

Il calcolo delle situazioni (Situation Calculus)

Linguaggio logico progettato per rappresentare mondi che cambiano dinamicamente sotto l'effetto di azioni eseguite dall'agente.

Ipotesi: il mondo evolve in una sequenza discreta e lineare di “situazioni”

Situazioni ed azioni sono considerate **oggetti** e rappresentate da **termini**

REIFICAZIONE di situazioni e azioni

Il linguaggio del calcolo delle situazioni ha due “sorte” (tipi) particolari per i termini che denotano situazioni e i termini che denotano azioni

- **Le azioni sono rappresentate da termini**

forward, turn(right), turn(left), ...

- **Le situazioni sono rappresentate da termini**

- La situazione iniziale è rappresentata da una **costante** S_0
- Le altre situazioni sono rappresentate da termini della forma $do(a, s)$, dove do è un **simbolo funzionale** particolare, che si applica ad azioni e situazioni

Un termine della forma $do(a, s)$ è di tipo situazione: denota la situazione che risulta dall'esecuzione dell'azione a nella situazione s

$do(\text{forward}, S_0)$
 $do(\text{turn}(\text{right}), do(\text{forward}, S_0))$

Fluenti

Ogni predicato che può variare nel tempo (**fluente**) ha un argomento in più: la **situazione**

$$\begin{aligned} &at(agent, stanza_1, S_0) \\ &alive(wumpus, S_0) \\ &at(robot_1, stanza_3, do(forward(robot_1), S_0)) \\ &closed(door_1, do(lock(door_1), S_0)) \end{aligned}$$

Un **fluente** è un predicato la cui verità, per dati argomenti, può variare nel tempo.

Nel calcolo delle situazioni è rappresentato da un simbolo di predicato con un argomento in più, che viene ugualmente chiamato fluente.

Rappresentazione del mondo nel Situation Calculus

Obiettivo: definire una **teoria nel Situation Calculus** (insieme di assiomi) che descriva il mondo e le regole della sua evoluzione

- Descrizione dello stato iniziale
- Descrizione delle azioni, con le condizioni della loro applicabilità e i loro effetti
- Descrizione dell'obiettivo

Pianificazione deduttiva nel situation calculus

Problema di pianificazione: trovare una sequenza di azioni che, se eseguite portano dalla situazione iniziale a una situazione obiettivo

Nel Situation calculus, un problema di pianificazione si risolve dimostrando che

$$\text{Assiomi} \vdash \exists s \text{Goal}(s)$$

Prova costruttiva: fornisce un'istanza S tale che

$$\text{Assiomi} \vdash \text{Goal}(S)$$

Un piano si può sintetizzare come effetto collaterale della dimostrazione di teoremi.

Soluzione: $S = do(a_1, do(a_2, do(a_3, S_0)))$,

Piano Soluzione del problema di planning: $\langle a_3, a_2, a_1 \rangle$.

Esempio: il mondo dei blocchi

Insieme di problemi di pianificazione

Esempio:

stato iniziale:

A
B

C

 obiettivo:

A
B
C

Regole:

- si può spostare un solo blocco alla volta: un blocco X si può mettere sopra un blocco Y o sul tavolo solo se sopra X non c'è alcun blocco
- non si può mettere un blocco sopra se stesso
- non si può mettere un blocco sopra un blocco già occupato
- un blocco libero si può sempre mettere sul tavolo (sul tavolo c'è sempre spazio a sufficienza)

Linguaggio per la rappresentazione del mondo dei blocchi

<i>fluenti(predicati)</i>	<i>azioni(termini)</i>	<i>situazioni(termini)</i>
$on(x, y, s)$	$putOn(x, y)$	S_0
$onTable(x, s)$	$putOnTable(x)$	$do(a, s)$

Rappresentazione del mondo

Descrizione delle proprietà del mondo sempre vere

Proprietà che riguardano “predicati statici”.

