

A_K

Pianificazione condizionale con A_K

Matteo Valdina

08/03/2008

INTRODUZIONE

In questo breve documento voglio spiegare la semantica e la sintassi del linguaggio A_k . Questo linguaggio descrive una sequenza di azioni per raggiungere un goal, un obiettivo. Prima di descrivere questo linguaggio, vorrei guardare come ragioniamo.

Noi esseri umani siamo abituati a usare, per scoprire cosa ci sta intorno i quattro sensi, olfatto, vista, tatto e gusto. Nel nostro cervello non sono presenti tutte le informazioni del mondo che ci circonda, le raccogliamo con i sensi. Distinguiamo un insieme di "cose" che conosciamo e "cose" che non conosciamo; e se vogliamo estendere quello che conosciamo dobbiamo usare almeno uno dei quattro sensi.

Ammettere che quelle determinate "cose" sono sconosciute, obbliga a considerarle nei ragionamenti. Esempio, non so se il bicchiere è pieno, quindi ho due casi:

1. Bicchiere pieno
2. Bicchiere vuoto

Nel primo caso per bere, l'azione sarà direttamente quella di bere. Nel secondo caso sarà riempire il bicchiere, bere.

A_k descrive dei piani per raggiungere un obiettivo in modo non molto differente. Distingue tra quello che è noto nel mondo, e l'agente non conosce, e quello che l'agente conosce. Prevede delle azioni di senso e deve gestire l'eventualità di un caso o del altro.

A_k è un linguaggio di pianificazione condizionale che possiede delle azioni di senso e distingue tra quello che è lo stato del mondo e quello che è lo stato della conoscenza dell'agente.

Per rendere più chiara la presentazione del linguaggio A_k , applichiamo questo linguaggio ad un esempio intitolato AWE.

AWE - AUTOMATIC WINDOWS EXPOSÉ

Quando si lavora in un sistema multitasking e si lavora contemporaneamente con più finestre, si ha spesso la necessità di cambiare rapidamente da una finestra all'altra. Per venire incontro all'utente vogliamo scrivere un programma che permetta di offrire in un attimo uno sguardo su tutte le finestre.

Usiamo come ambiente di riferimento quello di Windows e consideriamo che questo programma non sostituisce la shell di Windows (la shell di Windows è explorer.exe e si occupa della gestione delle finestre) ma sfrutti le API di Windows per identificare e spostare le finestre. Per mantenere semplice l'esempio non è trattato il caso d'insufficienza di spazio.

La parte intelligente del programma può essere modellata con un agente che esegue piani condizionali, nel caso specifico modelleremo il problema con A_k .

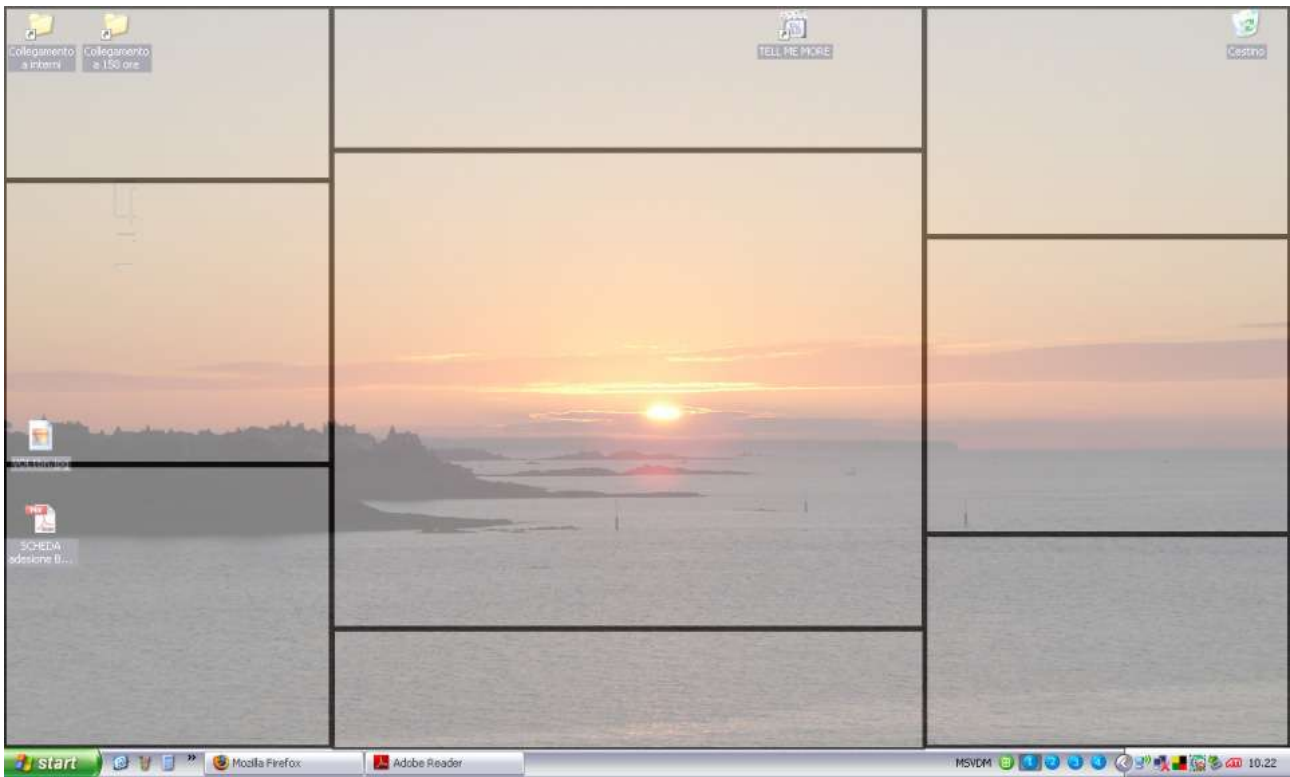
Il nostro agente dispone delle seguenti azioni di senso:

