

Ciao a tutti. Vi lascio i miei appunti di Agenti Intelligenti - parte di Alberto Martelli.

Punti FONDAMENTALI:

1. potrebbero (e ci sono) esserci errori grammaticali, una *e* non accentata, una *p* di troppo e robe del genere... Ma ne rendo SEMPRE conto una volta che studio;
2. questi appunti non sostituiscono le lezioni e soprattutto sono personali.

Vi voglio bene. Un abbraccio dal vostro **Cirino Pomicino**.

Contents

1 Logica e Agenti	2
1.1 La logica	2
1.1.1 Ruoli della logica per gli agenti	2
1.1.2 Logica classica	3
1.1.3 Logica modale	3
1.1.4 Logica (classica) proposizionale	4
2 Logica Modale e Agenti	6
2.1 La logica modale	6
2.1.1 Logica modale proposizionale: sintassi	6
2.1.2 Logica modale proposizionale: semantica	7
2.1.3 Definizione formale della semantica della logica modale	7
2.1.4 Piccolo esempio	8
2.1.5 Proprietà della logica modale	9
2.2 Logica modale per gli Agenti	9
2.2.1 Principali logiche modali per agenti	10
2.2.2 Proprietà dei frame	11
2.2.3 Corrispondenza	11
2.2.4 Sistemi modali	12
2.2.5 Proof theory	12
3 Logica Epistemica	13
3.1 Onniscienza Logica	13
3.2 Assiomi per knowledge e belief	14
3.3 Un esempio d'applicazione: Muddy children	15
4 Modellare tempo e azioni	16
4.1 La logica temporale	16
4.1.1 Linear Temporal Logic (LTL)	16
4.1.2 Computation Tree Logic (CTL*)	18
4.1.3 Da CTL* a CTL	19
4.2 Ragionare sulle azioni	19
4.2.1 Azioni: precondizioni e effetti	20
4.2.2 Frame problem	20
4.2.3 Successor State Axioms	20
4.2.4 GOLOG	21
5 Modellazione logica di agenti BDI	22
5.1 La logica intenzionale di Cohen e Levesque	22
5.1.1 Proprietà delle intenzioni	22
5.1.2 Formulazione della logica	23
5.1.3 La semantica	23
5.1.4 Modalità derivate	24

5.1.5	Achievement Goals	24
5.1.6	Goal Persistenti	25
5.1.7	Intenzione di eseguire un'azione	25
5.2	Rao and Georgeff's BDI logic	26
5.2.1	Semantica Informale	26
5.2.2	Il linguaggio	26
5.2.3	Commitments	27
6	Model Checking	29
6.1	Il Model Checking	30
6.1.1	Model checking per LTL	30
6.1.2	LTL e automi su stringhe infinite	31
6.1.3	Procedura per il model checking	32
6.1.4	Conclusioni	33
6.2	Model Checkin in SPIN	33
6.2.1	Model Checker in CTL	34
7	Agenti Ibridi e Reattivi	35
7.1	Architetture Reattive	35
7.1.1	L'approccio di Brooks	35
7.1.2	Architettura di Brooks	36
7.1.3	Selezione delle azioni	37
7.1.4	Un esempio: Steels' Mars Explorer (1990)	37
7.1.5	Steels' Mars Explorer: cooperation	38
7.1.6	Vantaggi degli agenti reatti	38
7.1.7	Limiti degli Agenti Reattivi	38
7.2	Architetture Ibride	39
7.2.1	Ferguson - TOURINGMACHINES	40
7.2.2	Muller InteRRaP	41
7.3	Conclusioni	41
8	Agenti Ibridi e Reattivi	42
8.1	Architetture Reattive	42
8.1.1	L'approccio di Brooks	42
8.1.2	Architettura di Brooks	43
8.1.3	Selezione delle azioni	44
8.1.4	Un esempio: Steels' Mars Explorer (1990)	44
8.1.5	Steels' Mars Explorer: cooperation	45
8.1.6	Vantaggi degli agenti reatti	45
8.1.7	Limiti degli Agenti Reattivi	45
8.2	Architetture Ibride	46
8.2.1	Ferguson - TOURINGMACHINES	47
8.2.2	Muller InteRRaP	48
8.3	Conclusioni	48
9	Linguaggi e architetture per agenti BDI	49
9.1	Procedural Reasoning System (PRS)	49
9.1.1	I piani del PRS	50
9.1.2	Control loop del PRS	51
9.1.3	AgentSpeak(L)	52
9.2	Agent-oriented programming (AOP)	52
9.2.1	AOP: Categorie mentali	52
9.2.2	AOP: AGENT-0	53
9.2.3	Commitment Rule e Interprete: AGENT-0	54

10 Linguaggi logici per agenti	56
10.1 Linguaggi Logici	56
10.1.1 GOLOG	56
10.1.2 Azioni complesse	57
10.1.3 GOLOG: un elevator controller	57
10.1.4 Come eseguire un programma	58
10.1.5 Estensioni del GOLOG	58
10.2 Concurrent METATEM	58
10.2.1 Operatori utilizzati in METATEM	59
10.2.2 Scambio di messaggi tra agenti	60
11 La teoria dei giochi	62
11.1 La teoria dei giochi	62
11.1.1 Proprietà principali della teoria dei giochi	63
11.1.2 Un esempio: Il gioco Morra a due-dita	63
11.1.3 Un esempio: Dilemma del Prigioniero	64
11.1.4 La strategia dominante	65
11.1.5 L'equilibrio di Nash	66
11.1.6 Un esempio: Costruttore hardware	67
11.1.7 Le strategie miste	68
12 Progettazione di meccanismi	71
12.1 Le aste	72
12.2 L'intradamento dei pacchetti	73

Chapter 1

Logica e Agenti

1.1 La logica

Gli argomenti hanno l'obiettivo di introdurre l'uso della logica. Incominceremo dicendo qualcosa in generale sulla logica per gli agenti, per poi proseguire con la logica classica. Questa in particolare non sarà adeguata per ragionare e operare. In merito, vedremo quelle che saranno le soluzioni da adottare per modificare certi aspetti della logica classica, in modo da essere utilizzata nel nostro contesto. La **logica modale**, è una variante della logica. Tale logica, viene normalmente utilizzata per ragionare sugli agenti. La logica fornisce degli strumenti formali per:

- **rappresentare** la conoscenza, fornisce dei formalismi per modellare certe proprietà degli agenti. La conoscenza viene rappresentata in modo formale utilizzando delle regole/formule della logica. C'è un linguaggio formale che ha una definizione ben precisa, che consente di rappresentare in modo rigoroso la **conoscenza**;
- **ragionare** sulla conoscenza. Una volta che l'agente possiede della conoscenza (es. descrive un certo problema con delle formule della logica), vorrebbe anche utilizzarla. È importante avere un meccanismo che permette di ragionare con questa. Nei corsi dei secoli sono state sviluppate tante regole d'inferenza che consentono appunto di ragionare sulla conoscenza. Presa una conoscenza espressa con una formula della logica, esistono diversi meccanismi che forniscono regole d'inferenza che permettono di ragionare su queste formule.

1.1.1 Ruoli della logica per gli agenti

Adesso vi è un richiamo su quelle che saranno le proprietà della logica degli agenti.

La logica può essere utilizzata per **rappresentare** e **ragionare** sulla conoscenza. Un agente, può utilizzare questa rappresentazione basata sulla logica per ragionare e formularizzare le conoscenze che ha sul mondo esterno. Da ciò, si deduce, come la logica viene vista come strumento importante utilizzato da parte di un agente. Dato che la conoscenza è espressa in un linguaggio formale, gli agenti possono usare metodi formali (inferenza) per derivare altra conoscenza.

N.B. La logica non viene usata solo per agenti che usano la conoscenza e meccanismi di ragionamento. La cosa interessante è spesso notare come la logica viene utilizzata per ragionare sul **comportamento** di un agente, in modo da verificare, se l'agente si comporta nel modo in cui vorremmo noi. Quindi, la logica, verrà utilizzata per verificare se l'agente fa effettivamente quello che dovrebbe fare in base a delle specifiche.

1.1.2 Logica classica

Quando si parla di logica nell'ambito di intelligenza artificiale (IA), si intende la cosiddetta logica classica, ossia la logica proposizionale o la logica del primo ordine. Tuttavia, la necessità di modelare concetti diversi e le esigenze hanno portato l'IA all'uso di logiche diverse da quelle classiche e anche alla definizione di nuove logiche come, ad esempio, le logiche non monotone. Nell'ambito degli agenti e sistemi multiagenti si preferisce utilizzare una logica non classica nota come **logica modale**

1.1.3 Logica modale

Questo tipo di logica, permette di formulare il contenuto di frasi come *Jhon crede che Superman voli*. Considerando la sola logica classica, potremmo dire che esiste un predicato che permette di definire le credenze di un agente.

$$\text{Bel}(\text{Jhon}, \text{vola}(\text{Superman}))$$

Questo tipo di formulazione non funziona per almeno due motivi.

1. *sintassi*. Le formule della logica classica hanno la seguente struttura.

$$\text{Predicato}(\text{termine}, \dots, \text{termine})$$

Però nell'esempio di prima il secondo argomento non è un termine, ma ben si una formula. Si nota come questa formulazione non viene descritta dalla sintassi appena citata;

2. *semantica*. Gli operatori intenzionali come *Bel*, sono **referentially opaque**, perché creano dei contesti chiusi, per cui non è possibile sostituire una formula con una equivalente, come in logica classica. Supponiamo di avere la stessa formula di prima, dove sia Superman che Clark siano lo stesso individuo e quindi potremmo usare sia uno o l'altro. Questo in realtà non possiamo farlo, quindi non potremmo derivare $\text{Bel}(\text{Jhon}, \text{vola}(\text{Clark}))$ a meno che Jhon non crede che Superman e Clark siano lo stesso individuo.

Per superare queste problematiche utilizzeremo la logica modale. Esistono degli operatori modali, come *Belief* che possono avere delle formule come argomenti. Sono state proposte anche altre soluzioni, ovvero quelle di usare un meta-linguaggio che permette di trasformare termini in formule. Questo permette inoltre di utilizzare la logica classica. Approccio meno diffuso rispetto a quello basato sulla logica modale.

La **logica modale** è partita da Aristotele (me cojoni). Quella attualmente utilizzata risale a definizioni degli anni 60. Definizioni, dovute principalmente a Hintikka e Kripke. Cosa interessante che vedremo nel dettaglio, è che la semantica della logica modale è basata sulla nozione di **mondi possibili**. Ogni mondo rappresenta una situazione considerata possibile dall'agente. Anziché esserci un solo mondo, ci si aspetta più mondi dove ognuno di essi ha le sue proprietà.

Un piccolo esempio

Supponiamo che un agente x stia giocando a carte. Ogni agente conosce le proprie carte ma non conosce le carte degli altri. Considerando l'agente x , ci sarà un mondo di questo agente che include quelle che sono le carte che ha. In aggiunta, lui stesso potrebbe immaginare quelle che sono le carte possedute dagli altri giocatori (agenti) e quindi poter ragionare su questi mondi. In particolare, saprà che se lui ha l'ASSO, gli altri giocatori non lo tengono e quindi x potrà avere delle definizioni/proprietà dei mondi degli altri giocatori che lui può conoscere e

altri che non può conoscere. Da un punto di vista filosofico, la proprietà fondamentale della logica modale è che tutto ciò che è vero in tutti i mondi possibili è **necessariamente vero** e tutto ciò che è vero in qualche mondo diciamo che è **possibile**.