Nel caso del mondo dei blocchi, potrebbe essere utile avere assiomi che definiscono l'insieme dei blocchi:

$$block(A), block(B), block(C)$$

O:

$$\forall x (block(x) \equiv x = A \vee x = B \vee x = C)$$

Assieme a un insieme di **assiomi del nome unico** per gli oggetti:

$$A \neq B, A \neq C, B \neq C$$

($t_1 \neq t_2$ è un'abbreviazione di $\neg(t_1 = t_2)$)

Altri esempi:

$$\neg human(pippo)$$

$$\forall x (supermarket(x) \rightarrow sells(x, milk))$$

$$\forall x \forall y \forall z (door(x) \wedge connect(x, y, z) \rightarrow connect(x, z, y))$$

Descrizione della situazione iniziale

Initial State Axiom

Rappresentazione completa di tutto ciò che è inizialmente vero e ciò che è falso

$$\begin{aligned} & on(A, B, S_0), \quad onTable(B, S_0), \quad onTable(C, S_0), \\ & \neg on(A, C, S_0), \quad \neg on(B, A, S_0), \quad \neg on(B, C, S_0), \\ & \neg on(C, A, S_0), \quad \neg on(C, B, S_0), \quad \neg onTable(A, S_0), \\ & \forall x \neg on(x, x, S_0) \end{aligned}$$

O, in alternativa:

$$\begin{aligned} & \forall x \forall y (on(x, y, S_0) \equiv x = A \wedge y = B), \\ & \forall x (onTable(x, S_0) \equiv x = B \vee x = C) \end{aligned}$$

Il secondo insieme di formule è logicamente equivalente al precedente in presenza degli **assiomi del nome unico** per gli oggetti.

Rappresentazione delle azioni

Per ogni azione a si deve rappresentare:

- sotto quali condizioni è possibile eseguire a
- quali sono gli effetti di a (cosa cambia)
- cosa non cambia nel mondo con l'esecuzione di a

Approccio “naive”

Per ogni azione a , con parametri x_1, \dots, x_n , si determina:

- Una formula $precond_a(x_1, \dots, x_n, s)$ che esprime le condizioni di applicabilità di $a(x_1, \dots, x_n)$ nella situazione s :
 $precond_a(x_1, \dots, x_n, s)$ rappresenta il fatto che l'azione $a(x_1, \dots, x_n)$ è eseguibile nella situazione s .
- Una formula $effect_a(x_1, \dots, x_n, s)$ che esprime l'effetto dell'esecuzione $a(x_1, \dots, x_n)$ nella situazione s

e si include nella teoria la descrizione dell'effetto di a :

$$\forall s \forall x_1 \dots \forall x_n (precond_a(x_1, \dots, x_n, s) \rightarrow effect_a(x_1, \dots, x_n, do(a, s)))$$

Esempio:

$$\forall s \forall x \underbrace{(\neg onTable(x, s) \wedge \neg \exists y on(y, x, s))}_{precond_{putOnTable}(x, s)} \rightarrow$$

$$\underbrace{onTable(x, do(putOnTable(x), s)) \wedge \neg \exists y on(x, y, do(putOnTable(x), s))}_{effect_{putOnTable}(x, s)}$$

**Ma si deve anche descrivere che cosa NON CAMBIA:
Frame Axioms**

Per ogni azione, si deve descrivere tutto quello che non cambia quando si esegue l'azione:

$$\forall s \forall x \forall x' (onTable(x, s) \wedge x \neq x' \wedge precondition_{putOnTable}(x', s) \rightarrow onTable(x, do(putOnTable(x'), s)))$$

$$\forall s \forall x \forall y \forall x' (on(x, y, s) \wedge x \neq x' \wedge precondition_{putOnTable}(x', s) \rightarrow on(x, y, do(putOnTable(x'), s)))$$

$$\forall s \forall x \forall x' (\neg onTable(x, s) \wedge x \neq x' \wedge precondition_{putOnTable}(x', s) \rightarrow \neg onTable(x, do(putOnTable(x'), s)))$$

$$\forall s \forall x \forall y \forall x' (\neg on(x, y, s) \wedge x \neq x' \wedge precondition_{putOnTable}(x', s) \rightarrow \neg on(x, y, do(putOnTable(x'), s)))$$

Numero di frame axioms: normalmente molto vicino a $2 \times \mathcal{F} \times \mathcal{A}$
(\mathcal{F} : numero fluenti, \mathcal{A} : numero di azioni)

\Rightarrow **FRAME PROBLEM**

Verso una soluzione del Frame Problem

(Reiter, 1991)

È possibile rendere la descrizione più compatta sfruttando la possibilità di quantificare sulle azioni.