- Rilevare posizione della finestra.

Inoltre può fare le seguenti azioni:

- Spostare la finestra in una posizione libera.

Per semplificare la trattazione di questo esempio si suddivide l'area di lavoro in nove parti, come si può vedere nell'immagine:



Le finestre possono essere spostate unicamente in una di queste aree.

SINTASSI DI A_k

La sintassi di A_k si basa su due insiemi disgiunti di simboli: i *fluents* e le *azioni*. I fluents rappresentano nell'intelligenza artificiale una condizione che può cambiare nel tempo. Le azioni rappresentano delle *azioni* che l'agente può intraprendere nell'esecuzione del piano. Si possono distinguere tra azioni e azioni di senso. E' da notare che le azioni di senso non modificano il mondo.

DESCRIZIONE DI DOMINIO

In questo linguaggio sono definite quattro proposizioni che sono:

- “*value proposition*” abbreviato con vi-proposition.
- “*effect proposition*” abbreviato con ef-proposition.
- “*executable proposition*” abbreviato con ex-proposition.
- “*knowledge proposition*” abbreviato con k-proposition

La descrizione di dominio di A_k è un insieme non contraddittorio di queste quattro tipi di proposizioni.

La “*value proposition*” è indicata con “**initially** f ”, dove f è un fluente; questa proposizione va a indicare che il fluente f è nota inizialmente con valore vero. Due vi-proposition “**initially** f ” e “**initially** g ” sono dette contraddittorie se f è uguale a $\neg g$.

La “*effect proposition*” è indicata con “**a cause** f if p_1, \dots, p_n ”, dove f è chiamato effetto della ef-proposition ed è un fluente; “ p_1, \dots, p_n ” con $n \geq 0$ sono definite precondizioni e sono dei fluents; a è un'azione. Intuitivamente dopo l'esecuzione di a , il fluente f diventa vero in tutti gli stati del mondo dove siano vere le precondizioni. Due ef-proposition con le seguenti precondizioni “ p_1, \dots, p_n ” e “ q_1, \dots, q_n ” sono dette contraddittorie se descrivono l'effetto della stessa azione a su fluents complementari e $\{p_1, \dots, p_n\} \cap \{\bar{q}_1, \dots, \bar{q}_n\} = \emptyset$.

La “*executable proposition*” è indicata con “**executable** a if p_1, \dots, p_n ”, dove a è un'azione e “ p_1, \dots, p_n ” con $n \geq 0$ sono definite precondizioni e sono dei fluents. Questa proposizione indica che l'azione a è eseguibile in tutti gli stati del mondo dove le precondizioni sono vere.

La “*knowledge proposition*” è indicata con “**a determines** p ”, dove a è un'azione e p è un fluente. Il significato di questa proposizione è il seguente: se viene eseguita l'azione a allora nella situazione risultante si conosce il vero valore di p e il valore di p diventa vero.

