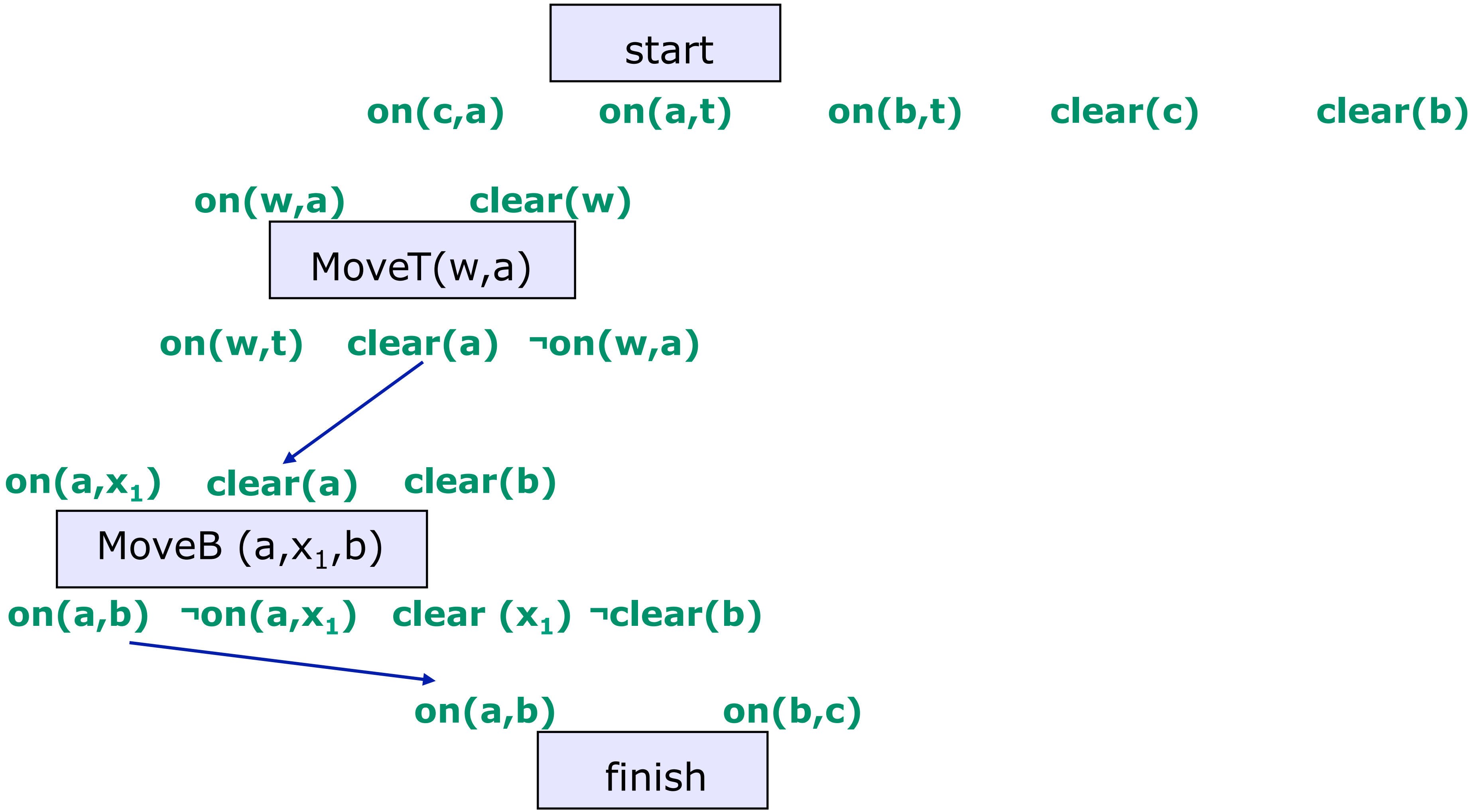
POP: esempio con variabili



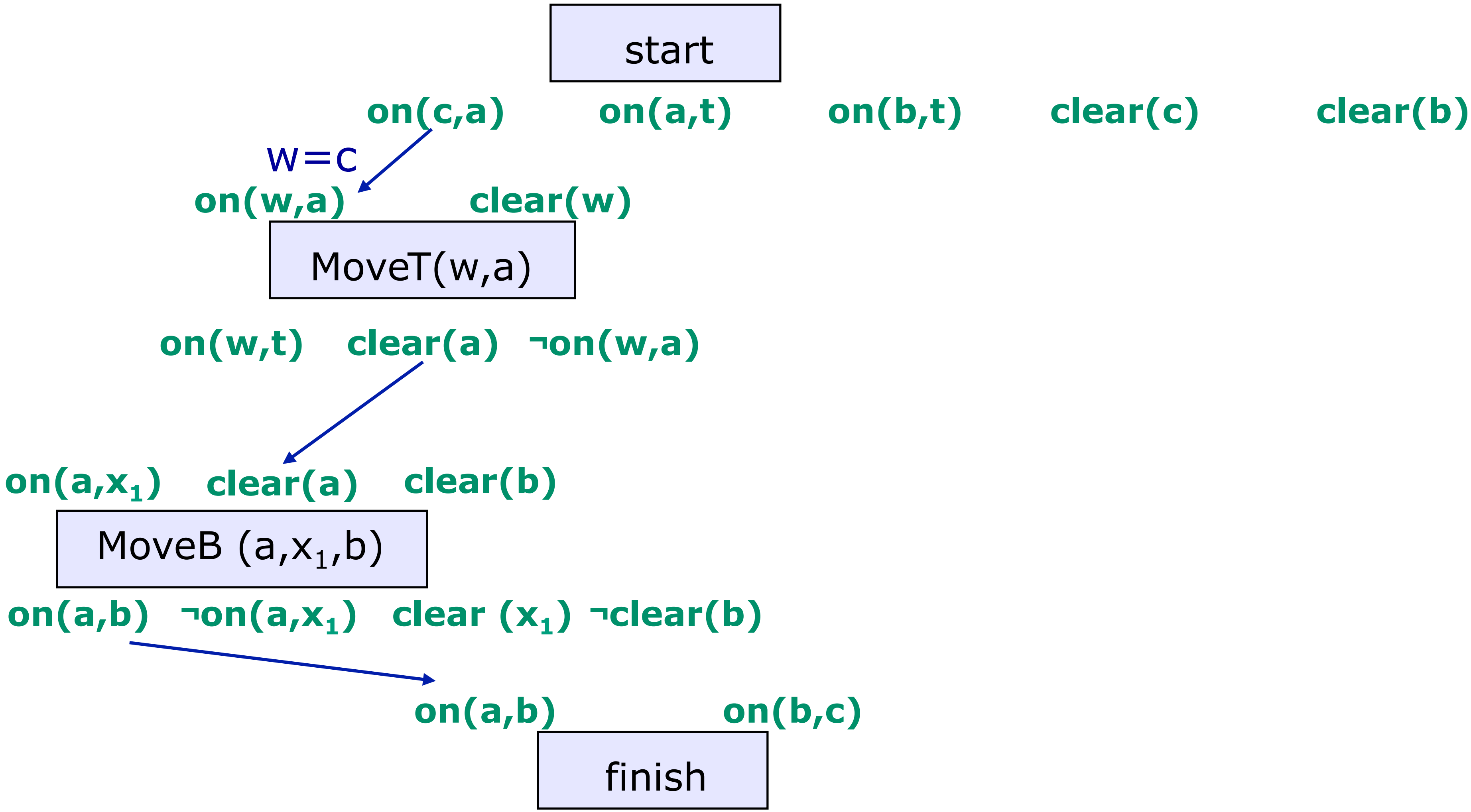
POP: esempio con variabili



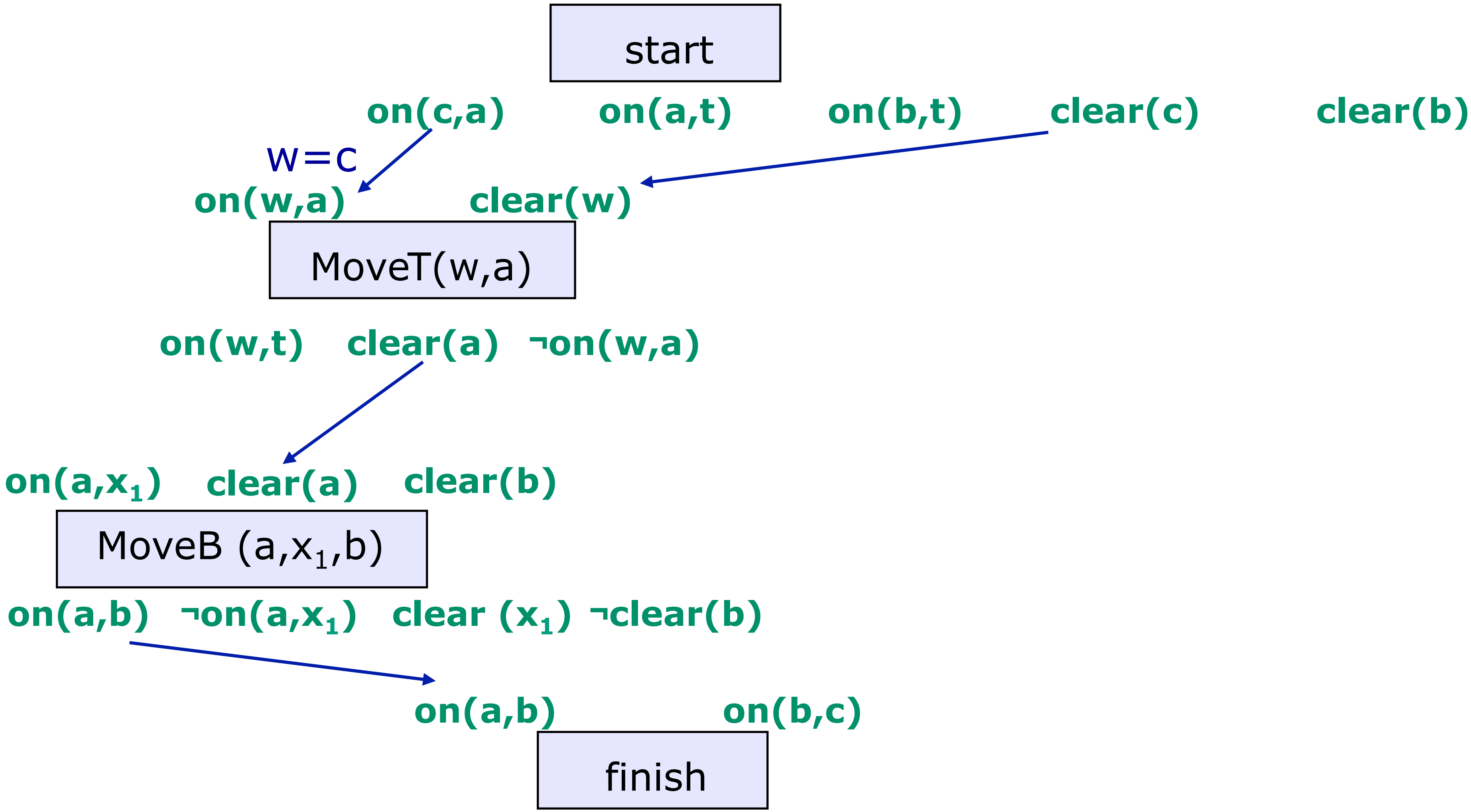
POP: esempio con variabili



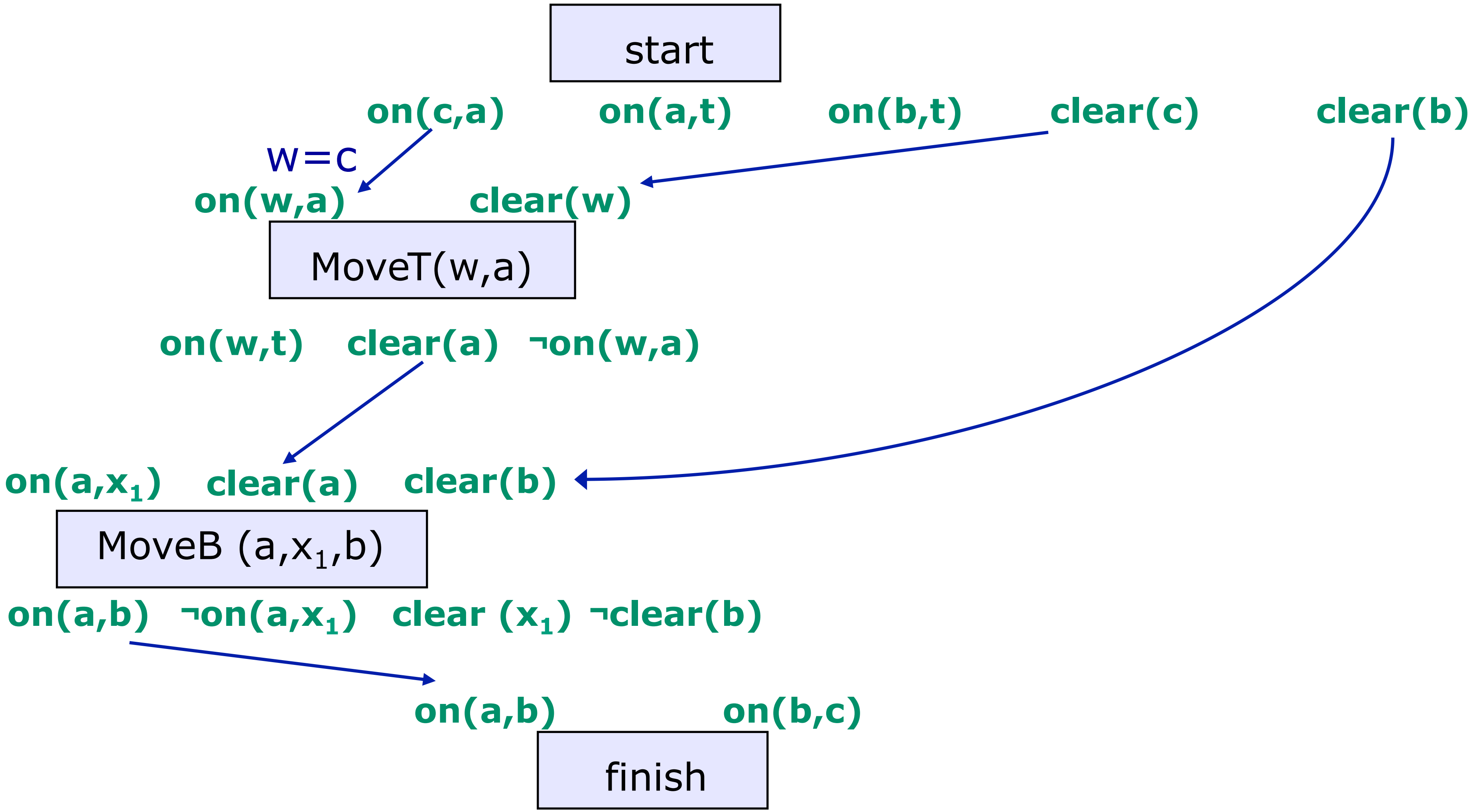