1.1.4 Logica (classica) proposizionale

Definiamo in primis quella che è la **sintassi** della logica proposizionale. Serve per dire come sono fatte le formule del linguaggio. Le **formule** sono costituite da un insieme di simboli (proposizioni atomiche) che appartengono ad un insieme P e da connettivi logici (\wedge, \vee, \neg) secondo la seguente formulazione

- p , proposizione atomica, dove p è un elemento di P ;
- $\varphi \wedge \psi$, sono formule collegate dall'operatore \wedge
- $\neg\varphi$, negazione della formula

N.B. Nella sintassi appaiono solo i connettivi $\neg\wedge$ per minimizzare la dimensione della sintassi, ma è ben noto che da questi connettivi se ne possono ricavare altri come ad esempio l'implicazione: $\varphi \Rightarrow \psi \equiv \neg\varphi \wedge \psi$

Esempio dell'aspirapolvere di Wooldridge

Abbiamo un insieme P che contiene un insieme di proposizioni che in questo caso particolare riguardano l'aspirapolvere e dellosporco.

$$P = \{In_{0,0}, In_{0,1}, \dots, Dirt_{0,0}, Dirt_{0,1}, \dots, FacingNorth, \dots, DoForward\dots\}$$

Da questo possiamo scrivere una formula fatta in questo modo.

$$In_{0,0} \vee FacingNorth \vee \neg DIRT_{0,0} \Rightarrow DoForward$$

Se tale formula risulta essere vera $\Rightarrow DoForward$. **FINE ESEMPIO**

La **semantica** permette di definire la verità delle formule in ogni modello. Nel nostro caso, un modello assegna un valore di verità (True oppure False) ad ogni simbolo proposizionale. Se ci sono n simboli, allora ci saranno 2^n modelli, quindi i modelli possono essere tanti ma sono in un numero finito. Questa logica ha il vantaggio rispetto a quella classica di essere **decidibile**. Di solito, un modello viene rappresentato in modo diverso, ovvero come un insieme M (insieme di simboli proposizionale) $\subseteq P$. Tutto quello che c'è in M possiamo assumere che sia vero e quello che non c'è che sia falso. La semantica definisce una relazione di **soddisfacibilità** $M \models \varphi$ di una formula φ in un modello M (interpretazione).

- $M \models p$ sse $p \in M$
- $M \models \varphi \wedge \psi$ sse $M \models \varphi$ or $M \models \psi$
- $M \models \neg\varphi$ sse $M \not\models \varphi$

Una formula è **soddisfacibile** se e solo se c'è **qualche** modello che la soddisfa

Una formula è **valida** se e solo se è soddisfatta da **ogni** modello (**tautologia**). Data una base di conoscenza (insieme di formule) KB , una formula α **segue logicamente** da KB

$$KB \models \alpha$$

se e solo se, in ogni modello in cui KB è vera, anche α lo è. Questa può essere descritta anche secondo il **teorema di deduzione**. Date due formule α, β , $\alpha \models \beta$ se e solo se la formula $(\alpha \Rightarrow \beta)$ è valida.

Modus Pones

Per concludere, sono stati definiti per la logica dei predicati dei meccanismi di inferenza per ragionare con la logica. Fino ad ora non abbiamo parlare di *ragionare*, o meglio l'abbiamo fatto parlando di soli modelli. Se vogliamo sapere se una formula è corretta oppure no, andiamo nel modello e verifichiamo se tutti i suoi componenti sono veri. Sono stati sviluppati comunque dei sistemi deduttivi (dove nella logica classica sono molto più complessi) che permettono di fare inferenza. **Modus Pones** ne è un esempio.

Chapter 2

Logica Modale e Agenti

2.1 La logica modale

Iniziamo ad usare la logica modale per modellare e realizzare agenti. Nel capitolo precedente abbiamo visto dei vincoli, delle formulazioni, che non portavano a modellare/descrivere gli agenti. Abbiamo confrontato gli aspetti della logica classica che servirebbero per modellare gli agenti, senza entrare troppo nel dettaglio. Abbiamo introdotto la **logica proposizionale**, utile come base per andare avanti. Adesso non faremo altro che entrare più nel dettaglio, cercando di capire come utilizzare la logica modale per implementare e usare degli agenti. Daremo una definizione precisa di quella che è la:

1. sintassi e semantica della logica modale;
2. le proprietà della logica modale;
3. corrispondenza di alcune proprietà con altre;
4. capire come poter realizzare un sistema modale, ossia una struttura basata sulla logica modale.

La logica modale è stata sviluppata da filosofi interessati alla distinzione fra **verità necessarie** e **verità contingenti**. Per esempio:

- il fatto che un partito abbia la maggioranza in Senato è un esempio di *verità contingente*, in quanto può avere una maggioranza adesso ma potrebbe non avere la stessa maggioranza nei giorni successivi;
- dare un'affermazione del tipo *la radice quadrata di 2 non è un numero razionale* è **necessariamente vera**.

Quello che affronteremo in questo capitolo sarà sempre rimanere nell'ambito della logica proposizionale e riuscire a vedere come questa può essere definita una logica modale proposizionale.

2.1.1 Logica modale proposizionale: sintassi

La logica modale proposizionale a differenza di quella proposizionale fa uso (in più) di due operatori modali. \Box e \Diamond . Facciamo riferimento ad un insieme P di proposizioni e la sintassi è data da p elemento dell'insieme P e da formule costruite usando i connettivi della logica classica.

- p
- $\varphi \wedge \psi$

- $\neg \varphi$
- $\Box \varphi$
- $\Diamond \varphi$

Le ultime due formule sono costruite tramite due **operatori** modali, operatori che hanno come argomento formule della logica. I due operatori catturano diverse modalità di verità:

- **necessariamente vero**, \Box ;
- **possibilità di essere vero** \Diamond .

I due operatori sono uno il duale dell'altro, ossia:

- $\Box \phi \equiv \neg \Diamond \neg \psi$. Se è necessario che ϕ sia vera, qualunque cosa sia ϕ , non è possibile che ϕ sia falsa.
- $\Diamond \phi \equiv \neg \Box \neg \phi$.

Si noti che, se volessimo minimizzare la sintassi, potremmo lasciare nella sintassi uno solo fra $\Box \phi$ e $\Diamond \phi$ e fare riferimento alle due equivalenze citate appena sopra. In seguito comunque useremo ambedue le formule per fare chiarezza.

2.1.2 Logica modale proposizionale: semantica

Nel capitolo 1 avevamo accennato che la semantica era basata su un insieme di mondi possibili. Quindi, un **modello** della logica modale è dato come un insieme di mondi possibili (sono tanti, nella logica classica era solo uno), dove ogni mondo è costituito da un insieme di proposizioni che sono considerate vere in quel mondo. In generale, i mondi possibili sono in numero qualunque e sono collegati attraverso **relazioni di accessibilità**. Il mondo w_1 è associato al mondo w_2 quindi dal mondo w_1 posso accedere al mondo w_2

Più precisamente un **modello M** è una tripla $\langle W, R, L \rangle$ dove:

- W è un insieme di mondi;
- $R \subseteq W \times W$ è una relazione di accessibilità fra mondi, e
- $L : W \rightarrow 2^P$, da l'insieme di proposizioni vere in ogni mondo $\in W$

Un esempio di modello

Una nota importante da ricorda che una relazione d'accessibilità può mettere in relazione anche un mondo con se stesso. vedere il mondo w_2

2.1.3 Definizione formale della semantica della logica modale

Un **modello M** è una tripla $\langle W, L, R \rangle$ dove W è un insieme di mondi, $R \subseteq W \times W$ è una relazione di accessibilità e, $L : W \rightarrow 2^P$ da l'insieme delle proposizioni vere in un mondo. La **soddisfabilità** di una formula φ è definita rispetto a un modello M e ad un mondo w di questo modello con la notazione $M \models_w \varphi$. In modo formale possiamo dire che:

- $M \models_w p$ iff $p \in L(w)$, p è soddisfatta nel mondo M_w se il simbolo p appartiene a $L(w)$, ovvero se $L(w)$ da l'insieme di tutti i simboli proposizionale che stanno nel mondo w . Quindi, se p sta nel mondo w è soddisfatto;
- $M \models_w \varphi \vee \psi$ iff $M \models_w \varphi$ or $M \models_w \psi$

- \models
- dato il modello M e il mondo w , $\Box\varphi$ è vera (soddisfatta in questo mondo) sse per ogni w' accessibile da w allora è vera φ
- se esiste un mondo w accessibile da w in cui vale φ

Se la formula φ è vera per tutte le coppie modello mondo si dice che la formula è **valida**.

I modelli della logica modale sono spesso chiamati **Kripke structures**. La coppia $\langle W, R \rangle$ è chiamata **Kripke frame**. È un normale grafo.

2.1.4 Piccolo esempio

Guardando l'esempio, ci chiediamo se una determinata formula sarà valida nel mondo w_4 , ma essendo quest'ultimo un mondo dove non c'è nessuna relazione d'accessibilità e quindi il mondo w_4 non accede a nessun modo e questo ci porta a dire che ad es. $\Box q$ non è valido in w_4 , appunto non essendoci nessun arco che va in un altro mondo. In un modello

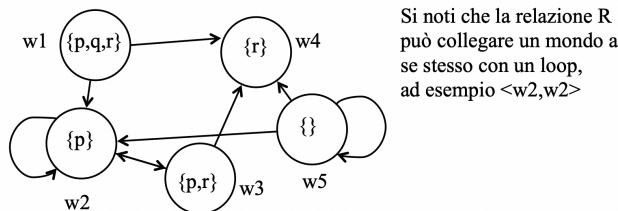
Un modello può essere rappresentato con un grafo i cui nodi contengono insiemi di proposizioni.

Esempio:

$$W = \{w_1, w_2, w_3, w_4, w_5\}$$

$$R = \{\langle w_1, w_2 \rangle, \langle w_1, w_4 \rangle, \langle w_2, w_2 \rangle, \langle w_2, w_3 \rangle, \langle w_3, w_2 \rangle, \langle w_3, w_4 \rangle, \langle w_5, w_2 \rangle, \langle w_5, w_4 \rangle, \langle w_5, w_5 \rangle\}$$

$$L = \{\langle w_1, \{p, q, r\} \rangle, \langle w_2, \{p\} \rangle, \langle w_3, \{p, r\} \rangle, \langle w_4, \{r\} \rangle, \langle w_5, \emptyset \rangle\}$$



Si noti che la relazione R può collegare un mondo a se stesso con un loop, ad esempio $\langle w_2, w_2 \rangle$

7

Figure 2.1

$M = \langle W, R, L \rangle$ la relazione di accessibilità R è utilizzata per definire la semantica degli operatori modali:

- la formula $\Box\phi$ è vera in un mondo $w \in W$ se ϕ è vera in tutti i mondi accessibili da w ;
- la formula $\Diamond\phi$ è vera in un mondo $w \in W$ se ϕ è vera in qualche mondo accessibile da w

Riprendiamo l'esempio di modello visto in precedenza 2.1.

- Consideriamo il mondo w_2 . Da questo mondo si accede a w_2 e w_3 . Ambedue i mondi contengono p e quindi $\Box p$ è vero nel mondo w_2 .
- Viceversa, $\Box r$, non è vero nel mondo w_2 perché w_2 non contiene r . Se non ci fosse il loop su w_2 , allora $\Box r$ sarebbe vero in w_2
- $\Diamond r$ è vero in w_2

Una piccola parentesi: Ex Falso Quodlibet

Consideriamo ancora l' esempio e chiediamoci se la formula $\diamond(r \wedge \square q)$ è vera nel mondo w_1 . Intuitivamente non lo è, visto che q non è raggiungibile da nessuna parte. Tuttavia, nel formulare la sintassi, abbiamo trascurato che normalmente la sintassi della logica prevede anche le due costanti logiche *true* e *false*. Da questo deriva la cosiddetta regola **Ex Falso Quodlibet**, che dice che da falso segue qualunque cosa. Questa regola può essere dimostrata e quindi ha un significato semantico, anche se è poco intuitiva. Tornando alla formula iniziale, da w_1 possiamo accedere a w_4 dove vale r . Da w_4 è falso che esista un qualunque mondo accessibile, e quindi, da questa affermazione segue qualunque cosa: in particolare in w_4 vale $\square q$. Quindi possiamo dire che la formula $\diamond(r \wedge \square q)$ è vera nel mondo w_1 .