INTERROGAZIONI IN A_k

In questo linguaggio sono definiti due tipi d'interrogazioni:

- **knows** f after c
- **whether** f after c

Il primo tipo d'interrogazione "**knows f after c** " chiede nella descrizione di dominio se dopo l'esecuzione del piano condizionale c è vero il fluente f rispetto alla situazione iniziale. Questa interrogazione non permette di determinare a priori se è falsa, ma solo se f è vera.

Il secondo tipo d'interrogazione "**whether f after c** " chiede alla descrizione di dominio se dopo l'esecuzione del piano condizionale c è vero o falso il fluente f rispetto alla situazione iniziale.

SEMANTICA DI A_k

La semantica del linguaggio A_k si basa sulla divisione di tre stati:

- Lo stato del mondo, indicato con il termine *state* e simboleggiato con le lettere latine minuscole, esempio s .
- Lo stato della conoscenza dell'agente che è abbreviato con *k-state* (knowledge state) ed è simboleggiato con le lettere greche maiuscole, esempio Σ .
- I combined-state (c-state), che sono delle coppie *state* e *k-state*.

La semantica della descrizione di dominio è definita in termini di modelli che sono delle coppie che consistono in un c-state iniziale e una funzione di transizione. La funzione di transizione mappa una coppia azione e un c-state in un altro c-state.

Uno *state* è un insieme di fluenti e un *k-state* è un insieme di *state*.

Per un'azione a e uno stato s , se esiste un'ex-proposition **executable** a if p_1, \dots, p_n :

$$E_a^+(s) = \left\{ \begin{array}{l} f \mid f \text{ è un fluente e esiste una ef - proposition} \\ \text{"a cause } f \text{ if } p_1, \dots, p_n" \in D \text{ dove } p_1, \dots, p_n \text{ sono contenuti in } s \end{array} \right\}$$

$$E_a^-(s) = \left\{ \begin{array}{l} f \mid f \text{ è un fluente e esiste una ef - proposition} \\ \text{"a cause } \neg f \text{ if } p_1, \dots, p_n" \in D \text{ dove } p_1, \dots, p_n \text{ sono contenuti in } s \end{array} \right\}$$

E

$$Res(a, s) = s \cup E_a^+ \setminus E_a^-$$

Se a non è eseguibile in s , diciamo che $Res(a, s)$ è indefinita.

$Res(a, s)$ definisce lo stato risultante dell'azione a nello stato s . Fintanto che non permettiamo ef-proposition contraddittorie tra di loro nella descrizione di dominio, per ogni coppia di azioni a e stati s , $E_a^+(s)$ e $E_a^-(s)$ sono due insiemi disgiunti e determinati unicamente, quindi $Res(a, s)$ è una funzione deterministica.

Dopo aver introdotto la nozione di *Res* possiamo definire, la funzione di transizione Φ .

DEFINIZIONE DI FUNZIONE DI TRANSIZIONE Φ

Una funzione Φ la quale mappa una coppia formata da un'azione e un c-state in un c-state è chiamata *funzione di transizione* di D se per ogni c-state $\sigma = \langle s, \Sigma \rangle$ e un'azione a :

1. Se a non è eseguibile in s allora $\Phi(a, \sigma)$ è indefinita, denotata da $\Phi(a, \sigma) = \perp$.
2. Se a è eseguibile in s e a non è un'azione di senso, allora:

$$\Phi(a, \sigma) = \langle Res(a, s), \{s' \mid s' = Res(a, s'') \text{ per ogni } s'' \in \Sigma \text{ tali che } a \text{ è eseguibile in } s''\} \rangle;$$

e

3. Se a è eseguibile in s e a è un'azione di senso le cui k-proposition sono a **determines** f_1, \dots, a **determines** f_m allora:

$$\Phi(a, \sigma) = \left\langle Res(a, s), \left\{ s' \mid s' \in \Sigma \text{ tali che } s \text{ e } s' \text{ sia in accordo su ogni } f_i, (i \leq m), \right. \right. \\ \left. \left. \text{e } a \text{ è eseguibile in } s' \right\} \right\rangle$$

Finché Res è una funzione deterministica allora la seguente proposizione è vera: *Ogni descrizione di dominio D possiede un'unica funzione di transizione ϕ .*

DEFINIZIONE STATO INIZIALE

1. Uno stato s è chiamato *stato iniziale* di una descrizione di dominio D se per ogni vi-proposition della forma "**initially** p " o **initially** $\neg p$ in D , p è vero o falso in s .
2. Un c-state $\langle s_0, \Sigma_0 \rangle$ è un *c-state iniziale* di D se s_0 è uno stato iniziale e Σ_0 è un insieme di stati iniziali di D .