POP: esempio con variabili



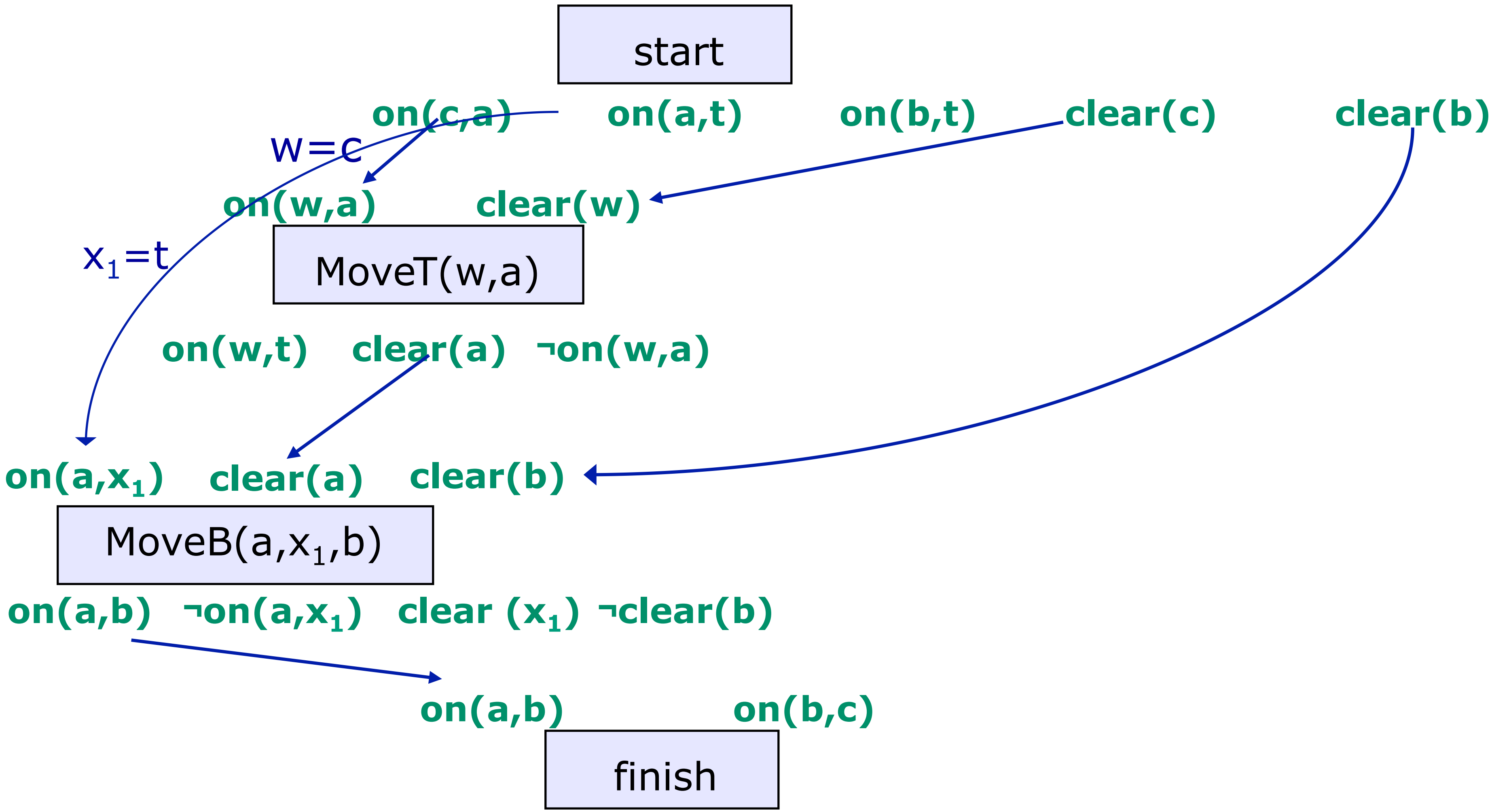
POP: esempio con variabili



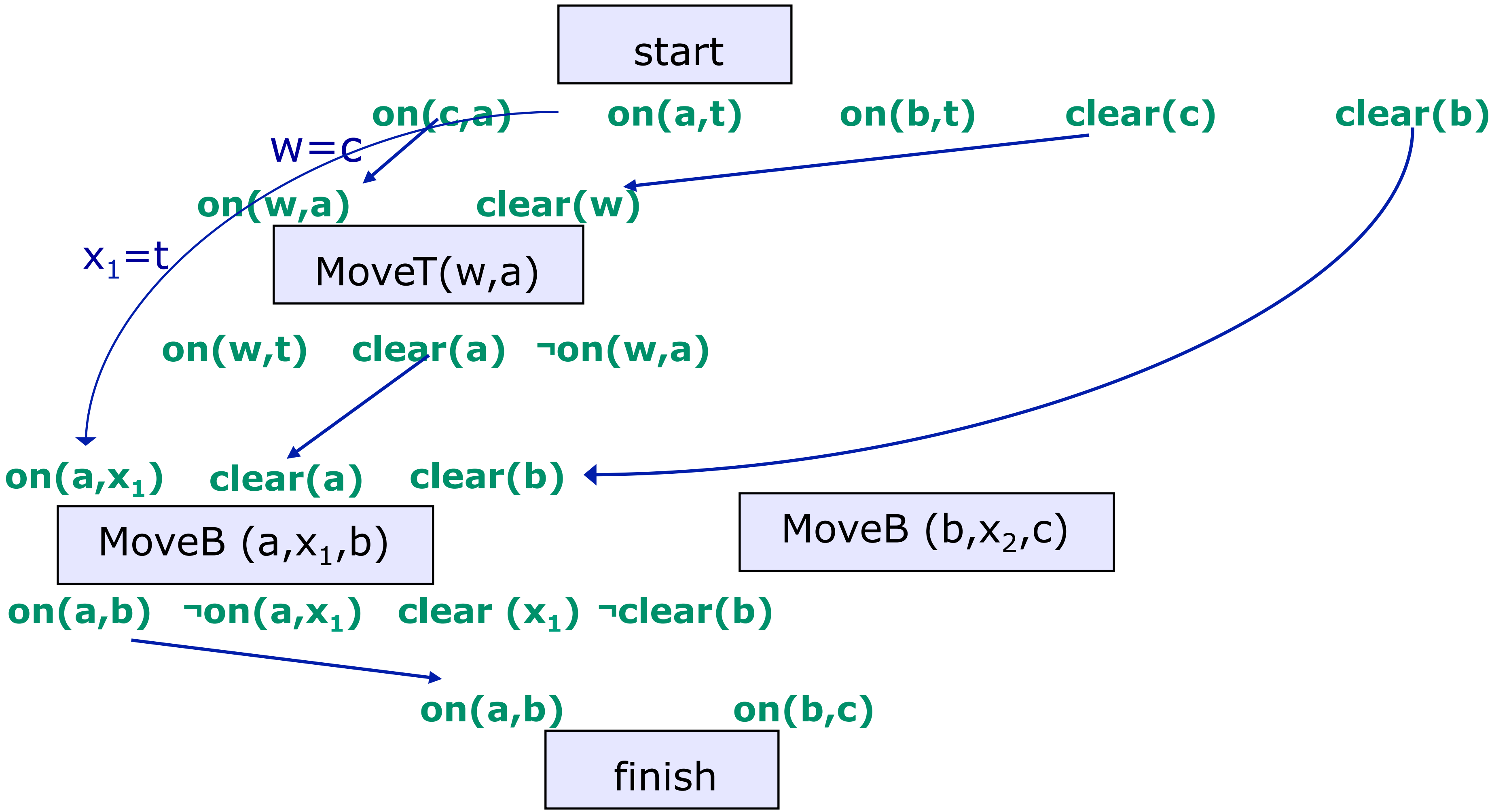
POP: esempio con variabili



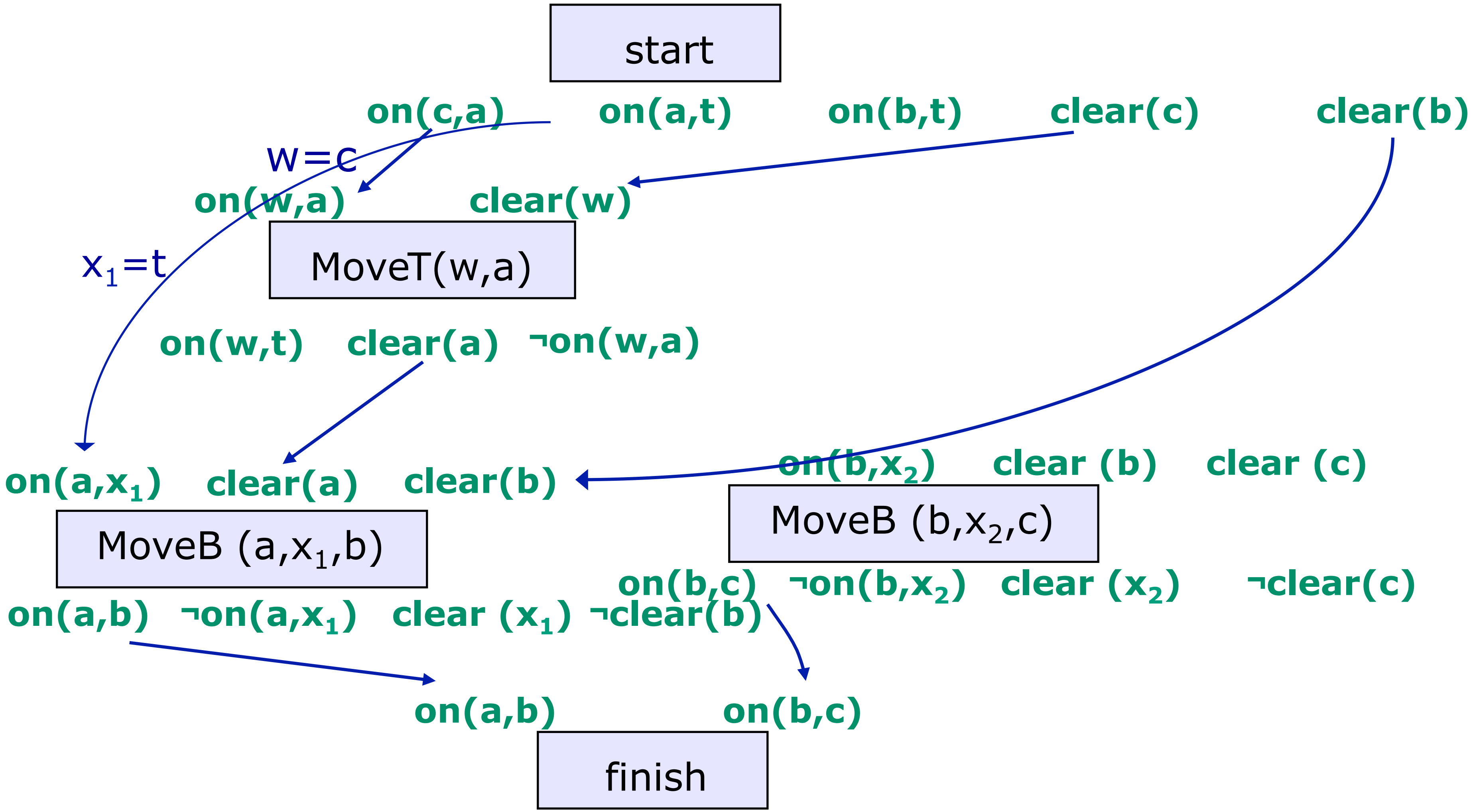
POP: esempio con variabili



