Rappresentazione della conoscenza

Lezione 11

Sommario

♢ Pianificazione Deduttiva nel calcolo delle situazioni (Reiter 3.3)

♢ Teoria del calcolo delle situazioni (Reiter 4.4)

♢ Regressione (Reiter 4.5)

La pianificazione deduttiva

Un problema di pianificazione deduttiva nel calcolo delle situazioni è definito da:

$$\mathcal{D} \models \exists s Goal(s)$$

dove $\mathcal{D}$ è la teoria delle azioni che comprende:

- ♢ Assiomi sulla situazione iniziale
- ♢ Assiomi di nome unico per le azioni
- ♢ Assiomi di precondizione
- ♢ Assiomi di stato successore

...

Correttezza e completezza

L'approccio deduttivo alla pianificazione garantisce correttezza e completezza, cioè:

- Se un piano per il $Goal$ esiste esso viene trovato
- Se una sequenza viene dedotta, questa è un piano.

L'indecidibilità del calcolo del primo ordine fa sì che delle deduzioni potrebbero non necessariamente terminare.

Eseguibilità

Con la formulazione data un piano potrebbe includere una azione che non può essere eseguita.

Un piano deve essere *legale*, o **eseguibile**

$$\mathcal{D} \models \exists s(executable(s) \wedge Goal(s))$$

La condizione $executable(s)$ garantisce che per ogni passo del piano vengano verificate le precondizioni per l'esecuzione delle azioni.

Ottimalità

Correttezza e completezza non eliminano la possibilità di dedurre piani che contengono cicli o piani non ottimali.

Per eliminare le sequenze inutili occorre evitare di ripetere certe azioni una volta che un determinato fluente è stato verificato;

Questo può essere fatto inserendo nella assiomatizzazione la descrizione delle cosiddette *bad situations*, cioè situazioni che non si vuole il pianificatore raggiunga, in modo da "potare" l'albero di ricerca ed evitare azioni inutili.

Interrogare un Situation Calculus Database

Esempio: vogliamo sapere se John è iscritto dopo la sequenza di transizioni (database *log*):

$$drop(John, C100), register(Mary, C100)$$

$$Database \models (\exists c).enrolled(John, c, do(register(Mary, C100), do(drop(John, C100), S_0))).$$

Problema della **Proiezione**: il problema di verificare se una sequenza di azioni specificata porta in uno stato in cui vale una data proprietà.

Esempio: precondizioni

$$Poss(pickup(r,x),s) \equiv \\ robot(r) \wedge [(\forall z)\neg holding(r,z,s)] \wedge nexto(r,x,s), \quad (1)$$

$$Poss(walk(r,y),s) \equiv robot(r), \quad (2)$$

$$Poss(drop(r,x),s) \equiv robot(r) \wedge holding(r,x,s). \quad (3)$$

Effetti

$holding(r, x, do(pickup(r, x), s)),$
$\neg holding(r, x, do(drop(r, x), s)),$
$nexto(r, y, do(walk(r, y), s)),$
$nexto(r, y, s) \supset nexto(x, y, do(drop(r, x), s)),$
$y \neq x \supset \neg nexto(r, x, do(walk(r, y), s)),$
$onfloor(x, do(drop(r, x), s)),$
$\neg onfloor(x, do(pickup(r, x), s)).$

Assiomi di stato successore

$$holding(r, x, do(a, s)) \equiv$$
$$a = pickup(r, x) \vee holding(r, x, s) \wedge a \neq drop(r, x), \quad (4)$$

$$nexto(x, y, do(a, s)) \equiv$$
$$a = walk(x, y) \vee (\exists r)[nexto(r, y, s) \wedge a = drop(r, x)] \vee \quad (5)$$
$$nexto(x, y, s) \wedge \neg(\exists z)[a = walk(x, z) \wedge z \neq y],$$

$$onfloor(x, do(a, s)) \equiv$$
$$(\exists r)a = drop(r, x) \vee onfloor(x, s) \wedge \neg(\exists r)a = pickup(r, x). \quad (6)$$

Situazione Iniziale

Inizialmente il robot R è vicino all'oggetto A; R non afferra oggetti.

$$chair(C), robot(R), nexto(R, A, S_0), (\forall z) \neg holding(R, z, S_0). \quad (7)$$