Si dice che lo stato iniziale $\sigma_0 = \langle s_0, \Sigma_0 \rangle$ è completo se Σ_0 è un insieme di tutti gli stati iniziali. La completezza dei c-state iniziali esprime l'assunzione che il nostro agente ha una completa conoscenza di quello che conosce e di quello che non conosce nello stato iniziale. Ci riferiamo a questo come "*assunzione della completa consapevolezza*".

DEFINIZIONE DI MODELLO IN A_K

Un modello di una descrizione di dominio D è una coppia (σ_0, Φ) , dove σ_0 è un c-state iniziale ground e Φ è una funzione di transizione di D . Un modello (σ_0, Φ) è chiamato completo se σ_0 è completo.

Grazie alla funzione di transizione, per definizione si riesce a dire quale c-state si raggiunge dopo l'esecuzione di un'azione nel c-state precedente. Questa definizione è la base per definire una funzione di transizione estesa che è capace di trattare sequenze di azioni.

DEFINIZIONE DI FUNZIONE DI TRANSIZIONE ESTESA

Dati una descrizione di dominio D e una funzione di transizione Φ , la funzione di transizione estesa di D , è denotata da $\hat{\Phi}$, la quale mappa coppie di piani condizionali e c-state in c-state, ed è definita come segue:

1. $\hat{\Phi} = ([], \sigma) = \sigma$
2. Per un'azione a , $\hat{\Phi} = (a, \sigma) = \Phi = (a, \sigma)$
3. Per

$c = \text{Case}$

$$\varphi_1 \rightarrow c_1$$

...

$$\varphi_n \rightarrow c_n$$

Endcase

$$\hat{\Phi}(c, \sigma) = \begin{cases} \hat{\Phi}(c_i, \sigma) & \text{se } \varphi_i \text{ è risaputoessere vera in } \sigma \\ \perp & \text{se nessuno } \varphi_1, \dots, \varphi_n \text{ è risaputoessere vera in } \sigma \end{cases}$$

4. Per $c=c_1;c_2$ dove c_1, c_2 sono piani condizionali, $\hat{\Phi} = (c, \sigma) = \hat{\Phi}(c_2, \hat{\Phi} = (c_1, \sigma))$.
5. $\hat{\Phi} = (c, \perp) = \perp$ Per ogni piano condizionale c .

Un piano condizionale c è eseguibile in un c-state σ se $\hat{\Phi} = (c, \sigma) \neq \perp$.

DEFINIZIONE DELLA RELAZIONE DI IMPLICAZIONE PER I DOMINI DI A_k

Dati una descrizione di dominio D , un piano condizionale c e un fluente φ , diciamo:

1. $D \models A_k$ **knows φ after c** se c è eseguibile in σ_0 e φ è risaputo vero in $\hat{\Phi} = (c, \sigma_0)$ per ogni modello (σ_0, Φ) di D .
2. $D \models A_k$ **whether φ after α** se α è eseguibile in σ_0 e φ è risaputo vero o falso in $\hat{\Phi} = (\alpha, \sigma_0)$ per ogni modello (σ_0, Φ) di D .

USO DI A_k PER LA RISOLUZIONE DI AWE

Riprendiamo il nostro esempio per darne una descrizione con la sintassi di A_k . Nel linguaggio usato si segue la seguente notazione per differenziare le costanti dalle variabili.

- Le variabili sono rappresentate dalle lettere latine minuscole in corsivo, esempio $p, v, w(h)$;
- Le costanti sono rappresentate da numeri per le possibili posizioni, esempio 1, 2, 4; e da numeri in notazione esadecimale, esempio 0x034, 0x045.

Il problema è focalizzato nel muovere delle finestre, le quali sono rappresentate da un *handle*, un puntatore. Le finestre vengono descritte dalla costante:

h

Esempi possibili di questo valore sono 0x024, 0x045.