NOTA: quanto detto in questa slide non è particolarmente significativo per la lezione. Chi fosse interessato all'argomento, può trovare molto materiale su Internet o su testi di logica.

2.1.5 Proprietà della logica modale

Dalla semantica della logica modale seguono due proprietà fondamentali:

- **assioma K** (in onore di Kripke) $\square(\varphi \Rightarrow \psi) \Rightarrow (\square\varphi \Rightarrow \square\psi)$
- **regola di inferenza**. Data una formula φ da questa si può inferire $\square\varphi$. Quindi, se φ è vera allora anche $\square\varphi$ sarà vera (se φ è valida allora anche $\square\varphi$ sarà valida). Questa si applica solo se φ è vera in tutti i modelli.

Altre proprietà

Si può notare come \square distribuisce sull' \wedge , ossia

- $\square(\varphi \vee \psi) \equiv (\square\varphi \vee \square\psi)$

mentre non distribuisce su \vee

- $\square(\varphi \vee \psi) \not\equiv \square\varphi \vee \square\psi$

Ad esempio in questo modello $\square(p \vee q)$ è vera in w , ma $(\square p \vee \square q)$ non lo è.

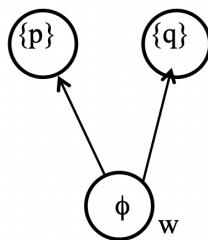


Figure 2.2

2.2 Logica modale per gli Agenti

Nel primo capitolo su Logica e Agenti abbiamo affermato che la logica modale viene normalmente usata per modellare e ragionare su diverse proprietà di un agente intelligente. A questo punto, dopo aver descritto in dettaglio la logica modale, possiamo chiederci cosa c'entri questa con le problematiche degli agenti, visto che la logica modale è la logica del *necessario* del *possibile*. La risposta è molto semplice: dare all'operatore modale \square (e al suo duale \diamond) un significato diverso da quello standard (necessario o possibile) che modelli una proprietà degli agenti, **senza modificare** la semantica basata sui mondi possibili. Ad esempio, volendo ragionare sulle sole credenze (*beliefs*) di un agente, potremmo attribuire all'operatore \square il significato appunto di credenza.

- $\Box\phi$, l'agente crede che ϕ sia vera.

N.B. Per modellare proprietà degli agenti si usano nomi diversi per l'operatore \Box .

Es. **B** per le credenze.

Di solito per quanto riguarda l'operatore duale non viene attribuito un nome, dunque si fa uso dell'equivalenza data in precedenza: $\Diamond\phi \equiv \neg\Box\neg\phi$ Concetto principale che per usare la logica modale l'operatore di base è \Box che può modellare delle cose diverse tra loro, non solo conoscenze ma anche credenze. Per modellare con la logica modale le diverse proprietà degli agenti (conoscenze, credenze, tempi, ...), può essere necessario modificare la logica in modo opportuno. Questo può essere fatto aggiungendo nuovi assiomi a quelli validi in generale (si ricorda che l'assioma **K** vale sempre).

Ad esempio, se \Box rappresenta la conoscenza, vorremmo che la logica avesse la proprietà che tutto ciò che è conosciuto è vero. Questo può essere espresso aggiungendo l'assioma $\Box\phi \Rightarrow \phi$. Oppure potremmo esprimere che le conoscenze dell'agente non devono essere contradditorie aggiungendo semplicemente l'assioma $\Box\phi \Rightarrow \neg\Box\neg\phi$

Nota: La formula $\Box\phi \Rightarrow \phi$ non vale per tutti i modelli.

2.2.1 Principali logiche modali per agenti

Tante logiche possono essere modellate con la logica modale e ognuna di queste logiche richiederà operatori con dei nomi diversi. Il problema è che per trattare e modellare la logica con questi operatori modali rinominati (invece del K usiamo il Box), obbliga in qualche modo la logica a essere modificata in modo che si possa modellare ciascuno di questi esempi. Citiamo adesso quelle che sono le logiche modali più usate per modellare agenti

- **Logica epistemica**(conoscenze e credenze)

- $K_a\phi$, l'agente a sa che ϕ è vero.
- $B_a\phi$, l'agente a crede che ϕ sia vero.

- **Logiche Belief-Desire-Intention**

- $B_a\phi$, l'agente a crede che ϕ sia vero
- $D_a\phi$, l'agente a desidera ϕ
- $I_a\phi$, l'agente a ha l'intenzione ϕ

- **Logiche deontiche (obblighi e permessi)**

- O_ϕ , è obbligatorio che ϕ
- $P\phi$, è permesso che ϕ

- **Logica temporale (lineare)**

- $X\phi$. ϕ , sarà vero al prossimo istante
- $G\phi$ ϕ , sarà sempre vero
- $F\phi$ ϕ , prima o poi diventerà vero
- $\psi U \pi$, ψ è vero fino a quando ϕ diventa vero.

- **Logica dinamica (delle azioni)**

- $[\pi]\phi$, dopo l'esecuzione del programma $\pi\phi$ è vero

π è un'azione complessa (programma) ottenuta combinando azioni elementari.

2.2.2 Proprietà dei frame

Abbiamo detto di aggiungere degli assiomi, ma quello che succede è che non siamo obbligati ad aggiungerli. È stato dimostrato che un'altra possibilità per modellare un certo comportamento o proprietà di un agente, oltre ad aggiungere assiomi, è quello di restringere la classe dei modelli mettendo delle condizioni per i **frame**. I frame ricordo vuol dire che il modello viene visto come un grafo senza nessuna informazione associata ai vari mondi. Ad esempio, per ragionare in un certo modo, uno può richiedere che valga una proprietà, la **riflessività**, ossia che ogni mondo di un frame $\langle W, R \rangle$ accede a se stesso: $\forall w \in W. (w, w) \in R$. Per ogni mondo w esiste una relazione tra w e se stesso. Quindi questo è un vincolo che viene messo sulla struttura del frame, ovvero del modello. Abbiamo delle proprietà che possiamo esprimere con degli assiomi, oppure esprimerle come proprietà del frame.

Le principali proprietà

- **reflexive** $\forall w \in W. (w, w) \in R$
- **serial** $\forall w \in W. \exists w' \in W. (w, w') \in R$. Vuol dire che non si arriva mai al punto in cui non si esce più da un mondo. Per ogni mondo esiste un mondo w' tale che w, w' sono in relazione
- **transitive** $\forall w, w', w'' \in W. ((w, w') \in R \wedge (w', w'') \in R) \Rightarrow (w, w'') \in R$
- **Euclidean** $\forall w, w', w'' \in W. ((w, w') \in R \wedge (w, w'') \in R) \Rightarrow (w', w'') \in R$

2.2.3 Corrispondenza

Possiamo formulare queste proprietà o con degli assiomi oppure con delle **condizioni sui frame**. Quello che è stato scoperto e dimostrato (anche se non è banale farlo) è che esiste una corrispondenza fra gli assiomi e le condizioni del frame. Ci sono tanti casi in cui esprimiamo una proprietà sottoforma di un assioma, esprimiamo la stessa proprietà sotto forma di condizione e le due sono equivalenti. Ad esempio si può dimostrare che $\Box\varphi \Rightarrow \varphi$ corrisponde alla definizione di **riflessività**. In seguito vengono elencate quelle che sono le principali corrispondenze. Lettere e numeri saranno i nomi date alle varie proprietà.

Le principali corrispondenze

Diversi sistemi modali interessanti sono basati sulle proprietà seguenti.

Axiom schema

condition on frames

T $\Box\varphi \Rightarrow \varphi$

reflexive

$\forall w \in W. (w, w) \in R$

D $\Box\varphi \Rightarrow \Diamond\varphi$

serial

$\forall w \in W. \exists w' \in W. (w, w') \in R$

4 $\Box\varphi \Rightarrow \Box\Box\varphi$

transitive

$\forall w, w', w'' \in W. ((w, w') \in R \wedge (w', w'') \in R) \Rightarrow (w, w'') \in R$

5 $\Diamond\varphi \Rightarrow \Box\Diamond\varphi$

Euclidean

$\forall w, w', w'' \in W. ((w, w') \in R \wedge (w, w'') \in R) \Rightarrow (w', w'') \in R$

Figure 2.3: Principali corrispondenze

Prendiamo la riflessività c'è un arco su stesso che torna su se stesso. Se un frame ha quella proprietà allora vale anche la serialità, questo perché la serialità stessa vuol dire essere collegato con se stessa.

2.2.4 Sistemi modali

A ciascuna delle proprietà elencate nella slide precedente è stato dato un nome: T, D, 4 e 5. Combinando quelle proprietà si ottengono 11 sistemi modali. In realtà sarebbero 16, ma alcune combinazioni sono equivalenti. Siccome questi sistemi modali sono ampiamente utilizzati, a ciascuno di essi è stato dato un nome:

- KT is known as T
- KT4 is known as S4
- KD45 is known as weak-s5
- KTD45 is known as S5

L'assioma K vale sempre.

Per curiosità il problema della validità per i sistemi modali sopra elencati è decidibile, e la complessità è PSPACE complete.

2.2.5 Proof theory

Come è ovvio, la logica dovrebbe fornire metodi e strumenti per ragionare. Questo vale per la logica classica: in altri corsi avrete visto alcuni di questi metodi come, ad esempio il metodo di Risoluzione particolarmente adatto per implementare theorem provers.

Purtroppo, il problema di definire una proof theory per la logica modale è più complesso per varie ragioni. La ragione principale è quella che abbiamo appena visto: non esiste solo una logica modale ma esistono diversi sistemi modali, ciascuno con proprietà diverse dagli altri. Per di più, spesso per modellare agenti e multiagenti si utilizzano logiche multimodali, ossia logiche che fanno uso di più operatori modali che hanno caratteristiche diverse uno dall'altro.

Per questi motivi noi non vedremo niente di sistematico sulla proof theory della logica modale. Più avanti vedremo qualche esempio di ragionamento in casi particolari.

Chapter 3

Logica Epistemica

Iniziamo a vedere come la logica modale può essere usato per modellare e descrivere proprietà degli agenti. In particolare, cominceremo dalla logica epistemica, la logica della **conoscenza** e delle **credenze**. Ci soffermeremo su alcune proprietà, per concludere con un esempio di ragionamento. È una logica molto studiata nel contesto di agenti e sistemi multiagenti, in generale quindi in AI. È una logica modale in cui gli operatori modali hanno un significato diverso rispetto a quelli standard. Vengono così introdotti due **operatori modali**:

- K_a , per la *knowledge*
- B_a , per le *credenze*

dove a è l'agente

- $K_a\varphi$, l'agente a sa che φ è vero, quindi rappresentiamo la conoscenza;
- $K_a\varphi$, intendiamo dire che l'agente crede che φ sia vera.

Entrambi hanno la semantica del \square . Nel caso ci dovesse servire il duale, così come succedeva nella logica introdotta precedentemente, possiamo utilizzare $\neg K_a \neg \varphi$ e $\neg B_a \neg \varphi$, rispettivamente per conoscenza e credenze

3.1 Onniscienza Logica

Vediamo come la logica modale può essere usata come logica epistemica e quali sono gli eventuali problemi. Come sappiamo ogni sistema modale soddisfa le seguenti proprietà:

- $K(\varphi \Rightarrow \psi) \Rightarrow (K\varphi \Rightarrow K\psi)$ **assioma K**¹;
- se φ è valida, allora anche $K\varphi$ è valida, **Necessitation**.