Action precondition axioms

Si introduce un predicato $Poss$:

$Poss(a, s)$ rappresenta il fatto che è possibile eseguire l'azione a nella situazione s

Per ogni azione $a \in \mathcal{A}$, con parametri x_1, \dots, x_n , la teoria contiene un **assioma delle precondizioni** di a :

$$\forall s \forall x_1 \dots \forall x_n (precond_a(x_1, \dots, x_n, s) \rightarrow Poss(a(x_1, \dots, x_n), s))$$

Nel mondo dei blocchi:

$$\forall s \forall x \forall y (\neg on(x, y, s) \wedge x \neq y \wedge \neg \exists z (on(z, x, s) \vee on(z, y, s)) \\ \rightarrow Poss(PutOn(x, y), s))$$

$$\forall s \forall x (\neg onTable(x, s) \wedge \neg \exists y on(y, x, s) \rightarrow Poss(putOnTable(x), s))$$

Successor State Axioms

Per ogni fluente $R(x_1, \dots, x_n, s)$, si determinano le formule:

$G_R^+(a, s, x_1, \dots, x_n)$, che rappresenta tutte le condizioni che possono causare il “passaggio” di $R(x_1, \dots, x_n)$ da falso a vero

$G_R^-(a, s, x_1, \dots, x_n)$, che rappresenta tutte le condizioni che possono causare il “passaggio” di $R(x_1, \dots, x_n)$ da vero a falso

e si include nella teoria l'**Assioma di stato successore** per il fluente $R(x_1, \dots, x_n, s)$:

$$\forall a \forall s \forall x_1 \dots \forall x_n \{ Poss(a, s) \rightarrow \\ [R(x_1, \dots, x_n, do(a, s)) \equiv \\ G_R^+(a, s, x_1, \dots, x_n) \vee (R(x_1, \dots, x_n, s) \wedge \neg G_R^-(a, s, x_1, \dots, x_n))] \}$$

Per ogni azione a , per ogni situazione s e per tutti gli oggetti x_1, x_2, \dots :

$R(x_1, \dots, x_n)$ vale nella situazione $do(a, s)$ se e solo se a è una delle azioni che “provocano” R oppure $R(x_1, \dots, x_n)$ è vero in s e a non lo modifica

$$\boxed{\text{Numero di successor state axioms} = \mathcal{F}}$$

Esempio

Il successor state axiom per il fluente *broken*, in un mondo in cui le azioni possibili sono far cadere un oggetto, far esplodere una bomba e riparare un oggetto, è:

$$\forall a \forall s \forall y \{ Poss(a, s) \rightarrow \{ broken(y, do(a, s)) \equiv \\ [(a = drop(y) \wedge fragile(y)) \vee \exists x (a = explode(x) \wedge nexto(x, y, s)) \\ \vee (broken(y, s) \wedge a \neq repair(y))] \} \}$$

*Per ogni azione a, per ogni situazione s e per ogni oggetto y:
y è rotto nella situazione do(a, s) se e solo se a è un'azione che rompe y (cioè con a si fa cadere y che è fragile, oppure si fa esplodere una bomba vicino a y), oppure y è rotto in s e l'azione a non lo ripara*

Successor State Axioms per il mondo dei blocchi

Condizioni che “modificano” il valore di $on(x, y)$ nel passaggio da s a $do(a, s)$:

$$G_{on(x,y)}^+(a, s, x, y) : a = putOn(x, y)$$

$$G_{on(x,y)}^-(a, s, x, y) : \exists y' a = putOn(x, y') \vee a = putOnTable(x)$$

Condizioni che “modificano” il valore di $onTable(x)$ nel passaggio da s a $do(a, s)$:

$$G_{onTable(x)}^+(a, s, x) : a = putOnTable(x)$$

$$G_{onTable(x)}^-(a, s, x) : \exists y a = putOn(x, y)$$

Assiomi di stato successore:

$$\begin{aligned} &\forall a \forall s \forall x \forall y \{ Poss(a, s) \rightarrow \\ &\quad [on(x, y, do(a, s)) \equiv \\ &\quad \quad a = putOn(x, y) \\ &\quad \quad \vee (on(x, y, s) \wedge \neg(\exists y' a = putOn(x, y') \vee a = putOnTable(x)))] \} \end{aligned}$$

$$\begin{aligned} &\forall a \forall s \forall x \{ Poss(a, s) \rightarrow \\ &\quad [onTable(x, do(a, s)) \equiv \\ &\quad \quad a = putOnTable(x) \\ &\quad \quad \vee (onTable(x, s) \wedge \neg \exists y a = putOn(x, y))] \} \end{aligned}$$

Altri assiomi

Assiomi del dominio finito: non vi sono altri oggetti nel dominio oltre quelli menzionati.