Le finestre che devono essere spostate sono descritte dal fluente “should be moved” che per brevità viene scritto “sbm”:

$sbm(h)$

Il fluente sbm ha come variabile la finestra

Il nostro agente (il programma) può spostare le finestre in una posizione libera, quindi definiamo un'azione $move(h, p)$ la quale muove una finestra h in una posizione p , che in A_k si scrive:

executable $move(h, p)$ if $free(p)$

La finestra può essere spostata unicamente se il fluente $free(p)$ è vero. Il fluente $free(p)$ è vero quando la posizione p è libera da finestre. Da notare che non ho inserito nella condizione di eseguibilità il fluente che ne indica la necessità di muoverla. Infatti il programma è libero di spostare la finestra anche se non è strettamente necessario.

Il nostro agente è in grado di compiere un azione di senso per comprendere se una posizione p è libera. Questa azione non ha nessuna preconditione:

executable $checkFree(p)$

Il compiere un azione porta dei cambiamenti nello stato del mondo infatti muovere la finestra in una posizione libera provoca due cose:

1. La posizione di destinazione non è più libera
2. Non c'è più la necessità di spostare la finestra

L'azione $move(h, p)$ comporta che il fluente $free(p)$ diventa falso e $sbm(h)$ diventa anche esso falso.

$move(h, p)$ **cause** $\neg free(p)$ if $free(p)$

$\text{move}(h, p) \text{ cause } \neg \text{sbm}(h) \text{ if } \text{free}(p), \text{sbm}(h)$

$\text{move}(h, p) \text{ cause } \neg \text{sbm}(h) \text{ if } \text{free}(p), \neg \text{sbm}(h)$

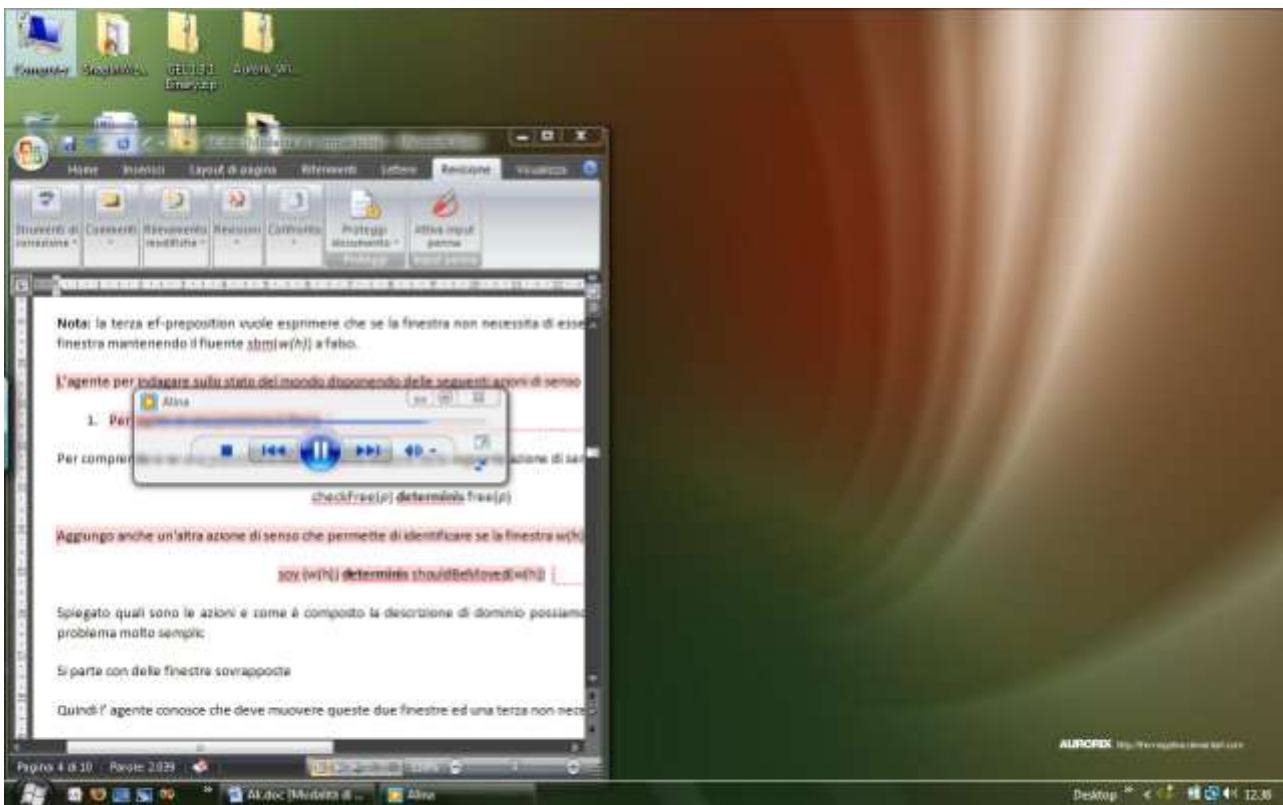
Nota: la terza ef-proposition vuole esprimere che se la finestra non necessita di essere spostata si può spostare la finestra mantenendo il fluente $\text{sbm}(h)$ falso.