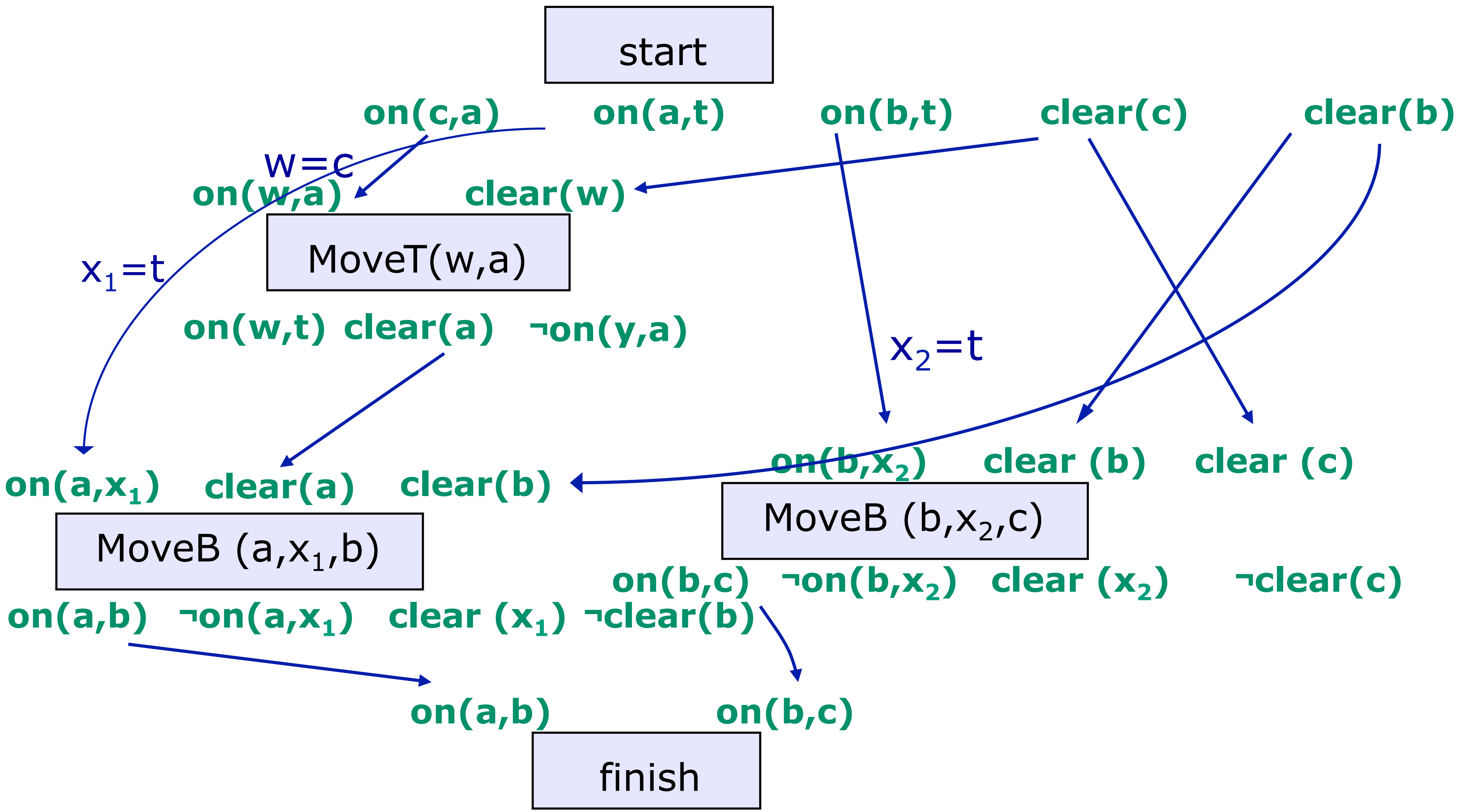
POP: esempio con variabili



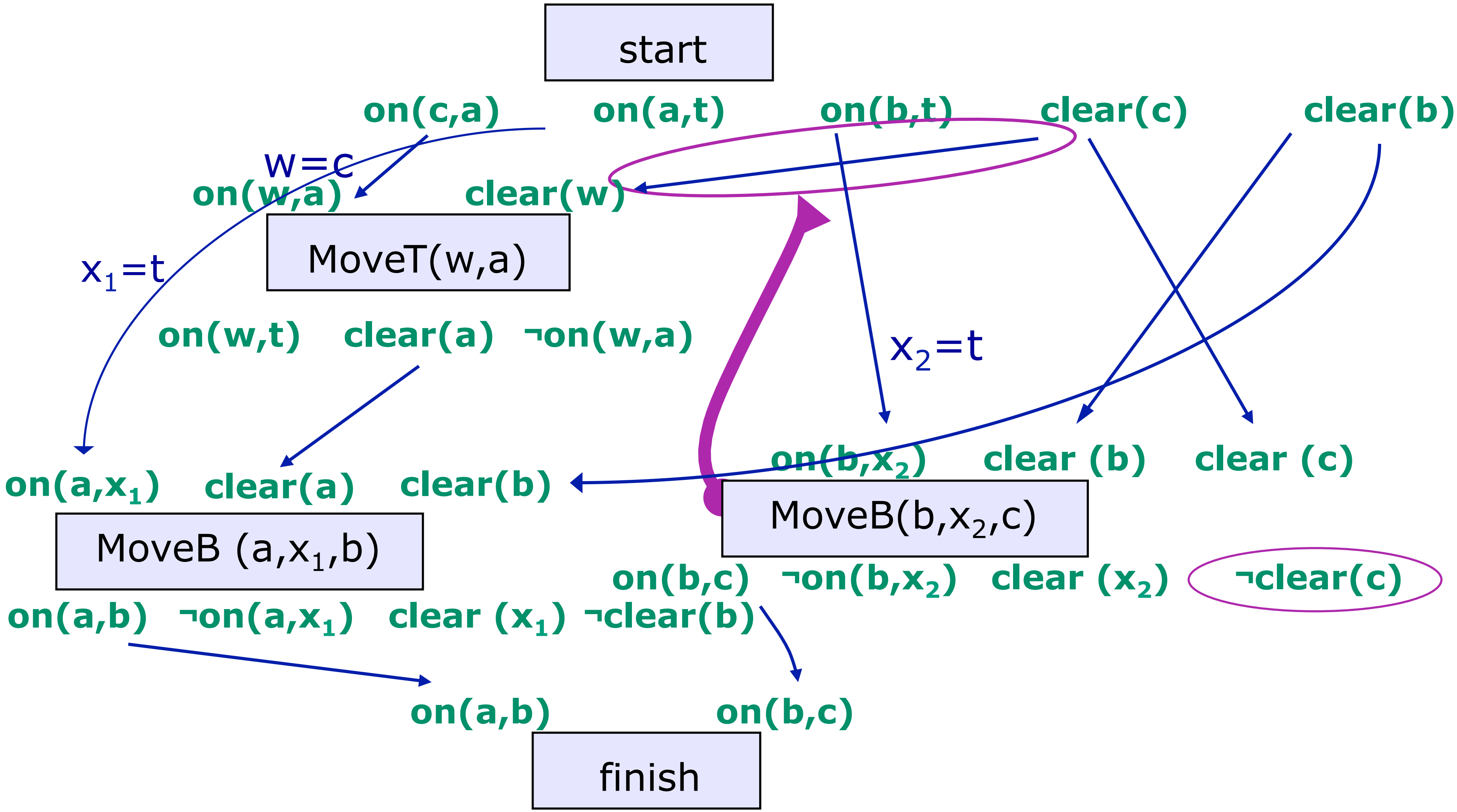
POP: esempio con variabili



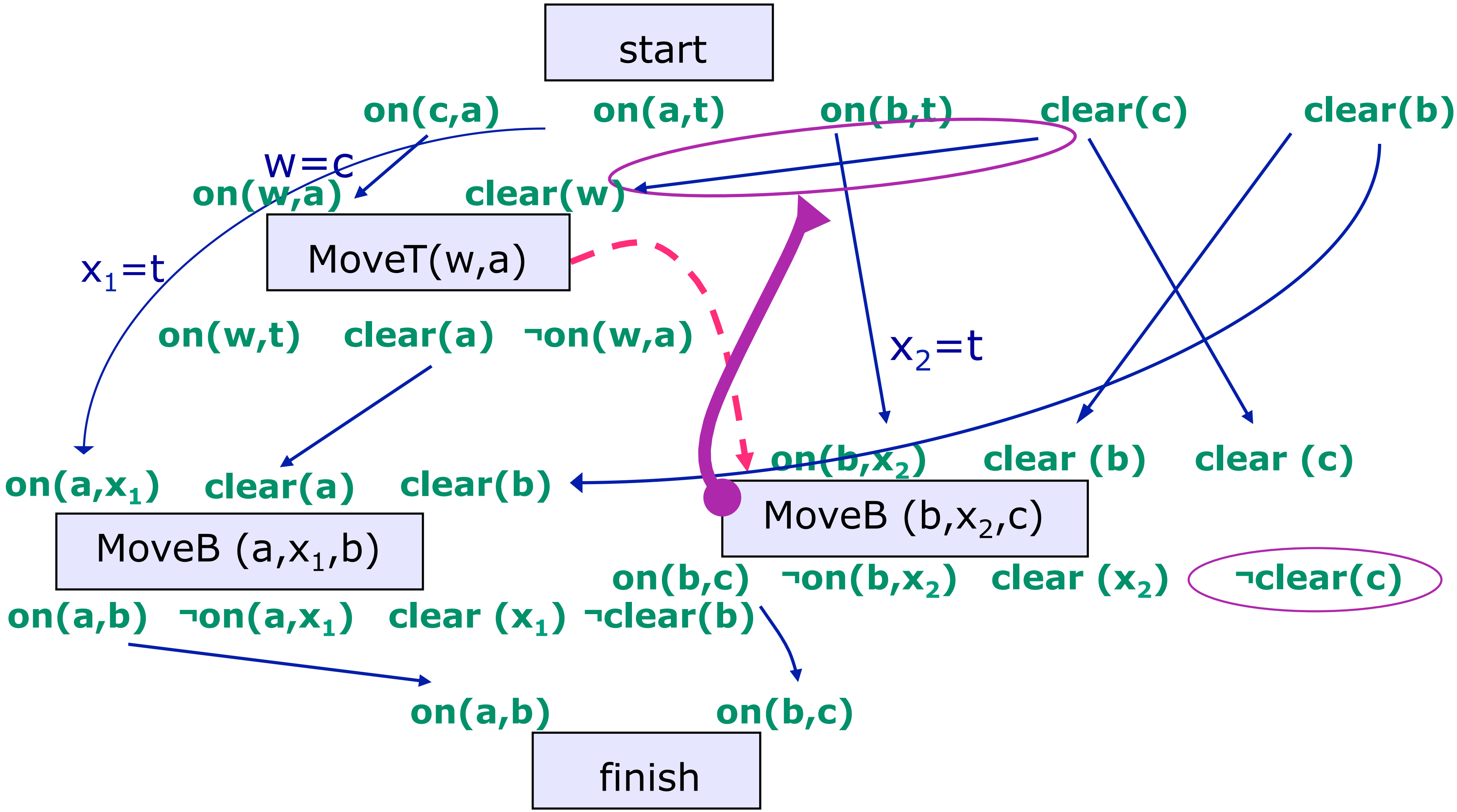
POP: esempio con variabili



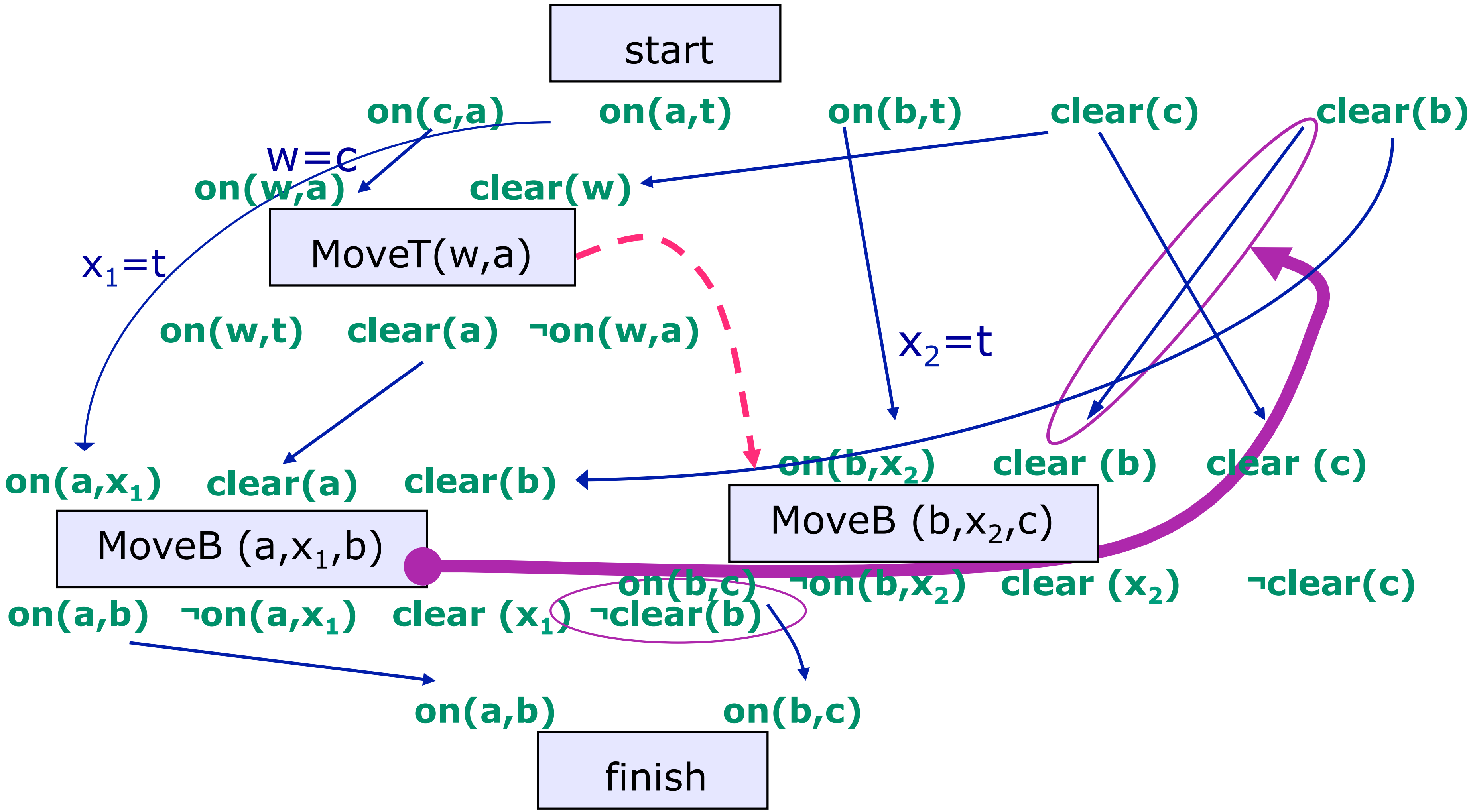