Queste due proprietà, messe assieme portano a fare delle considerazioni/conclusioni. Con questa logica modale (ovvero quella epistemica), la modellazione che diamo dell'agente, porta ad un problema, noto come **onniscienza logica**. L'agente dovrebbe sapere tutto. Questa considerazione "salta fuori" nel seguente modo:

- assumiamo che ψ sia una conseguenza logica di φ . Allora $\varphi \Rightarrow \psi$ deve essere valida. Secondo la *necessitation*, questa formula deve essere conosciuta dall'agente:
 $K_a(\varphi \Rightarrow \psi)$

¹assioma K è diverso da K, lassioma K fa riferimento all'assioma di Kripke. DA NON CONFONDERE CON K - knowledge

- per di più possiamo anche applicare l'**assioma K**. L'assioma dice che se l'agente conosce φ , ossia $K_a\varphi$, allora per l'**assioma K** deve conoscere anche ψ . Significa che l'agente deve conoscere tutte le conseguenze che può derivare da quello che sa. Nel modellare un agente reale uno non pensa di fare assunzioni di questo genere. Un agente reale sarà in grado di ragionare con qualche vincolo di ragionamento, ma no di certo con dei vincoli come quelli visti, ovvero quello di *sapere per forza tutto*. **Da questo punto di vista, la logica modale, non è lo strumento giusto per modellare agenti reali**

Per quanto ci riguarda, avendo fatto queste considerazioni, noi rimaniamo con la logica modale vedendo altri aspetti importanti che sono ricavabili dalle proprietà della logica modale.

3.2 Assiomi per knowledge e belief

Per quello che abbiamo visto alla fine del capitolo precedente, si possono realizzare dei sistemi modali, aggiungendo nuovi assiomi alla logica modale standard. Quelli visti erano **D**, **T**, **4**, **5**, questi possono essere usati per modellare KNOWLEDGE e BELIEFE.

- **assioma D (serialità)** dice che la conoscenza di un agente è non-contraddittoria (ricordo come al posto del \square mettiamo **K**)
 $K\varphi \Rightarrow \neg K\neg\varphi^2$
 "se l'agente conosce φ allora non conosce $\neg\varphi$ ". Ragionevole per **K** e **B**

Concludiamo dicendo che questo assioma è buon per modellare knowledge e belief e quindi lo possiamo aggiungere alla logica modale standard in modo da raffinare la proprietà della conoscenza.

- **assioma T (riflessività)**: $K\varphi \Rightarrow \varphi$ (cio che l'agente conosce è vero).
 Axioma accettabile per la conoscenza ma non per le credenze: non vogliamo che un agente conosca qualcosa che è falso, ma accettiamo che l'agente creda vero qualcosa che è falso.

Per poter capire e vedere cosa succede con i Beliefs, possiamo sostituire K con B , $B\varphi \Rightarrow \varphi$ e questo trasforma l'assioma in qualcosa di poco corrispondente a quella che è la realtà del B . Abbiamo detto che l'agente può credere un po a quello che gli pare rispetto al mondo reale quindi, per un agente possiamo accettare che creda a qualunque cosa, ma **non possiamo accettare** la regola che dice che se un agente crede φ allora φ è vero. NON POSSIAMO DIRLO. Ciò che l'agente crede può essere sia falso che vero.

- **assioma 4 (introspezione positiva)** $K\varphi \Rightarrow KK\varphi$. Axioma della transitività

Axioma della transitività. Se lo leggiamo leggendo K come conoscenza, c'è scritto che se l'agente sa φ , sa di sapere che φ sia vera. È anche noto come axioma d'introspezione positiva, quindi se l'agente sa una cosa sa di saperla.

- **assioma 5 introspezione negativa** $\neg K\varphi \Rightarrow K(\neg K\varphi)$

Complicato solo come notazione, ma tranquilli è na strunzat. Se un agente non sa di conoscere φ , allora sa di non saperlo.

Questi quattro assiomi, possono essere utilizzati per modellare, quindi essere aggiunti alla logica modale per ottenere un **sistema modale**. In particolare, per quanto riguarda la conoscenza, tutti e quattro vanno bene. Per quanto riguarda i Beliefs, lasciamo fuori l'assioma T e quindi andremo a considerarne solo tre.

² $\neg K\neg\varphi$ è l'equivalente di $\diamond\varphi$

3.3 Un esempio d'applicazione: Muddy children

È un esempio di ragionamento che si può fare usando la logica modale, ma utilizzando le sole regole appena introdotte.

Dopo aver giocato nel fango, due bambini sanno che almeno uno di loro ha del fango sulla fronte (in realtà tutti e due lo hanno). Di solito i bambini sono almeno tre. Per semplicità qui vediamo solo l'esempio con due. Ogni bambino può vedere la fronte dell'altro, ma non la propria. Inizialmente il bambino B dice "Non so se ho la fronte infangata" Successivamente il bambino A dice "Io so di avere la fronte infangata" Indichiamo con K_A e K_B le conoscenze del bambino A e di B . Vediamo alcune formule che descrivono le conoscenze di A dopo che B ha parlato.

$$1. K_A(\neg \text{muddy}(A) \Rightarrow K_B(\neg \text{muddy}(A)))$$

A sa che, se A non ha la fronte infangata B lo sa

$$2. K_A(K_B(\neg \text{muddy}(A) \Rightarrow \text{muddy}(B)))$$

A sa che B sa che, se A non ha la fronte infangata, allora
 B ha la fronte infangata

$$3. K_A(\neg K_B(\text{muddy}(B)))$$

A sa che B non sa se ha la fronte infangata.

Vogliamo dimostrare $K_A(\text{muddy}(A))$ partendo dalle tre formule qui sopra.

Figure 3.1: Muddy children - situazione iniziale

$$1. K_A(\neg \text{muddy}(A) \Rightarrow K_B(\neg \text{muddy}(A)))$$

$$2. K_A(K_B(\neg \text{muddy}(A) \Rightarrow \text{muddy}(B)))$$

$$3. K_A(\neg K_B(\text{muddy}(B)))$$

$$4. \neg \text{muddy}(A) \Rightarrow K_B(\neg \text{muddy}(A)) \text{ da 1, assioma della conoscenza}$$

$$5. K_B(\neg \text{muddy}(A) \Rightarrow \text{muddy}(B)) \text{ da 2, assioma della conoscenza}$$

$$6. K_B(\neg \text{muddy}(A) \Rightarrow K_B(\text{muddy}(B))) \text{ da 5, assioma K}$$

$$7. \neg \text{muddy}(A) \Rightarrow K_B(\text{muddy}(B)) \text{ da 4 e 6 per transitività}$$

$$8. \neg K_B(\text{muddy}(B)) \Rightarrow \text{muddy}(A) \text{ contrapposto di 7}$$

$$9. K_A(\neg K_B(\text{muddy}(B)) \Rightarrow \text{muddy}(A)) \text{ da 8 per necessitation}$$

$$10. K_A(\neg K_B(\text{muddy}(B)) \Rightarrow K_A(\text{muddy}(A))) \text{ da 9, assioma K}$$

$$11. K_A(\text{muddy}(A)) \text{ da 3 e 10 per modus ponens}$$

modus ponens: se α e $\alpha \Rightarrow \beta$, allora β

Figure 3.2: Muddy children

Chapter 4

Modellare tempo e azioni

Analizziamo alcuni degli strumenti formali che vengono utilizzati per modellare e descrivere proprietà di agenti. Vedremo strumenti di tipo logico formale che possono essere utilizzati per modellare il tempo e le azioni. Ci concentriamo sulle **logiche temporali** e di meccanismi/proposte fatte, che hanno permesso il ragionamento su azioni. Quest'ultimo punto è legato principalmente sulla logica classica.

D. Ci si chiede, perchè viene utile ragionare sulle azioni?

R. La principale caratteristica di un agente è quello di essere in un **ambiente**, muoversi e sulla base di qualche tipo di percezione perseguire nei propri obiettivi. Quello che vedremo e cosa può servire per ragionare su agenti usando **tempo** (quindi evoluzione nel tempo) o le **azioni**(cosa succede se un agente esegue un'azione anzichè un'altra)

4.1 La logica temporale

La **logica temporale** è una logica modale che può essere vista appunto come una logica temporale. Ci sono tante logiche temporali che sono usate per trattare aspetti diversi degli agenti. In particolare, noi tratteremo i casi in cui:

- il tempo è discreto;
- ha un istante iniziale;
- è infinito nel futuro;

Comunque sia fatta questa logica temporale, supponiamo che gli operatori della logica siano fatti in modo che operino verso il futuro. Noi consideremo solo due tipi di logiche:

- **linear-time logic**, dove la struttura del tempo è lineare. Gli istanti di tempo sono in fila uno dietro l'altra e descrivono un'evoluzione nel tempo che va avanti fino all'infinito;
- **branching-time logic**, la struttura del tempo è fatta ad albero, dove ogni istante può avere più istanti successori, quindi alberi infiniti.

4.1.1 Linear Temporal Logic (LTL)

La semantica

Abbiamo un **modello** M visto come una struttura lineare $\langle S, x, L \rangle$ dove:

- S è l'insieme di stati (quelli che erano i mondi nella logica modale);
- $x : N \rightarrow S$, sequenza infinita di stati. Dicevamo lineare perchè dopo uno stato si passa ad un altro stato, fino all'infinito. Gli stati sono rappresentati da **numeri naturali**;

- $L : S \rightarrow 2^P$, dove L associa ad ogni stato l'insieme di proposizioni che sono vere in quello stato (così come accade nella logica modale)

Si nota come la struttura è simile a quella della logica modale. Cosa importante che il modello viene visto come una **struttura lineare**, dove ognuno può muoversi un passo per volta seguendo questa sequenza.

Significato di struttura lineare

Dallo stato 1, passiamo allo stato 2 e così via. Quindi, gli stati possono essere visti (se rappresentati graficamente), sottoforma di modello della logica modale, come una **sequenza di relazioni di accessibilità** dove dallo stato s_0 si accede allo stato s_1 e così via. Alla fine, la semantica della logica, si nota come assomiglia a quella modale.

La sintassi

Le formule della **Propositional Linear Temporal Logic (PLTL)** sono:

- p
- $\alpha \vee \beta$
- $\neg\alpha$
- $\mathbf{X}\alpha$
- $\alpha \mathbf{U} \beta$

dove $p \in P$ e $\alpha, \beta \in \text{PLTL}$. Si nota come in aggiunta ci sono due nuovi operatori modali $\mathbf{X}\alpha$ e $\alpha \mathbf{U} \beta$ che hanno la seguente semantica informale:

- $\mathbf{X}\alpha$, operatore di **next-time**. Siamo in uno stato e questo significa che nel prossimo stato (prossimo istante di tempo) α sarà vera. Questo $\mathbf{X}\alpha$ indica quindi cosa deve essere vero nello stato successi;
- $\alpha \mathbf{U} \beta$, questo operatore modale è vero al tempo t iff β è vera in un futuro istante t' e α è vera in tutti gli istanti tra t e t' . **operatore non tantissimo usato**.

Gli operatori appena introdotti permettono di derivare altri operatori modali

- $\mathbf{F}\alpha \equiv \text{true } \mathbf{U}\alpha$ ("prima o poi α "). È definito come **until**, in cui il primo parametro è sempre vero e α è quello successivo. Semantica che prima o poi si arriva ad α . La differenza con l'until di prima è il primo argomento della **U**, adesso deve essere sempre *true*. Operatore modale che si tende ad utilizzare parecchio e spesso viene chiamato con F
- $\mathbf{G}\alpha \equiv \neg\mathbf{F}\neg\alpha$ ("sempre α "). Viene definito come $\neg\mathbf{F}\neg\alpha$, e quindi come duale di F . Sarà sempre vero α in tutti gli stati successivi. Non è vero che prima o poi si trova $\neg\alpha$.