$$\forall x (x = A \vee x = B \vee x = C)$$

Altrimenti, non si potrebbe ad esempio derivare nel nostro esempio:

$$\neg \exists x (on(x, C, S_0))$$

Assiomi del nome unico per le azioni:

$$\forall x \forall y (putOn(x, y) \neq putOnTable(x))$$

$$\forall x \forall y \forall x' \forall y' (putOn(x, y) = putOn(x', y') \rightarrow x = x' \wedge y = y')$$

$$\forall x \forall y (putOnTable(x) = putOnTable(y) \rightarrow x = y)$$

Assiomi del nome unico per le situazioni:

$$\forall a \forall s S_0 \neq do(a, s)$$

$$\forall a \forall s \forall a' \forall s' (do(a, s) = do(a', s') \rightarrow a = a' \wedge s = s')$$

Assioma di induzione, che richiede la logica del secondo ordine (non vi sono altre situazioni se non quelle raggiungibili dalla situazione iniziale eseguendo una sequenza di azioni):

$$\forall P [P(S_0) \wedge \forall a \forall s (P(s) \rightarrow P(do(a, s)))] \forall s P(s)$$

Assiomatizzazione di un dominio

1. Assiomi che descrivono proprietà sempre vere
2. **Initial state axiom:** assioma che descrive lo stato iniziale
3. **Action Precondition axioms:** assiomi che descrivono le precondizioni delle azioni (uno per ogni azione)
4. **Successor State Axioms:** assiomi che descrivono la verità o falsità di un fluente nello “stato successore” (uno per ogni fluente)
5. Assiomi del nome unico (per oggetti, azioni e situazioni)
6. Assiomi del dominio finito
7. Assioma di induzione

Pianificazione deduttiva nel Situation Calculus

Problema di pianificazione: trovare una sequenza di azioni che, se eseguite, portano dalla situazione iniziale a una situazione obiettivo

Nel Situation calculus, un problema di pianificazione si può risolvere dimostrando che

$$\text{Assiomi} \vdash \exists s \text{Goal}(s)$$

Perché dalla dimostrazione si possa estrarre una soluzione al problema di pianificazione, deve essere una **dimostrazione costruttiva**: deve contenere una dimostrazione di

$$\text{Assiomi} \vdash \text{Goal}(S)$$

per qualche termine S di tipo situazione.

Un piano si può allora sintetizzare come effetto collaterale della dimostrazione di teoremi

Esempio: obiettivo $\exists s (\text{nexto}(A, B, s) \wedge \text{onfloor}(A, s))$

Se $S = \text{do}(\text{drop}(A), \text{do}(\text{walk_to}(B), \text{do}(\text{pickup}(A), S_0)))$:

$$\text{Assiomi} \vdash \text{nexto}(A, B, S) \wedge \text{onfloor}(A, S)$$

E con un'applicazione della regola di introduzione di \exists :

$$\text{Assiomi} \vdash \exists s (\text{nexto}(A, B, s) \wedge \text{onfloor}(A, s))$$

Dalla dimostrazione si “estrae” la soluzione rappresentata da S :

il piano è: $[\text{pickup}(A), \text{walk_to}(B), \text{drop}(A)]$

Esempio nel mondo dei blocchi

stato iniziale: $\begin{array}{|c|} \hline A \\ \hline B \\ \hline \end{array}$ $\begin{array}{|c|} \hline C \\ \hline \end{array}$ obiettivo: $\begin{array}{|c|} \hline A \\ \hline B \\ \hline C \\ \hline \end{array}$

$$\text{Assiomi} \vdash \exists s (on(A, B, s) \wedge on(B, C, s) \wedge onTable(C, s))$$

La dimostrazione contiene una dimostrazione di

$$\text{Assiomi} \vdash on(A, B, S) \wedge on(B, C, S) \wedge onTable(C, S)$$

dove

$$S = do(putOn(A, B), do(putOn(B, C), do(putOnTable(A), S_0)))$$

Dalla **soluzione**: S si estrae il **piano**

$$[putOnTable(A), putOn(B, C), putOn(A, B)]$$

Applicazioni

The Cognitive Robotics Group – University of Toronto

GOLOG (alGOl in LOGic): un linguaggio di programmazione logica (di alto livello) per agenti.

Implementato in Prolog.

Il linguaggio è stato utilizzato per costruire sistemi di controllo di robot, ed altre applicazioni.