POP: esempio con variabili



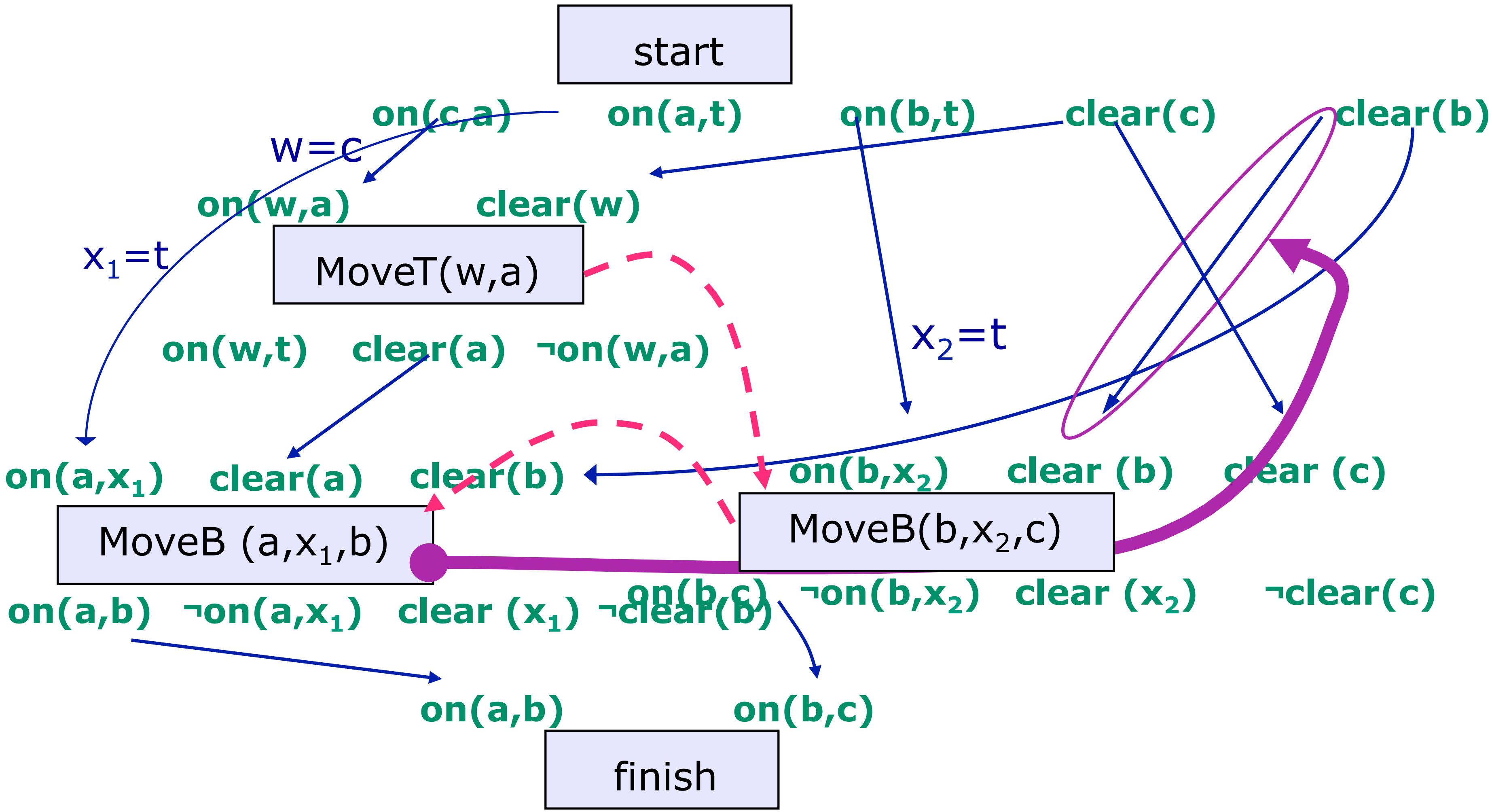
POP: esempio con variabili



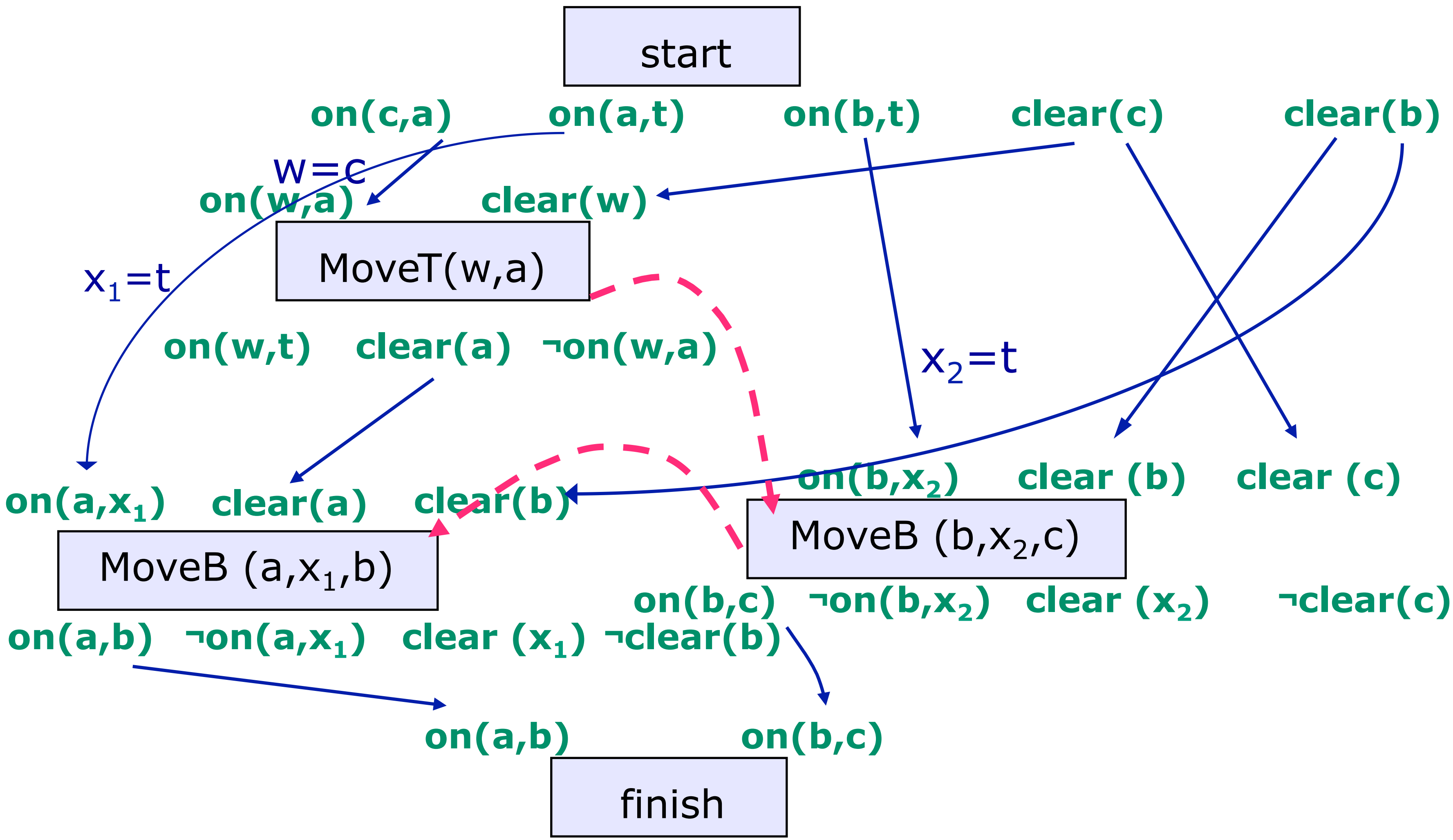
POP: esempio con variabili



POP: esempio con variabili



POP: esempio con variabili