Esempi di formule LTL

- $\mathbf{G}(p \Rightarrow \mathbf{X}q)$, è sempre vero che $p \Rightarrow \mathbf{X}q$. Quindi nello stato in cui si fa riferimento, è vero che se p è vero allora $\mathbf{X}q$, quindi che nello stato successivo (prossimo istante di tempo) sarà vero q
- $\mathbf{GF}p$ (sempre vero che), prima o poi p ($\mathbf{F}p$) e questa è sempre soddisfatta. Visto che i modelli sono sequenza infinite, andando verso l'infinito si trova in continuazione p
- $\mathbf{G}p \wedge \mathbf{F}\neg p$, è sempre vero che in uno stato è vero p e è falso $\neq p$, quindi che p è sempre vero in ogni stato ma che prima o poi si troverà con una contraddizione.

4.1.2 Computation Tree Logic (CTL*)

Nel caso di LTL, il modello era lineare, quindi la computazione che si vuole modellare in LTL è costituita da una sequenza di stati. Qui, a differenza di quello che succedeva nell'LTL, la struttura, quindi la semantica del modello è fatta ad **albero** (non è più una sola sequenza). Consideriamo sempre che l'evoluzione sia sempre da uno stato di partenza e ci si muove sempre in direzione verso l'infinito, non considerando mai la possibilità di tornare indietro.¹. In **CTL***, le caratteristiche principali sono:

- essere più complicata rispetto all'LTL e tale complicazione è dovuta a:
 - **path formulas**, formule relative ad un cammino. Prendiamo un qualunque cammino nell'albero e se ragionassimo su un singolo cammino dell'albero e come se usassimo la LTL. Queste sono formule che si riferiscono ad un cammino infinito nella struttura temporale;
 - **state formulas**, che riguardano tutti i cammini infiniti uscenti da un stato.

La sintassi

- **state formulas**:

- p ;
- $\alpha \vee \beta$;
- $\neg\alpha$;
- **A** π , la proprietà che stiamo utilizzando è che tutti i cammini uscenti da uno stato valgono π
- **E** π , formula di **quantificazione esistenziale** dove esiste un cammino uscente da uno stato per cui vale π

dove $p \in P$, α e β sono state formulas, π è una path formulas

A significa "per tutti i cammini uscenti da uno stato vale π "

E significa "esiste un cammino uscente da uno stato per cui vale π "

A e **E** sono duali.

- **paths formulas**:

- α ;
- $\pi \vee \rho$;
- $\neg\pi$;
- **X** π ;
- $\pi \mathbf{U} \rho$;

dove α è una state formula, e π, ρ sono path formulas.². Le paths formulas sono essenzialmente come le formule della logica LTL

La semantica

Il modello M è una tripla $\langle S, T, L \rangle$ dove S e L sono come in LTL, mentre informalmente T è un **albero infinito** i cui nodi sono stati. Dato un modello M , la semantica **state formulas** si riferisce ad uno stato $(M, s \models \alpha)$ dove se l'agente è in uno stato s , può chiedersi se soddisfa α e la semantica di un path formula si riferisce ad un cammino $(M, x \models \pi)$, ovvero se c'è la sequenza x va a vedere se questa vale π .

Ad esempio:

¹Alcune logiche linear time comunque permettono di tornare indietro

²quello che si legge adesso è da chiarire. Con le paths formulas si considerano i cammini che partono da uno stato

- $M, s_0 \models \mathbf{A}\pi$ iff $\forall \text{path } x = (s_0, s_1, s_2, \dots) M, x \models \pi$ se abbiamo un modello M , stato iniziale s_0 , la formula $\mathbf{A}\pi$ è soddisfatta iff per ogni cammino x valga π

4.1.3 Da CTL* a CTL

La logica CTL* abbiamo visto che fa riferimento a delle paths formulas che sono sostanzialmente quelle della logica LTL e quindi vedere che la logica LT è un sottoinsieme di questo CTL*, in quanto questa, comprese la linear time logic, ma anche tante altre cose. Essendo CTL* molto complessa, risulta molto difficile usarla in pratica e quindi quello che è stato fatto è di prendere anche qui (come avviene in LTL) un sottocaso di CTL*. Tale sottoclasse prende il nome di CLT e consiste nell'andare a ridurre le proprietà di questa logica in modo da rendere più comoda e più facile da utilizzarla nelle applicazioni. Questa logica non permette **combinazioni e annidamenti** di operatori linear time. Quindi, essendo più vicina all'LTL anche le paths formulas diventano:

- $\mathbf{X}\alpha$
- $\alpha\mathbf{U}\beta$

dove α e β sono state formulas

CTL vs LTL

Ci si chiede quando scegliere una anzichè l'altra. Naturalmente questa scelta viene "fuori" perchè sono diverse. Il fatto di usare LTL o CTL è dovuto al tipo di problema che si sta per affrontare. Si può vedere che:

- non c'è nessuna formula CTL che sia equivalente alla formula LTL. $\mathbf{FG}p$ (c'è uno stato lungo un cammino dopo il quale p sarà per sempre vero). La formula $\mathbf{FG}p$ non può essere espressa in CTL perchè questo non accetta **operatori modali annidati**
- non c'è nessuna formula LTL che sia equivalente alla formula CTL. $\mathbf{AG}(\mathbf{EF}p)$ (per tutti i cammini uscidi da uno stato, allora sarà vero p). Questa essendo formula CTL non può essere rappresentata come formula LTL

4.2 Ragionare sulle azioni

All'inizio del capitolo, abbiamo richiamato che un agente intelligente deve essere in grado di ragionare sul *tempo* e sugli *effetti delle azioni* (sue o di qualche altro agente). Il **ragionamento sulle azioni**, è stato argomento trattato e analizzato in un articolo del 1969. Il ragionamento sulle azioni si basa sulla logica classica.

Aspetti principali del calcolo delle situazioni

- **Situazioni:** sono i *mondi, stati del mondo* a qualche istante di tempo. Dove, S_0 rappresenta la **situazione iniziale**, $do(a, s)^3$ denota una **situazione risultante** dall'esecuzione dell'azione a nello stato s ;
- **Fluenti:** proposizioni il cui valore varia da una situazione ad un'altra. Esempio $sta_portando(robot, oro, s_0)$. Il robot sta portando dell'oro nella situazione iniziale; **Azioni:** causano un cambiamento nello stato del mondo

³serve per dire cosa succede nello stato successivo. Siamo nello stato s ed eseguiamo l'azione a e finiamo nello stato $do(a, s)$

4.2.1 Azioni: precondizioni e effetti

Ogni azione è descritta da due assiomi: un **assioma di possibilità** e un **assioma di effetto**. Il primo dice quando è possibile seguire l'azione, il secondo specifica quello che accade quando un'azione possibile è eseguita.

Esempio da Russell e Norvig:

$\text{Posizione}(\text{Agente}, x, s) \wedge \text{Adiacente}(x, y) \Rightarrow \text{Poss}(\text{Vai}(x, y), s).$

$\text{Oro}(g) \wedge \text{Posizione}(\text{Agente}, x, s) \wedge \text{Posizione}(g, x, s) \Rightarrow \text{Poss}(\text{Afferra}(g), s).$

$\text{Portando}(g, s) \Rightarrow \text{Poss}(\text{Lascia}(g), s).$

$\text{Poss}(\text{Vai}(x, y), s) \Rightarrow \text{Posizione}(\text{Agente}, y, \text{do}(\text{Vai}(x, y), s)).$

$\text{Poss}(\text{Afferra}(g), s) \Rightarrow \text{Portando}(g, \text{do}(\text{Afferra}(g), s)).$

$\text{Poss}(\text{Lascia}(g), s) \Rightarrow \neg \text{Portando}(g, \text{do}(\text{Lascia}(g), s)).$

Figure 4.1: Assioma dell'effetto

4.2.2 Frame problem

Un aspetto analizzato dagli autori (McCarthy e Hayes) è il seguente.

- se eseguo un'azione, questa in qualche modo modifica lo stato. Il problema è capire come esprimere in modo parsimonioso (senza usare troppe cose tecniche) che tutto il resto non cambia

Quello appena detto è chiamato **frame problem**. È stato chiamato in questo modo perchè il frame è quel filmetto che ognugno di noi vede scorrere. Quando si passa da un fotogramma ad un altro cambia solo qualcosa (davvero poco) rispetto a tutto quello che rimane fero. Il problema sembra banale perchè occorre capire il modo per descrivere il passaggio da uno stato all'altro in cui ci sono *fluenti* che cambiano all'esecuzione di una certa azione e altri che non cambiano, quindi che rimangono com'erano al passo precedente. Nel **mondo dei blocchi**, l'esecuzione di un blocco porta il blocco stesso in un'altra posizione, quindi in un altro stato. Nel nuovo stato i vecchi blocchi, ovvero quelli presenti nello stato di partenza prima che l'azione portasse allo spostamento del blocco, rimangono fermi. Il problema è capire come fare a dire che un fluenti si sposta quando si esegue un'azione e tutti gli altri rimangono fermi.

- da raccontare sembra banale, ma l'implementazione vorrebbe evitare la soluzione banale in cui tutto quello che non cambia non si sposta e quindi elencare tutte le cose che non sono cambiate diventa un'operazione estremamente inefficiente in cui molti hanno studiato

Un modo per poter descrivere questo fenomeno in cui tutti gli oggetti rimangono li dov'erano è ad esempio con l'uso della **negazione per fallimento** come viene fatto in Prolog. Questo uso può essere visto come un meccanismo interessante per la modellazione del frame problem e quindi dire che solo qualcosa cambia da uno stato all'altro.

4.2.3 Successor State Axioms

Reiter ha proposto una soluzione del frame problem introducendo i **Successor State Axioms**, che fanno riferimento ad un fluente per volta. Andiamo a vedere come cambia il valore di un fluente per volta. Nell'esempio, questo può essere rappresentato dicendo che:

$$\begin{aligned} \text{Poss}(a,s) \Rightarrow [\text{broken}(x,\text{do}(a,s)) = \\ \exists r. \{a=\text{drop}(r,x) \wedge \text{fragile}(x,s)\} \vee \\ \text{broken}(x,s) \wedge \neg \exists r. a=\text{repair}(r,x))] \end{aligned}$$

Figure 4.2: Esempio di fluente

- se a è un'azione (a è una variabile che sta per azione, ma non sappiamo precisamente di che azioni si tratti) ed è possibile eseguirla nello stato s , allora [se viene eseguita l'azione x risulta rotto quindi *broken*, e questa cosa possiamo scriverla come esiste un $\exists r$ (entità non specificata) tale che l'azione a sia l'azione di $\text{drop}(r,x)$, ovvero azione che r faccia cadere x e (\wedge) x è fragile nello stato s OPPURE (\vee) x è rotto nello stato s and non esiste nessun r per cui ci sia un'azione a che aggiusti x]

Per la logica classica è una notazione pesante da utilizzare ma, il significato importante è quello di fare il più possibile per ridurre il numero di fluenti e proprietà che vengono valutate man mano che la computazione prosegue e quindi quello che può succedere è che ci possa essere un'azione (nel caso specifico) in cui *borken* era già vero prima e rimane rotto perché non c'è nessuno che lo aggiusta oppure viene lasciato cadere e si rompe perché fragile. Un **fluente** può rimanere vero nello stato s perché o non c'è nessuna azione che lo modifica oppure diventare un'altra caratterizzazione che li faccia cambiare stato se eseguita altra azione. In sostanza, abbiamo uno stato di partenza con un certo numero di fluenti, dove alcuni cambieranno stato, dove tale cambiamento è prodotto dall'esecuzione di un'azione, e altri fluenti non cambieranno in quanto essere già veri.