Assiomi di nome unico per le azioni

$pickup(r, x) \neq drop(r', y)$,

$pickup(r, x) \neq walk(r', y)$,

$walk(r, y) = walk(r', y') \supset r = r' \wedge y = y'$,

$etc.$

Fatti derivabili dalla teoria

Da (4)

$$holding(R, A, do(pickup(R, A), S_0)). \quad (8)$$

Da (8), (4) e dagli assiomi di nome unico per le azioni,

$$holding(R, A, do(walk(R, y), do(pickup(R, A), S_0))). \quad (9)$$

Da (9) e (5),

$$nexto(A, y, do(drop(R, A), do(walk(R, y), do(pickup(R, A), S_0)))). \quad (10)$$

Da (6) e (9),

$$onfloor(A, do(drop(R, A), do(walk(R, y), do(pickup(R, A), S_0)))). \quad (11)$$

Pianificazione

Supponiamo di voler derivare che si può raggiungere una situazione in cui A è vicino a B e A è sul pavimento:

$(\exists s).nexto(A, B, s) \wedge onfloor(A, s).$

Prima abbiamo visto una prova costruttiva di:

$s = do(drop(R, A), do(walk(R, B), do(pickup(R, A), S_0))).$

Questa è un *piano* per portare A sul pavimento vicino a B: R afferra A, va in B, lascia A.

Piani non eseguibili

La prova di (10) e (11), non ha usato le precondizioni. Consideriamo:

$$nexto(A, y, do(drop(C, A), do(walk(C, y), do(pickup(C, A), S_0)))),$$
$$onfloor(A, do(drop(C, A), do(walk(C, y), do(pickup(C, A), S_0)))).$$

Anche queste dimostrano: $(\exists s).nexto(A, B, s) \wedge onfloor(A, s)$,

ed il piano relativo è:

$$s = do(drop(C, A), do(walk(C, B), do(pickup(C, A), S_0))).$$

Cioè la sedia C afferra A ...

Eseguibilità del piano

Il piano è *eseguibile* se:

$$Poss(pickup(R, A), S_0) \wedge Poss(walk(R, B), do(pickup(R, A), S_0)) \wedge$$

$$Poss(drop(R, A), do(walk(R, B), do(pickup(R, A), S_0))).$$

In generale:

$$executable(s) \stackrel{def}{=} (\forall a, s^*).do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)$$

I programmi come piani: il calcolo del fattoriale

Un piano è la sequenza di operazioni da eseguire affinché le postcondizioni siano soddisfatte.

Es.: Dato un numero n, $n \geq 0$, si vuole derivare la sequenza di operazioni necessarie per il calcolo del fattoriale di n. Il goal è:

$$\exists s fatt(n, s) = n!$$

Implementazione in PROLOG

- Precondizioni ed assiomi di stato successore sono equivalenze.
- La loro parte-if si può implementare in Prolog con la semantica del completamento di Clark;
- Alle formule che non sono direttamente rappresentabili in clausole di Horn con negazione come fallimento si applicano le trasformazioni Lloyd-Topor

Sotto opportune ipotesi l'implementazione in Prolog si dimostra corretta per una classe di interrogazioni che comprende la costruzione di piani `G(s)`.

Limitazioni dell'approccio

♢ Corretto, ma non completo

♢ Richiede l'ipotesi di nome unico sui termini

♢ Richiede una rappresentazione di mondo chiuso nello stato iniziale:

- no disgiunzione
- no esistenziali

La teoria dei numeri

La teoria del calcolo delle situazioni si basa sulla teoria dei numeri.

$s1.$ $\neg s < 0$,
$s2.$ $\sigma(n_1) = \sigma(n_2) \leftrightarrow n_1 = n_2$
$s3.$ $n < \sigma(n') \leftrightarrow n \leq n'$
$s4.$ $P(0) \wedge (\forall n)[P(n) \rightarrow P(\sigma(n))] \rightarrow (\forall n)P(n)$

La teoria del calcolo delle situazioni

Assiomatizzazione per le situazioni Σ:

$s1.$ $\neg s < S_0$,
$s2.$ $do(a_1, s_1) = do(a_2, s_2) \leftrightarrow a_1 = a_2 \wedge s_1 = s_2$
$s3.$ $s < do(a, s') \leftrightarrow s \leq s'$
$s4.$ $P(S_0) \wedge (\forall a, s)[P(s) \rightarrow P(do(a, s))] \rightarrow (\forall s)P(s)$