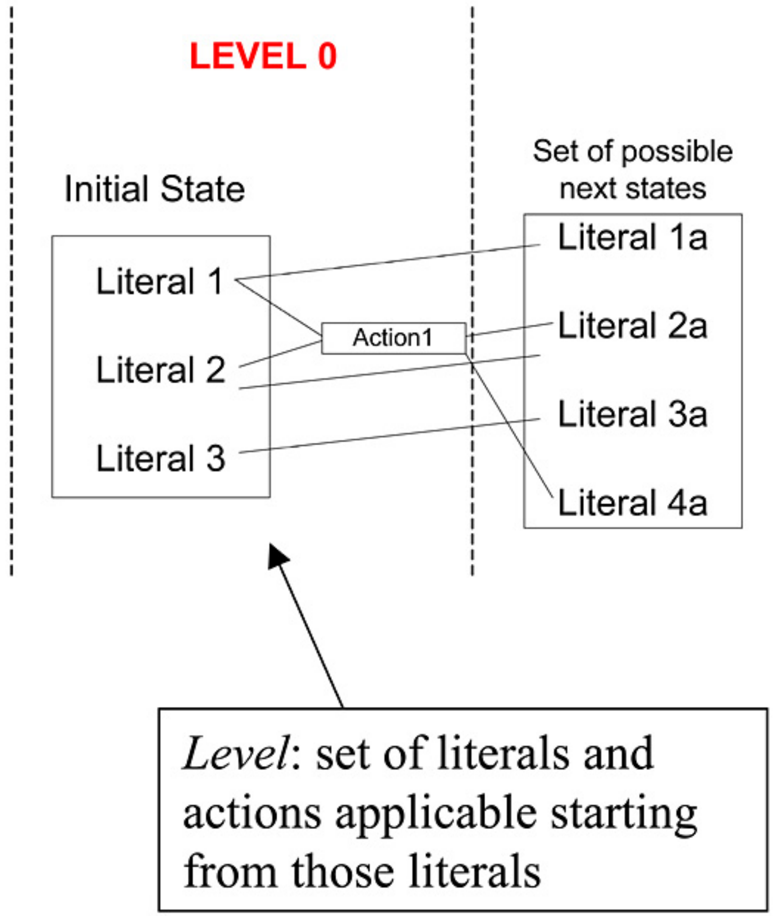


Grafi di Planning

- Necessità di avere delle buone euristiche
  - sia per planning lineari che per POP
  - nei problemi di planning non è facile derivare delle euristiche ammissibili
- Uno strumento molto utile per costruire delle euristiche ammissibile è il grafo di planning
  - raccoglie informazioni su quali piani sono impossibili non prendendo in considerazione le minacce e il “consumo” dei letterali che chiudono le precondizioni