4.2.4 GOLOG

GOLOG (1994) è un linguaggio di programmazione, che vedremo più avanti, basato sul calcolo delle situazioni. Il linguaggio è usato per programmare robot di alto livello (cognitive robotics) e agenti software intelligenti.

Le azioni primitive in GOLOG sono specificate dandone le *precondizioni* e *effetti* (rappresentati come *successor state axioms*).

Chapter 5

Modellazione logica di agenti BDI

Cerchiamo di capire quello che si è discusso nei capitoli precedenti (logica e formalismi vari) per essere utilizzati in modo da modellare agenti basati sulla logica, in particolare utilizzando l'**approccio BDI** (belief, desire, intentions). Quello che vedremo sono due approcci di modellazione logica ad agenti BDI:

1. logica intenzionale di Cohen e Levesque;
2. logica BDI di Rao e Georgeff

Noi ci concentriamo su una struttura ad agenti che permette di combinare diversi aspetti(fino ad adesso ci si è sempre concentrati su un solo aspetto dell'agente). Ci aspettiamo che la logica dell'agente, così come citato nel libro di Wooldridge, sia in grado di rappresentare gli aspetti dinamici. Una teoria dell'agente completa deve essere in grado di

- mostare come sono correlate le informazioni e le intenzioni di un agente;
- come cambia lo stato cognitivo di un agente nel tempo;
- come informazioni e intenzioni di un agente portino lo stesso a compiere azioni

Secondo Wooldrige e Ciancarini, i metodi formali giocano tre ruoli:

- **specificare** sistemi. Dire come è fatto un agente/sistema di agenti. Un po quello che vedremo in questo capitolo
- **programmare direttamente** sistemi. Un sistema d'agenti può essere programmato direttamente utilizzando la logica. Un esempio classico è il Prolog, che utilizza la logica classica;
- **verificare** sistemi. La logica può servire per andare a vedere quelle che sono le proprietà di un agente in modo da verificare se le proprietà di un agente siano quelle corrette;

5.1 La logica intenzionale di Cohen e Levesque

Loro si focalizzano sulla modellazione di un agente in modo da considerarlo come un agente BDI, ovvero ritengono che il comportamento razionale dell'agente dovrebbe essere analizzato in termini di **beliefs, desires and intentions**(BDI).

5.1.1 Proprietà delle intenzioni

Gli autori elencano quelle che secondo loro sono le proprietà di un'intenzione:

- Le intenzioni pongono dei problemi per gli agenti, che devono determinare modi di soddisfarle.

- Le intenzioni forniscono un "filtro" per adottare altre intenzioni che non devono entrare in conflitto. Quindi se un agente ha un'intenzione, non è possibile che vi siano altre intenzioni che entrino in conflitto con quella;
- Gli agenti "tracciano" il successo delle loro intenzioni, e sono disposti a tentare di nuovo se il loro tentativo fallisce;
- Gli agenti credono che le loro intenzioni siano possibili;
- Gli agenti non credono che non riusciranno a soddisfare le loro intenzioni;
- Gli agenti non credono che non riusciranno a soddisfare le loro intenzioni;
- Gli agenti non si aspettano tutti i side-effects delle loro intenzioni.

I side-effects non sono intenzioni

Ultimo punto della lista risulta essere di particolare importanza. I side-effects non sono intenzioni, se un agente ha un'intenzione che comporta questo effetto collaterale, non è detto che l'effetto collaterale sia anch'esso un'intenzione.

Esempio del dentista. L'intenzione dell'agente è quella di andare dal dentista, ma questa provocherà un dolore. Questo dolore non può essere vista come un'**intenzione**. *side-effects* ≠ *intenzione*. Bratman sostiene che ciò che uno intende è **approssimativamente** un sottinsieme di ciò che uno sceglie. Cioè se uno si trova di fronte diverse intenzioni, ne sceglierà qualcuna lasciando da parte quelli che sono gli effetti collaterali

5.1.2 Formulazione della logica

Logica modale che permette di ragionare sia su belief, quindi formule, che su azioni. Ci sono degli operatori modali.

- (BEL $i \varphi$), l'agente i crede φ ;
- (GOAL $i \varphi$), i ha come goal φ , ma visto che ci riferiamo ad agenti BDI diciamo che GOAL = DESIRE
- (HAPPENS α), l'azione α sarà vera (si verificherà) al passo successivo;
- (DONE α), l'azione α si è appena verificata.

I due ultimi operatori modali non si verificano (come si vede) ad una formula, ma all'azione α , dove queste non sono da considerare come singole, ma come sequenze di azioni. Queste azioni, assomigliano molto a quelle presenti nella **logica dinamica** di cui avevamo solo dato un piccolo riferimento, parlando di quali fossero le logiche usate per gli agenti. La dinamica interessante per quanto riguarda la modellazione delle azioni, che tratta, sequenza di azioni o espresioni costruite con azioni, in sostanza sono dei programmi formalizzati.

Piccola nota. Si nota come *intention* non è un operatore modale primitivo ma, come vedremo, può essere derivato da quelli primitive descritti qui sopra.

5.1.3 La semantica

La logica con cui costruiamo gli agenti, si basa su diverse relazioni. Prendiamo i mondi (nodi del grafo in 5.1) e questi secondo la semantica modale sono legati sia da **relazioni**, ma da relazioni BEL, GOAL. Le relazioni quindi sono etichettate come dei BEL, GOAL, o altri aspetti non considerati in figura. Nei mondi, è contenuto un **insieme di proposizioni** che

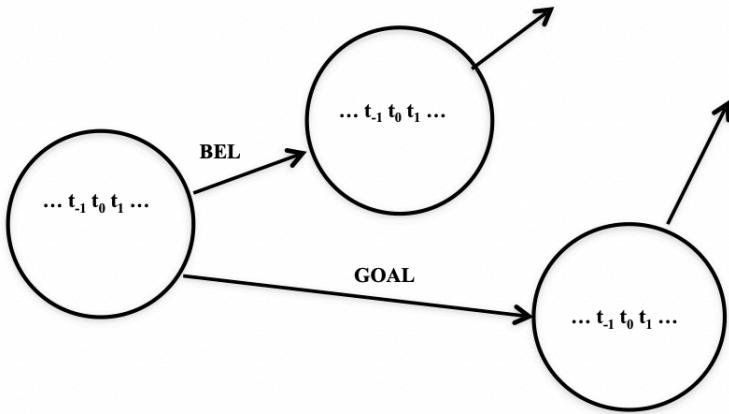


Figure 5.1: Semantica di Cohen

sono vere in quel mondo. In questo caso, il punto interessante è che la semantica si basa su una logica temporale LTL. Quello che gli autori dicono è che ogni mondo è basato su una logica temporale LTL. È lineare, quindi il modello del tempo è costituito da una sequenza lineare di istanti di tempo. Qui volendo, secondo la loro formulazione, possiamo andare anche indietro. Comunque, in uno stato c'è una struttura temporale linear time (sequenza di instanti di tempo). Le formule sono valutate rispetto a qualche mondo w_i , e un "indice" t_j in quel mondo, che rappresenta un punto nella sequenza degli eventi.

5.1.4 Modalità derivate

Se ci sono degli aspetti che vogliamo modellare e che non rientrano in quella che è la formulazione della logica di Cohen di base, possiamo allora modellarne di nuovi. Per poter ragionare dentro gli stati utilizziamo la logica LTL e quindi prendiamo gli operatori visti nella logica LTL, dove **G** e **F** possono essere definiti come abbreviazioni, con l'aggiunta di un altro operatore: **LATER**

- $\mathbf{F}\varphi \equiv \exists\alpha. (\text{HAPPENS } \alpha; \varphi?)^1$. Quindi per dire che prima o poi φ sarà vera, allora dovremmo dire che ci sarà una sequenza di passi, uno dietro l'altro, che porteranno ad un certo punto a trovare φ vera. Esiste un α (sequenza di azioni) tale che esiste una sequenza di azioni α che dopo la sua esecuzione verrà eseguita φ e φ risulterà vera;
- $(\text{LATER } \varphi) \equiv \neg\varphi \wedge \mathbf{F}\varphi$. Nello stato attuale φ è falso (quindi è vero $\neg\varphi$) ma che prima o poi diventerà vero

È possibile derivare anche un operatore di precedenza temporale (**BEFORE** $\varphi \psi$), ossia φ vale prima di ψ .

Per ogni sequenza di α , succede che viene eseguita α e poi ψ è vera. Se succede questo, però prima di ψ , deve valere φ , quindi esiste un β (sottostringa di α) in cui φ è vera.

5.1.5 Achievement Goals

Achievement goals sono quelli che l'agente crede falso, ma desidera che in futuro diventino veri. È un goal, un obiettivo, di arrivare ad una certa situazione in cui qualche cosa sia vera. Prendiamo il planning, la pianificazione rientra in questo, quando descriviamo un planning, diciamo di arrivare in uno stato in cui sia vera una certa formula. Loro lo definisco in questo modo

$$(\mathbf{A-GOAL}_i \varphi)(\mathbf{GOAL}_i(\mathbf{LATER} \varphi)) \wedge (\mathbf{BEL}_i \neg\varphi)$$

i crede che φ sia falso, ma desidera che in futuro diventi vero.

¹? è l'operatore di test.

5.1.6 Goal Persistenti

Da questo possiamo rendere l'achievement goal un **persistent goal**.

$$\begin{aligned} (\mathbf{P-GOAL}_i \varphi) \equiv & (\mathbf{GOAL}_i (\mathbf{LATER} \varphi)) \wedge (\mathbf{BEL}_i \neg\varphi) \\ & \wedge [\mathbf{BEFORE} ((\mathbf{BEL}_i \varphi) \vee (\mathbf{BEL}_i \mathbf{G} \neg\varphi)) \\ & \neg (\mathbf{GOAL}_i (\mathbf{LATER} \varphi))] \end{aligned}$$

La prima riga è esattamente quella che era la definizione di achievement goal. Abbiamo un **BEFORE** e da definizione sappiamo che un qualcosa deve avvenire prima di un'altra. Quindi prima riga è quella del **BEFORE**. Come ultima riga abbiamo un \neg . Se l'agente verifica in qualche modo che il goal non è raggiungibile, allora questo goal viene eliminato e questo viene fatto cona la negazione che c'è davanti. (se non ha il goal di arrivare a φ prima o poi allora questo lo può cancellare)

Prima di buttare via il goal deve essere vero $((\mathbf{BEL}_i \varphi) \vee (\mathbf{BEL}_i \mathbf{G} \neg\varphi))$, cioè o l'agente i crede φ quindi siamo arrivati ad un punto dove l'agente crede φ e quindi possiamo buttare via il goal oppure crede $\mathbf{G}\neg\varphi$, ovvero sarà sempre vero $\neg\varphi$ e quindi mai vero φ .