ConGolog = Concurrent Golog. Incorpora in Golog la concorrenza, interrupts e eventi esogeni. Utilizzato per la progettazione di sistemi di controllo più flessibili, per agenti che operano in scenari complessi.

Presso il DIA: utilizzato in un sistema di controllo reattivo per robot operanti in scenari di salvataggio.

Planning with loops: un pianificatore iterativo (con IF-THEN-ELSE e cicli)

Esercizi

1. Scrivere action precondition axioms (ragionevoli) per le seguenti azioni:
 - (a) andare da x a y
 - (b) comprare x
 - (c) riparare l'oggetto x
 - (d) colorare l'oggetto x con il colore y
 - (e) mettere l'oggetto x dentro la scatola y
 - (f) far cadere l'oggetto x
 - (g) riparare l'oggetto x

2. Scrivere successor state axioms per i seguenti fluenti (senza esagerare nell'immaginare quali azioni possono provocare il cambiamento di valore del fluente!):
 - (a) x si trova in y
 - (b) x ha il colore y
 - (c) il robot e' in direzione y
 - (d) l'oggetto x si trova sopra l'oggetto y
 - (e) la scatola x è vuota
 - (f) la luce della stanza x è accesa
 - (g) la porta della stanza x è aperta
 - (h) il contenuto della scatola x è y (in un mondo in cui è possibile mettere e togliere oggetti dalle scatole).

3. Modificare la rappresentazione del mondo dei blocchi introducendo un nuovo fluente $clear(x, s)$ il cui significato è $\neg\exists y on(y, x, s)$. Utilizzare questo fluente negli action precondition axioms e scrivere il successor state axiom per $clear(x, y)$.
4. Assiomatizzare i problemi seguenti nel calcolo delle situazioni:
- (a) Si hanno a disposizione 3 contenitori, inizialmente vuoti, $c1, c2, c3$, ed una brocca, di capacità superiore a quella di ciascun contenitore. L'obiettivo è quello di riempire i tre contenitori. Le azioni possibili sono: riempire la brocca dal rubinetto, svuotare la brocca nel lavandino, riempire un contenitore mediante travaso dalla brocca (possibile solo se la brocca è piena e il contenitore è vuoto).
- (b) Il robot si trova fuori di una stanza, deve aprire la porta, entrare, prendere il libro che si trova sul tavolo e uscire. Le azioni possibili sono:
- *pass*: può essere eseguita solo se la porta è aperta (*open_door*), ed ha come effetto *in* (essere dentro la stanza), se l'agente si trova fuori, altrimenti (se è già dentro la stanza), l'effetto è $\neg in$.
 - *open_door*: possibile solo se la porta è chiusa, ha come effetto *open* (la porta è aperta);
 - *take_book*: possibile solo se il robot è dentro la stanza e non ha il libro in mano, ha come effetto *have_book*.

- (c) Si hanno due scatole, $s1$ e $s2$, e una penna p . Nella situazione iniziale $s1$ e $s2$ sono entrambe chiuse, $s2$ è sopra $s1$ e p è sopra $s2$. L'obiettivo è quello di avere p dentro $s1$, $s2$ sopra $s1$ e $s1$ e $s2$ entrambe chiuse. Le azioni possibili sono:
- mettere l'oggetto x dentro y (possibile se y è una scatola aperta e x è una penna),
 - aprire x (possibile se x è una scatola sulla quale non è collocato alcun oggetto),
 - chiudere x (possibile se x è una scatola sulla quale non è collocato alcun oggetto),
 - togliere x dalla superficie di y (possibile se x è sopra y e nessun oggetto è sopra x),
 - mettere l'oggetto x sopra y (possibile se y è una scatola e la superficie di x e quella di y sono entrambe libere).
- (d) L'agente è fuori della stanza, e deve entrare, oltrepassando due porte, p_1 e p_2 che sono chiuse. Le possibili posizioni dell'agente sono 3: a_1 (davanti alla porta p_1), a_2 (tra la porta p_1 e la porta p_2), a_3 (dentro la stanza). Alcune proprietà sono sempre vere: il fatto che p_1 collega a_1 e a_2 e il fatto che p_2 collega a_2 e a_3 . Le azioni possibili sono $open(x, y, z)$ (aprire la porta x , che collega y con z , quando si è nella posizione y) e $pass(x, y, z)$ (passare oltre la porta x , dalla posizione y alla posizione z).