Grafi di Planning

- Un grafo di planning è un grafo costruito a livelli:
  - il primo livello è costituito dai letterali dello stato iniziale
  - i successivi livelli sono ottenuti dalla applicazione ripetuta delle azioni che hanno i prerequisiti soddisfatti
  - Inoltre i letterali di un livello sono riportati al livello successivo (persistence actions)
- No variabili ! No troppi oggetti !



Grafi di Planning

Vediamo un esempio su un problema semplice

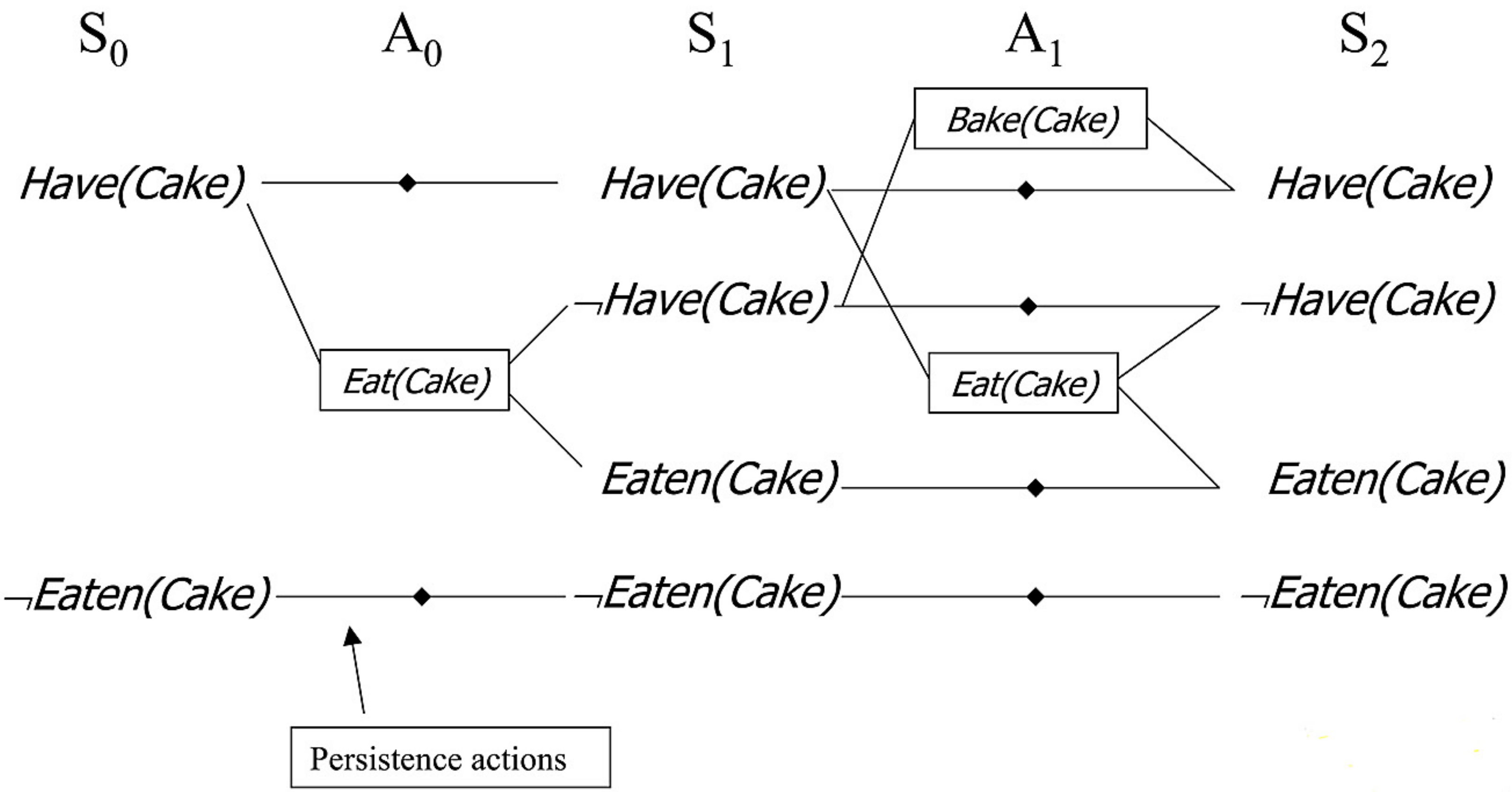
$Init(Have(Cake))$

$Goal(Have(Cake) \wedge Eaten(Cake))$

$Action(Eat(Cake)$
  $PRECOND: Have(Cake)$
  $EFFECT: \neg Have(Cake) \wedge Eaten(Cake)\ )$