Definizione come in slide

Un agent ha il **persistent goal** φ se:

- ha un goal che φ prima o poi diventerà vero, e crede che φ attualmente non lo sia (achievement goal)
- prima di abbandonare il goal, una delle seguenti condizioni deve essere vera:
 - l'agente crede che il goal sia stato soddisfatto
 - l'agente crede che il goal non sarà mai soddisfatto

5.1.7 Intenzione di eseguire un'azione

- se un agente ha l'intenzione di un qualcosa allora questo agente si porterà dietro questa intenzione fino a quando non arriva ad eseguire quello che vorrebbe eseguire.
- in questo caso le intenzioni fanno riferimento a delle azioni e non a delle formule;

$$(\mathbf{INTEND}_i \alpha) \equiv (\mathbf{P-GOAL}_i [\mathbf{DONE}_i (\mathbf{BEL}_i (\mathbf{HAPPENS} \alpha)) ?; \alpha])$$

Quindi, se un agente ha l'intenzione di un azione, vuol dire che ha l'intenzione di eseguirla e quindi si aspetta di arrivare nel punto in cui sta per avvenire l'azione α (**Happens** α) e poi la esegue. L'agente è commit a credere di stare per eseguire l'azione α e quindi non è che possa modificare/eseguire cose che non intendeva. Il problema che volevano modellare gli autori è bloccare gli agenti che avevano un intenzione, quindi l'agente è bloccato/commit a quella intenzione. Per giustificare la definizione di **INTEND**, gli autori descrivono la seguente situazione: *Un agente intende uccidere suo zio. Nel percorso verso la casa dello zio, questa intenzione lo rende così agitato che lui perde il controllo della sua auto, e ammazza un pedone che risulta essere suo zio. Sebbene lo zio sia morto, noi certamente diremmo che l'azione eseguita dall'agente non sia ciò che lui intendeva.* Nell'esempio che abbiamo descritto, l'intenzione dell'agente è di eseguire un'azione. In modo analogo, è possibile definire l'intenzione come qualcosa che prima o poi si avverrà.

5.2 Rao and Georgeff's BDI logic

In questa logica l'aspetto BDI dell'agente è più evidente. La differenza che adesso fra le modalità di base (quelle di partenza) c'è anche l'intenzione² e questa non è costruita più come prima. Altra cosa che differisce dalla precedente è che la semantica di un modello è simile, ma la struttura **temporale** dei mondi cambia. Prima nei mondi consideravamo un modello LTL, qui invece le strutture sono **branching time**, ovvero sono fatte ad albero.

5.2.1 Semantica Informale

- I mondi come detto sono **branching time temporal structures** chiamate *time trees*
- I *rami* di un time tree rappresentano le scelte disponibili all'agente in ogni momento
- È possibile distinguere tra esecuzioni riuscite di eventi e i loro fallimenti;
- le formule temporali sono simili a quelle di CTL*, ossia ci sono state formulas e path formulas;
- Beliefs, goal, and intention sono modellate nel solito modo. In ogni situazione c'è un insieme di mondi (belief/goal/intention)-accessible

L'esempio del dentista

Modellato dal grafo 5.2. La cosa interessante è che abbiamo i mondi (quadrati), dove *belief, goal and intention* sono rappresentati da quadrati diversi. Tutti i mondi sono legati da relazioni (Belief and intentions) che in figura non si vedono. Per dare un'idea, considerando il mondo Belief (B), dove i B dell'agente sono che lui partendo dallo stato iniziale può eseguire

- azioni *d1*, dove il risultato sarebbe sentir male e avere il dente otturato
- azione *b*
- azione *d2*

Guardano i goal, questi sono due:

- l'azione *d1* può essere definita in un primo goal possibile con l'effetto *p*. La stessa azione può descrivere anche un secondo mondo dove non vale *p*. Quindi scegliere il *d1* potrebbe portare o in *p* o in *not p*, ma comunque non sa (come vista dalla semantica modale) dove andare e quindi non è possibile avere goal o intenzione che sia strettamente selezionata
- ma essere

Dato il goal si derivano le intenzioni e ad esempio dal primo mondo si va in *p,f*

APPUNTO PRESO AD MINCHIAM... ME SCAZZAVA

5.2.2 Il linguaggio

Il linguaggio estende le formule di stato di CTL* con:

- $\text{BEL}(\varphi)$, $\text{GOAL}(\varphi)$, $\text{INTEND}(\varphi)$
- *succeeds(e)*, *fails(e)*, *does(e)*, *succeeded(e)*, *failed(e)*, *done(e)*, dove *e* è un evento;

²Le intenzioni sono trattate come cittadini di prima classe: ci sono tre modalità, BEL, GOAL (Desire) and INTEND

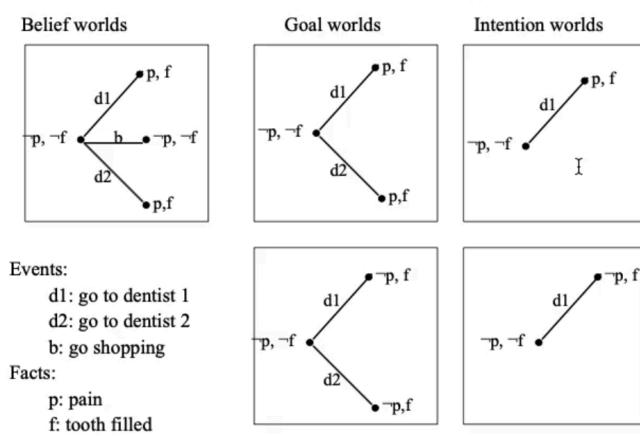


Figure 5.2: Esempio di dentista modellato secondo Rao e Georgeff

La formula $succeeded(e)$ al tempo t_1 denota l'esecuzione con successo dell'event e dal tempo t_0 a t_1 . Analogamente per $failed(e)$.

- $succeeds(e)$ può essere definito come $\mathbf{AX}succeeded(e)$ (l'evento è valido in tutti i cammini uscenti dall'istante di tempo in cui la formula è valutata). \mathbf{A} è un operatore di stato che per ogni cammino uscente dallo stato corrente deve valere la formula successiva

5.2.3 Commitments

Quello che possiamo fare è quella di riprendere il discorso sui commitment. Visti nella parte iniziale del corso. Un agente *blind commitment* è un agente che mantiene le sue intenzioni fino a quando non arriva a credere di averle soddisfatte. Se queste non saranno soddisfatte, l'agente continua ad andare avanti fino all'infinito. Commitment poco interessante. In questo caso, la formulazione del commitment è la seguente, quindi se vogliamo modellare un commitment fanatico, proseguiamo nel seguente modo

$$\text{INTEND}(\mathbf{AF}\varphi) \Rightarrow \mathbf{A}(\text{INTEND}(\mathbf{AF}\varphi)\mathbf{U} \text{BEL}(\varphi))$$

- I, A tutti i cammini, F prima o poi, φ sarà vero.
- blind commitments vuol dire che l'agente mantiene sempre lo stesso goal fino a quando riesce a soddisfarlo. Questo viene detto con l'*until* \mathbf{U} . Il primo argomento dell'*until* deve essere sempre vero fino a quando non arriva al risultato. In questo caso l'intenzione rimane sempre quella fino a quando non si arriva al punto (secondo argomento *until*) che φ sia creduto ($\text{BEL}(\varphi)$)

Stiamo parlando di commitment su formule, quindi si verifica φ , ma la stessa cosa può essere detta anche per le azioni (come visto nel linguaggio precedente, nell'esempio dello zio, dove uno può avere l'intenzione che è un'azione). Per concludere il discorso viene possibile modellare altre altri commitment. Con il *single-minded commitment*, conosciuto anche come commitment classico, un agente mantiene le sue intenzioni finché crede che queste possano essere realizzabili.

$$\text{INTEND}(\mathbf{AF}\varphi) \Rightarrow \mathbf{A}(\text{INTEND}(\mathbf{AF}\varphi)\mathbf{U}(\text{BEL}(\varphi) \vee \neg \text{BEL}(\mathbf{EF}\varphi)))$$

Mantendendo sempre questa intenzione con l'*until* si arriva a credere che φ sia vero oppure arrivare a non credere che prima o poi φ sarà vero. Quindi l'agente va avanti fino a quando o trova φ oppure si accorge che φ non possa essere raggiunti (seconda parte del \vee). Questa formulazione può essere leggermente modificata introducendo il termine *open-minded* che differisce dalla precedente in quanto anziché un BEL abbiamo un GOAL .

$$\text{INTEND}(\mathbf{AF}\varphi) \Rightarrow \mathbf{A}(\text{INTEND}(\mathbf{AF}\varphi)\mathbf{U}(\text{BEL}(\varphi) \vee \neg \text{GOAL}(\mathbf{EF}\varphi)))$$

L'agente mantiene le sue intenzioni fino a quando queste intenzioni rimangono ancora dei goal, quindi fino a quando uno non arriva a cancellare il goal.

Varianti della logica e conclusioni

In conclusione la modellazione delle proprietà della logica BDi è complicata da maneggiare. Difficile che questo possa essere utilizzato per modellare agenti basati. Gli autori hanno analizzato altri aspetti della logica, quindi diverse relazioni tra belief desire e intentions.

- Hanno considerato il caso in cui una relazione sia un sottoinsieme di un'altra,
- oppure che una relazione sia un sotto-mondo (sotto-albero) di un'altra relazione

Mettendo assieme queste relazioni, gli autori hanno dimostrato tutti questi aspetti e quindi studiato e proposto parecchie varianti di logica a seconda di quelle che sono le proprietà che valgono. Le proposte da loro fatte erano costruite sulla struttura base vista, ma con proprietà diverse utilizzate per modellare diversi tipi di agenti. Per certi aspetti è ragionevole, per altri no. Complessivamente quello visto porta alla conclusione che la **modellazione delle proprietà della logica BDI è complicata** da maneggiare. Difficile convincersi che quello visto possa essere utilizzato per modellare agenti basati sulla logica, perché le due logiche sono molto complicate e che quindi da un punto di vista formale sono interessanti, ma in pratica non sono mai state molto utilizzate

Chapter 6

Model Checking

Cerchiamo di capire cos'è e a cosa serve. Possiamo fare all'inizio riferimento a quello detto nella lezione precedente. Abbiamo descritto la struttura dell'agente utilizzando metodi formali (logiche modali di vario tipo). Abbiamo visto che utilizzare metodi formali per modellare agenti non è tanto semplice, ma è interessante. All'inizio della volta scorsa era che i metodi formali possono essere usati per tante cose diverse

- modellare e descrivere comportamenti d'agenti
- abbiamo citato anche altri aspetti importanti. In particolare, in questo capitolo, vedremo come, utilizzando i metodi formali, possiamo verificare proprietà di agenti o sistemi in generale. Quindi, quello che vogliamo fare è cercare di spiegare almeno qualcosa (argomento ahimè troppo complesso) di cos'è il **model checking** (mc) e come può essere realizzato.

Il tema sarà quello di verificare delle proprietà di un particolare sistema. Il model checking è un metodo (molto usato) per verificare proprietà di sistemi, o più generali di sistemi concorrenti, sistemi in cui ci possono essere più partecipanti che interagiscono fra di loro.

Il mc è un approccio basato sui modelli e quindi basato su un aspetto semantico.

- dato un modello M in una logica L
- e una formula φ di L

La verifica della proprietà vuol dire andare a determinare se φ è valida in M .

- normalmente questa logica L è una logica temporale¹, quindi il mc si riferisce a formule della logica temporale
- il mc può essere applicato quindi a situazioni in cui si usano le logiche temporali, sostanzialmente dei due tipi definiti

Il comportamento di un sistema (sistema per cui vogliamo verificare una proprietà), può essere formulato come un **transition system** π . Il transition system non è altro che un automa

- insieme di stati;
- insieme di trasizioni tra gli stati;
- funzione che associa ad ogni stato un insieme di proposizioni.

Si ricorda come questa descrizione corrisponde ai modelli della logica modale. Il transition system può essere visto come una **Kripke strucuture** M_π , ossia un modello della logica temporale (modale). Il nostro obiettivo è quello di ragionare su questo modello. Supponiamo che questo transition system descriva tutte le possibili computazioni di un agente o generico sistema.