Nota: situazione $\neq$ stato

La teoria del calcolo delle situazioni

Il modello di Σ è:

$$\mathcal{GS} = (\mathcal{A} \dot{\cup} \mathcal{S}, \prec, cons, [\,])$$

Dove $\dot{\cup}$ è l'unione disgiunta; $\mathcal{A}$ è il dominio della sorte $azione$, i cui elementi saranno indicati con α; $\mathcal{S}$ è il dominio della sorte $situazione$, ed è definito come l'insieme induttivo che soddisfa:

$[\,] \in \mathcal{S}$
$\sigma \in \mathcal{S}$ implica per ogni $\alpha \in \mathcal{A}, cons(\alpha, \sigma) \in \mathcal{S}$

dove $[\,]$, è l'interpretazione di S_0; $cons$ interpreta la funzione do ed è definito induttivamente come segue:

$cons(\alpha, [\,]) = [\alpha]$
$cons(\alpha, cons(\alpha', \sigma)) = cons(\alpha, [\alpha', \sigma])$

La teoria dinamica del dominio

Il comportamento dinamico del dominio è definito da:

$\mathcal{D} = \Sigma \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss}$ Dove

Σ	è l'insieme degli assiomi fondazionali;
$\mathcal{D}_{una}$	è un insieme di assiomi del tipo $a \neq a'$ per tutte le azioni;
$\mathcal{D}_{S_0}$	è l'insieme di assiomi che descrivono lo stato iniziale;
$\mathcal{D}_{ap}$	è l'insieme di assiomi della forma: $Poss(a(x_1, \ldots, x_n), s) \leftrightarrow \Pi_a((x_1, \ldots, x_n), s)$ Π_a è una formula con variabili libere al più $(x_1, \ldots, x_n, s)$ denota che l'azione a è possibile nella situazione s;
$\mathcal{D}_{ss}$	è l'insieme di assiomi della forma: $F(x_1, \ldots, x_n, do(a, s)) \leftrightarrow \Psi_F((x_1, \ldots, x_n), a, s)$ Ψ_F è una formula con variabili libere al più $(x_1, \ldots, x_n, a, s)$ denota quando il fluente F è vero nella situazione $do(a, s)$.

Condizioni aggiuntive

Le formule che definiscono ap e ss devono essere **uniformi** in s:

- non menzionare $Poss$, uguaglianza e disuguaglianza di situazioni
- non quantificare sulle situazioni
- s deve essere l'unico termine situazione che contengono

Questa condizione sintattica corrisponde alla ipotesi di **Markov**: ap e ss dipendono solo dalla situazione precedente.

La regressione

La *Regressione* è un metodo per verificare la derivabilità di un piano:

$$\mathcal{D} \models Goal(do(a_k, \ldots, do(a_1, S_0)))$$

se e solo se

$$\mathcal{D}_{S_0} \models \mathcal{R}(Goal(do(a_k, \ldots, do(a_1, S_0))))$$

dove $\mathcal{R}$ è un operatore che, data una formula **regredibile**, restituisce una formula in cui l'unico termine di tipo situazione che vi occorre è la costante S_0.

Formule "regredibili"

Una formula si definisce "regredibile" sse

- ciascun termine relativo ad una situazione si riconduce ad S_0.
- le azioni in $Poss(\alpha, \sigma)$ sono istanziate $A(t_1, \ldots, t_n)$
- non ci sono quantificazioni sulle situazioni
- non ci sono uguaglianze e disuguaglianze tra situazioni

Operatore di regressione: forma semplificata

1. Se W è un atomo, si hanno 4 casi:

   (a) W è un atomo indipendente dalla situazione (es. uguaglianza tra due termini della stessa sorte *object* o *action*):
   $$\mathcal{R}[W] = W.$$
   (b) W è un fluente relazionale della forma $F(\vec{t}, S_0)$:
   $$\mathcal{R}[W] = W.$$
   (c) W è un atomo $Poss(A(\vec{t}), \sigma)$ per i termini $A(\vec{t})$ e σ di sorte *action* e *situation*. $A \in \mathcal{L}_{sitcalc}$. con precondizione del tipo $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$:
   $$\mathcal{R}[W] = \mathcal{R}[\Pi_A(\vec{t}, \sigma)].$$
   Cioè, sostituisce l'atomo $Poss(A(\vec{t}), \sigma)$ con un'opportuna istanza della parte destra dell'assioma di precondizione di A.