$Action(Bake(Cake)$
  $PRECOND: \neg Have(Cake)$
  $EFFECT: Have(Cake)$

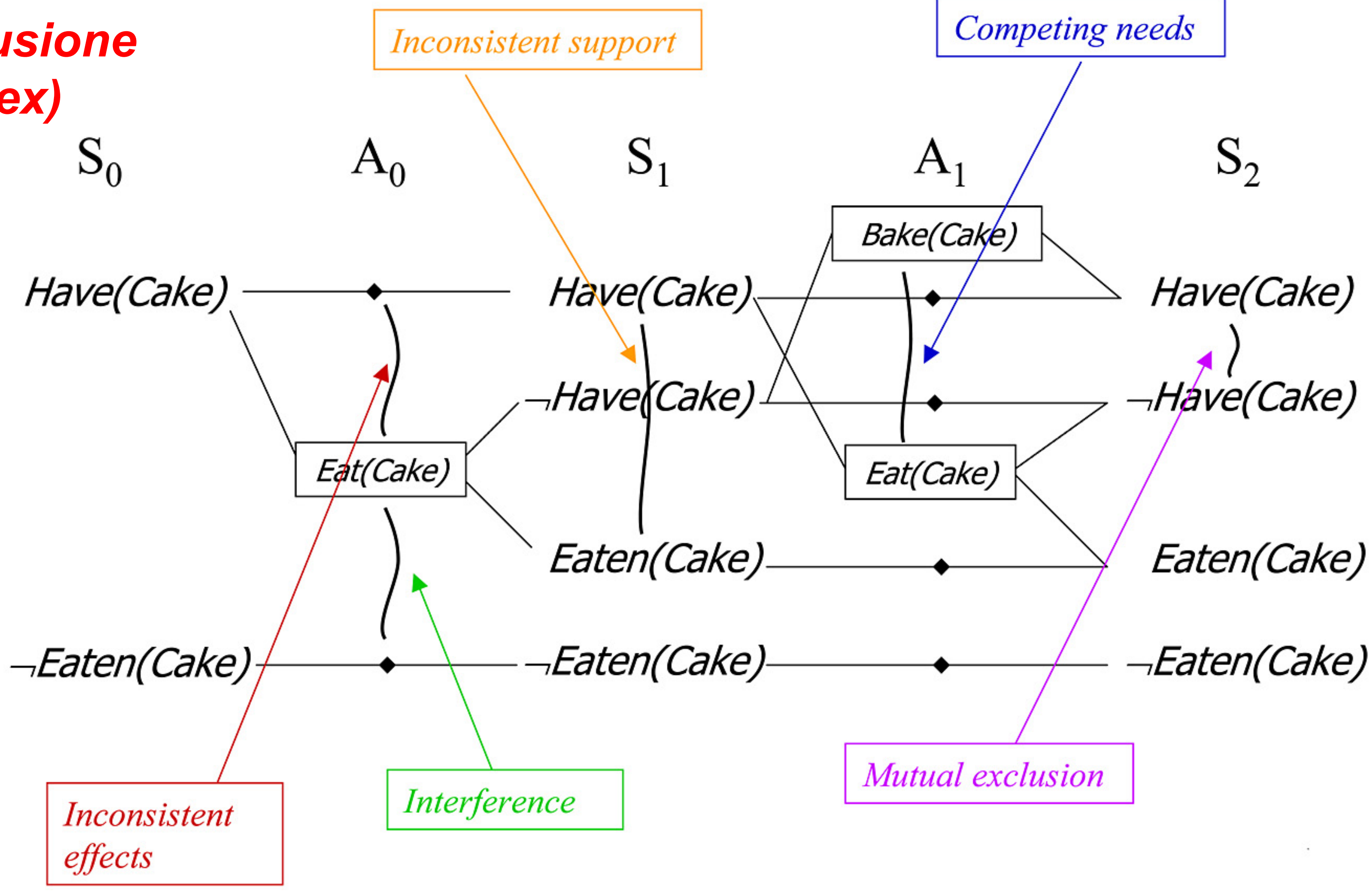
Grafi di Planning



L'espansione del grafo termina quando → livello i+1 = livello i

Grafi di Planning

<u>Conflitti</u>:
link di *mutua esclusione (mutex)*



Grafi di Planning

- *Inconsistent effect*: una azione nega l’effetto di un’altra
- *Interference*: uno degli effetti di una azione è la negazione di una precondizione di un’altra
- *Competing needs*: una delle precondizioni di una azione è la negazione di una delle precondizioni di un’altra
- *Inconsistent support*: letterali allo stesso livello sono in conflitto se uno è la negazione dell’altro o se ogni possibile coppia di azioni che potrebbe raggiungere i due letterali è mutuamente esclusiva

Grafi di Planning

- Ogni livello contiene:
  - tutti i letterali che ***potrebbero*** essere veri in quel passo, dipendentemente dalle azioni eseguite
  - tutte le azioni che ***potrebbero*** avere le precondizioni soddisfatte in quel passo
- Sono trascurate tutte le possibili interazioni negative fra azioni e letterali
- Un letterale che appare per la prima volta a livello *n* non implica l'esistenza di un piano in *n* passi che lo raggiunge…

  … però sicuramente non esiste un piano con meno di *n* passi che lo raggiunge !

Costruzione in tempo polinomiale con grado basso

Grafi di Planning

- Grafo di planning usato per costruire euristiche ammissibili: h(s), **distanza tra lo stato s e il goal**
  - Un letterale che non compare nel grafo di planning implica la non esistenza di un piano, $h(s) = +\infty$
  - *level cost* di un letterale: primo livello in cui compare
  - Stima migliore di *level cost* se si usa grafo di planning seriale (usa mutua esclusione fra coppie di azioni (azioni persistenti escluse): una sola azione alla volta)

- Max-level: massimo livello fra tutti i sottogoal (ammissibile)
- Level sum:somma i livelli dei sottogoal (inammissibile)
- Set-level: livello dove tutti i sottogoal appaiono e nessuna coppia di sottogoal è in mutua esclusione (ammissibile e buono!)

Graphplan

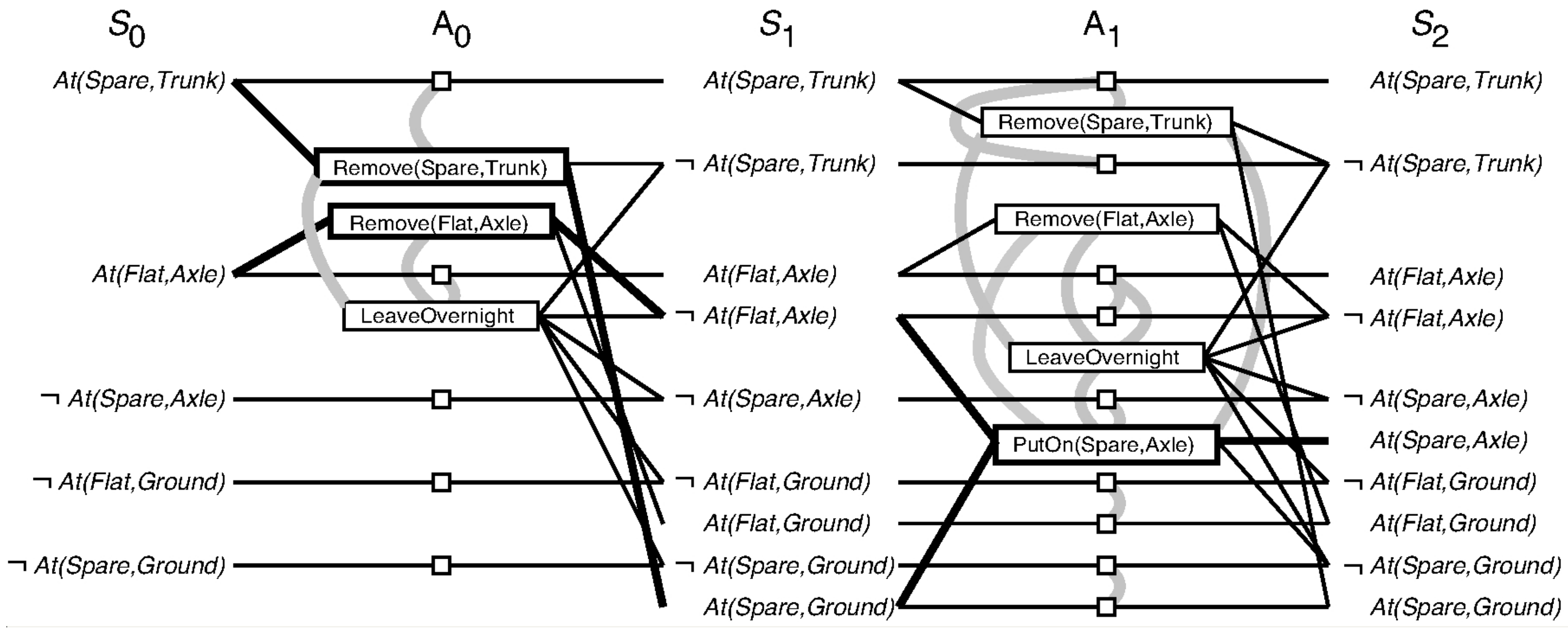
function GRAPHPLAN(*problema*) **returns** una soluzione, o il fallimento

    grafo ← GRAFO-PIANIFICAZIONE-INIZIALE(*problema*)
    obiettivi ← CONGIUNTI(*problema*.OBIETTIVO)
    nogood ← una tabella hash vuota
    for *tl* = 0 **to** ∞ **do**
        if *obiettivi* sono tutti non-mutex in S_t di *grafo* **then**
            soluzione ← ESTRAI-SOLUZIONE(*grafo*, *obiettivi*, NUMLIVELLI(*grafo*), *nogood*)
            if *soluzione* ≠ *fallimento* **then return** *soluzione*
        if *grafo* e *nogood* si sono livellati entrambi **then return** *fallimento*
        grafo ← ESPANDI-GRAFO(*grafo*, *problema*)

Esempio

$Init(Gomma(Bucata) \wedge Gomma(Scorta) \wedge Posizione(Bucata, Asse) \wedge Posizione(Scorta, Bagagliaio))$
$Obiettivo(Posizione(Scorta, Asse))$
$Azione(Rimuovi(ogg, pos)$
PRECOND: $Posizione(ogg, pos)$
EFFETTO : $\neg Posizione(ogg, pos) \wedge Posizione(ogg, Terreno))$
$Azione(Monta(t, Asse)$
PRECOND: $Gomma(t) \wedge Posizione(t, Terreno) \wedge \neg Posizione(Bucata, Asse)$
EFFETTO: $\neg Posizione(t, Terreno) \wedge Posizione(t, Asse))$
$Azione(AbbandonaDiNotte,$
PRECOND:
EFFETTO: $\neg Posizione(Scorta, Terreno) \wedge \neg Posizione(Scorta, Asse)$
$\wedge \neg Posizione(Scorta, Bagagliaio) \wedge \neg Posizione(Bucata, Terreno)$
$\wedge \neg Posizione(Bucata, Asse) \wedge \neg Posizione(Bucata, Bagagliaio))$

Esempio



S0
A0
S1
A1
S2
At(Spare,Trunk)
At(Flat,Axle)
¬ At(Spare,Axle)
¬ At(Flat,Ground)
¬ At(Spare,Ground)
Remove(Spare,Trunk)
Remove(Flat,Axle)
LeaveOvernight
¬ At(Spare,Trunk)
¬ At(Flat,Axle)
At(Flat,Ground)
At(Spare,Ground)
PutOn(Spare,Axle)
At(Spare,Axle)