L'agente può indagare se una determinata posizione p è libera utilizzando un azione di senso:

$\text{checkFree}(p) \text{ determinis } \text{free}(p)$

Dopo aver spiegato quali sono le azioni e come è composto la descrizione di dominio possiamo un esempio molto semplice.

Si deve spostare una configurazione composta da due finestre una che copre l'altra, come si può vedere nel immagine seguente:



La descrizione di dominio D_1 del problema formulato sarebbe la seguente:

executable $\text{move}(h, p) \text{ if } \text{free}(p)$

executable $\text{checkFree}(p)$

$\text{move}(h, p) \text{ cause } \neg \text{free}(p) \text{ if } \text{free}(p)$

```
move(h, p) cause ¬ sbm(h) if free(p), sbm(h)

move(h, p) cause ¬sbm(h) if free(p), ¬sbm(h)

checkFree(p) determines free(p)
```

L'insieme delle possibili posizioni è {1, 2, 3} e l'insieme delle finestre conosciuto è {0x022, 0x061}, quindi una possibile istanza del problema è la seguente:

```
initially sbm(0x022)

initially ¬sbm(0x061)
```

Il programma costruisce il seguente piano per la finestra 0x022, che per comodità di presentazione è stato suddiviso in 3 piani condizionali (C₁, C₂ e C₃):

```
Piano C1

checkFree(1)

Case

  ¬free(1) → Piano C2

  free(1) → move(0x022, 1)

Endcase

Piano C2

checkFree(2)

Case

  ¬free(2) → Piano C3

  free(2) → move(0x022, 2)

Endcase
```

Piano C₃

checkFree(3)

Case-free(2) → **Il piano è impossibile.**

free(3) → move(0x022, 3)

Endcase

I possibili stati del mondo sono:

 $s_1 = \{\text{sbm}(0x022), \text{free}(1), \text{free}(2), \text{free}(3)\}$ $s_{10} = \{\text{free}(1), \text{free}(2), \text{free}(3)\}$ $s_2 = \{\text{sbm}(0x022), \text{free}(1), \text{free}(2)\}$ $s_{11} = \{\text{free}(1), \text{free}(2)\}$ $s_3 = \{\text{sbm}(0x022), \text{free}(1), \text{free}(3)\}$ $s_{12} = \{\text{free}(1), \text{free}(3)\}$ $s_4 = \{\text{sbm}(0x022), \text{free}(2), \text{free}(3)\}$ $s_{13} = \{\text{free}(2), \text{free}(3)\}$ $s_5 = \{\text{sbm}(0x022), \text{free}(1), \text{free}(3)\}$ $s_{14} = \{\text{free}(1)\}$ $s_6 = \{\text{sbm}(0x022), \text{free}(1)\}$ $s_{15} = \{\text{free}(2)\}$ $s_7 = \{\text{sbm}(0x022), \text{free}(2)\}$ $s_{16} = \{\text{free}(3)\}$ $s_8 = \{\text{sbm}(0x022), \text{free}(3)\}$ $s_{17} = \emptyset$ $s_9 = \{\text{sbm}(0x022)\}$

Quindi dato uno dei possibili c-state:

$$\sigma_1 = \langle s_1, \{s_9\} \rangle$$

Un esecuzione possibile:

$$\hat{\phi}([\text{checkFree}(1)], \sigma_1) = \phi(\text{checkFree}(1), \sigma_1) = \langle s_1, \{s_9, s_6\} \rangle$$

$$\hat{\phi}([\text{checkFree}(1); \text{move}(0x022, 1)], \sigma_1) = \phi(\text{move}(0x022, 1), \langle s_1, \{s_9, s_6\} \rangle) = \langle s_{13}, \{s_{17}\} \rangle$$

L'esecuzione tramite le funzioni di transizione porta semplicemente a passare da uno degli stati possibili della descrizione di dominio ad un altro dei suoi stati.