(d) W è un fluente relazionale $F(\vec{t}, do(\alpha, \sigma))$. Sia l'assioma di stato successore di F in $\mathcal{D}_{ss}$ $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$:

$$\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)].$$

Cioè, sostituisce l'atomo $F(\vec{t}, do(\alpha, \sigma))$ con la parte destra dell'assioma di stato successore di F

2. per le formule non atomiche, la regressione è definita induttivamente.

$$\mathcal{R}[\neg W] = \neg\mathcal{R}[W],$$

$$\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2],$$

$$\mathcal{R}[(\exists v)W] = (\exists v)\mathcal{R}[W].$$

Regressione

L'operatore di regressione elimina gli atomi $Poss$ con le definizioni delle precondizioni, e sostituisce i fluenti $do(\alpha, \sigma)$ con le espressioni equivalenti σ date dagli assiomi di stato successore.

La formula ottenuta contiene solo riferimenti alla situazione S_0.

Th. Se W è una formula regredibile di $\mathcal{L}_{sitcalc}$ che non contiene fluenti funzionali, e $\mathcal{D}$ è una teoria di azioni. Quindi $\mathcal{R}[W]$ è una frase uniforme in S_0 e:

$$\mathcal{D} \models W \equiv \mathcal{R}[W],$$

Teorema della regressione

Poiché W e $\mathcal{R}[W]$ sono logicamente equivalenti, per dimostrare W dalla teoria $\mathcal{D}$, è sufficiente dimostrare $\mathcal{R}[W]$.

$\mathcal{R}[W]$ fa riferimento solo al termine S_0, quindi (Th.)

$$\mathcal{D} \models W \quad \text{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W].$$

La valutazione di una formula regredibile si riconduce ad una dimostrazione al primo ordine nella *teoria iniziale* $\mathcal{D}_{S_0}$

Esempi

$\mathcal{R}[Poss(drop(Bill, C100), S_0)] = enrolled(Bill, C100, S_0).$

$\mathcal{R}[(\exists p) Poss(register(p, C100), S_0)] =$
$(\exists p)(\forall p').prerequ(p', C100) \supset (\exists g).grade(p, p', g, S_0) \wedge g \geq 50.$

$\mathcal{R}[Poss(drop(Bill, C100), do(register(p, c), S_0))] =$
$\mathcal{R}[enrolled(Bill, C100, do(register(p, c), S_0))] =$
$\mathcal{R}[register(Bill, C100) = register(p, c) \vee$
$enrolled(Bill, C100, S_0) \wedge register(p, c) \neq drop(Bill, C100)] =$
$register(Bill, C100) = register(p, c) \vee$
$enrolled(Bill, C100, S_0) \wedge register(p, c) \neq drop(Bill, C100)$

Semplificando:

$Bill = p \wedge C100 = c \vee enrolled(Bill, C100, S_0).$

$$\mathcal{R}[enrolled(Sue, c, do(drop(Bill, c), do(register(p, c), S_0)))] =$$
$$\mathcal{R}[drop(Bill, c) = register(Sue, c) \vee$$
$$\quad enrolled(Sue, c, do(register(p, c), S_0)) \wedge drop(Bill, c) \neq drop(Sue, c)] =$$
$$\mathcal{R}[enrolled(Sue, c, do(register(p, c), S_0))] \wedge Bill \neq Sue] =$$
$$\mathcal{R}[\{register(p, c) = register(Sue, c) \vee$$
$$\quad enrolled(Sue, c, S_0) \wedge register(p, c) \neq drop(Sue, c)\} \wedge Bill \neq Sue] =$$
$$\{p = Sue \vee enrolled(Sue, c, S_0)\} \wedge Bill \neq Sue$$

Usi della regressione

♢ verifica di **eseguibilità**

$\mathcal{D} \models executable(do([a_1, \ldots, a_n], S_0))$ sse $\mathcal{R}[Poss(\alpha_i, do([\alpha_1, \ldots, \alpha_{i-1}, S_0))]$ in congiunzione per ogni $i \in 1, \ldots, n$

♢ **proiezione**

$\mathcal{D} \models G(do([a_1, \ldots, a_n], S_0))$

Pianificazione

- pianificazione deduttiva (semplice linguaggio per esprimere piani)
- verifica/monitoraggio di piani (linguaggio di pianificazione espressivo)

- verifica delle transizioni di stato
- ricerca della soluzione