¹LTL o CTL

Un esempio: Il forno a microonde

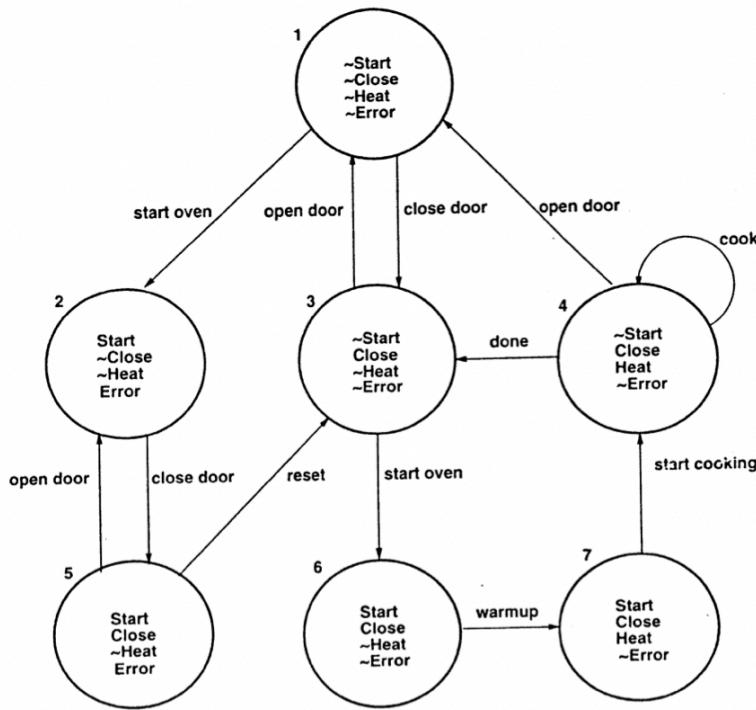


Figure 6.1: Esempio del Model Checking. Esempio di approccio

- abbiamo i nodi che rappresentano gli stati
- archi che collegano stati tra loro
- ogni stato contengono dei predicati che sono T or F.

Questo transition system, come da titolo rappresenta il funzionamento del microonde. Rispetto ad un modello della logica modale, le etichette applicate agli archi, di solito in un modello non ci sono. Il modello di solito è costituito da un insieme di mondi, dove ogni mondo contiene un qualcosa che può essere vero o falso senza che venga dato nessun nome su ogni arco

6.1 Il Model Checking

6.1.1 Model checking per LTL

Visto che usiamo la logica temporale, possiamo incominciare a parlare di come può essere realizzato il model checking. Per il momento ci limiteremo alla linear time logic.²

- supponiamo di avere un sistema π formulato come un transition system che descrive il funzionamento del sistema (proprio come nel caso del microonde)
- una formula LTL φ che descrive una proprietà che vogliamo verificare
- per mostrare che φ vale per π , dobbiamo dimostrare che φ è vera per ogni esecuzione di π

Siccome sappiamo che un modello in LTL è una sequenza infinita di stati, dimostrare che φ è vera per ogni esecuzione di π , significa che ogni cammino infinito di π è un modello di φ , quindi in ogni cammino di π , la formula φ è vera.

²considereremo solo questo caso, più che altro per mancanza di tempo.

Come procedere

In teoria, dovremmo ragionare su tutte le esecuzioni di π in modo che tutte soddisfino φ . Abbiamo detto che normalmente questo model checking si applica su sistemi che non termino mai. Se si fa riferimento al sistema microonde si vede che ci sono esecuzioni che non termina mai, ci sono tanti loop in cui il sistema continua eventualmente ad eseguire sequenze infinite. Ad esempio, vogliamo vedere se per il microonde vale la proprietà LTL:

- $\mathbf{G}(\text{Start} \Rightarrow \mathbf{F} \text{ Heat})$, G sempre vero e F significa prima o poi sarà vero. Se siamo in uno stato in cui c'è *start*, prima o poi arriveremo in uno stato in cui c'è *heat*. Per ogni cammino, da ogni stato in cui vale Start si raggiunge uno stato in cui vale Heat

E facile vedere che la formula non è vera per tutti i cammini. In figura vediamo che non è vero. Se considerassimo i mondi in questa sequenza: 1, 2, 5, 2... possiamo avere un RUN infinito che parte da 1 arriva in 2, per poi andare in 5 per poi ritornare in 2.

Non va bene rispetto a quello che volevamo dimostrare. Non è una proprietà corretta perché non è vera per tutti i cammini. Viceversa potremmo prendere un'altra formula

- $(\neg \text{Heat}) \mathbf{U} \text{ Close}$ è valida. L'**U**, fino a quando non si arriva a Close deve valere \neq Heat, quindi deve essere vero l'antecedente. In questo caso è impossibile che il forno diventi caldo finché la porta è aperta, e questo è effettivamente vero

Come verificare le proprietà?

Di solito, la dimostrazione che una formula φ è valida, ossia è vera per tutti i cammini infiniti del modello, viene fatta per **refutazione**:

- dimostrare che $\neg\varphi$ è insoddisfacibile nel modello, ovvero non c'è nessuna computazione che soddisfi $\neg\varphi$

In altre parole, se si trova un cammino che soddisfa $\neg\varphi$, questo costituisce un **controesempio**³ che contraddice la validità di φ che stavamo cercando.

Ad esempio, sempre facendo riferimento alla figura, si può vedere che il cammino infinito 1, 2, 5, 2, ... soddisfa la formula $\neg\mathbf{G}(\text{Start} \Rightarrow \mathbf{F} \text{ Heat})$ e quindi è un controesempio per la validità della formula $\mathbf{G}(\text{Start} \Rightarrow \mathbf{F} \text{ Heat})$.

6.1.2 LTL e automi su stringhe infinite

Per capire come usare la logica LTL per verificare transition system, di deve necessariamente passare attraverso gli automi di Büchi, ossia automi che accettano stringhe infinite.

- L'automa $\langle \Sigma, S, \delta, S_0, F \rangle$
- alfabeto Σ
- insieme di stati S ;
- una funzione di transizione $\delta \in S \times \Sigma \times S$
- un insieme di stati iniziale S_0
- insieme di **stati di accettazione** F . Unica differenza con l'automa a stati finiti è la terminazione, quindi viene accetta una sequenza. L'automa a stati finiti si ferma, mentre questo accetta stringhe infinite, quindi praticamente non si ferma mai

Una stringa infinita (**run**) w è accettata dall'automa se è compatibile con δ (è un cammino infinito nel grafo dell'automa) e almeno uno stato in F appare infinite volte in w .

Per trovare una stringa infinita, basta trovare un *run* (percorso) sull'automa che esegue un ciclo infinito.

³qualche cosa che non è valido, contraddice la validità

Model checking e automi di Buchi

Sia il transition system che la formula π possono essere entrambi riportati all'automa di Buchi

- infatti il transition system di un sistema π , ossia il modello, può essere visto come un automa di Buchi(π) *i cui stati sono tutti di accettazione*. Se prendiamo l'esempio del microonde possiamo vederlo come un modello di Buchi, in cui tutti gli stati sono di accettazione. Rispetto a quello che dicevamo prima che un run infinito può essere rappresentato come un run che termina in un ciclo, in questo caso, visto che tutti gli stati sono di accettazione, qualunque run di questo sistema π che finisce in un loop va sempre bene. Possiamo definire dei run che terminano in un loop in qualunque parte del modello. Quindi, banale vedere che il sistema π può essere visto come automa di Buchi(π)
- è stato dimostrato che una formula LTL può essere tradotta in un automa di Buchi. Parte interessante non affrontata durante il corso, ma la prendiamo come buona. Quindi, possibile ricavare automa di Buchi(π) che accetta esattamente tutte le sequenze infinite che sono modelli di π

6.1.3 Procedura per il model checking

- siano dati il transition system π e la formula LTL φ da verificare. Entrambi danno la possibilità di ottenere un automa di Buchi.
- costruire i due automi di Buchi $B(\pi)$ e $B(\neg \varphi)$, quest'ultimo accetterà tutti i modelli della formula φ
- costruire l'automa di Buchi che accetta l'**intersezione** dei linguaggi accettanti da $B(\pi)$ e $B(\neg \varphi)$ (il prodotto dei due automi di Buchi). Il risultato sarà un automa prodotto e questo sarà quello dove noi andremo a ragionare se φ è vera o non vera.
- ogni run dell'intersezione è un run infinito di π sia un modello di φ . Siccome questo automa è stato ottenuto mettendo assieme i due automi (quelli del punto 2), ogni run, per come è fatta l'intersezione è sia un run infinito di π sia un modello di $\neg \varphi$. Ogni run dell'intersezione è un modello di entrambi, ovvero sia del transition system che della formula φ
- se l'intersezione è vuota, allora φ vale per π , altrimenti un run dell'intersezione fornisce un **controesempio**. Se l'intersezione è vuota, vuol dire che non c'è nessun run del prodotto che sia un run di tutti e due gli automi, allora φ sarà vera per π , altrimenti se c'è un run che comprende entrambi siamo in una situazione in cui troviamo un run che sia di π che di $\neg \varphi$ e questo fornisce un controesempio.

Il problema di questo approccio è quello che richiede la costruzione di questo prodotto. Una volta trovato il prodotto, dobbiamo poter ragionare su di esso e quindi cercare un cammino sull'automa di Buchi richiede parecchio lavoro, ovvero richiede un'esecuzione lineare rispetto alle dimensioni dell'automa.

- Per trovare un controesempio è sufficiente trovare un cammino dell'automa intersezione che termina con un loop.

Esempio di applicazione

Tornando all'esempio del microonde. Facciamo il prodotto dell'automa microonde con l'automa ottenuto dalla formula negata $\neg \mathbf{G}(\text{Start} \Rightarrow \mathbf{F} \text{ Heat})$. Il problema di utilizzare questo approccio è nel fare il prodotto di questi due automi. Ci potrebbe essere una crescita esponenziale. L'automa risultante è **non deterministico**, dove con la stessa azione si può

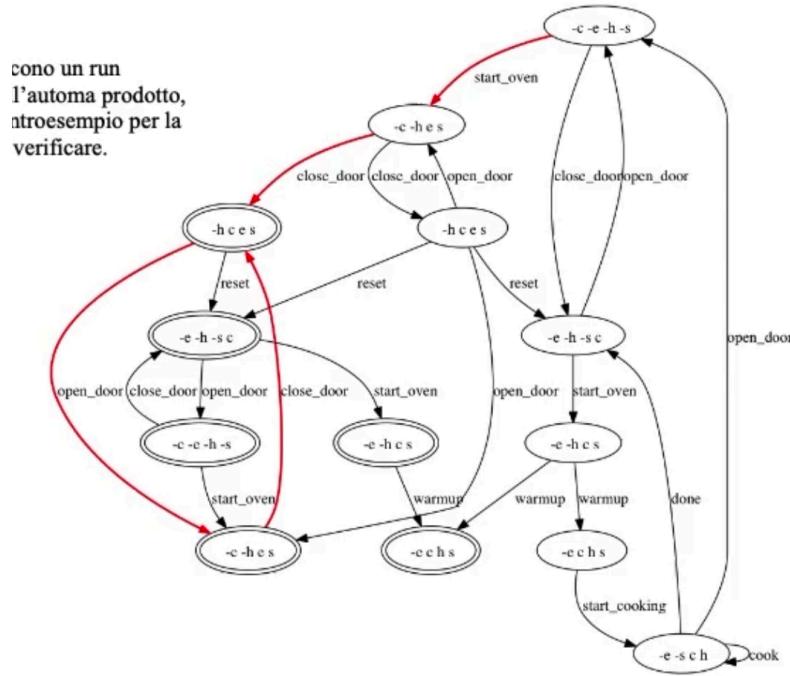


Figure 6.2: Prodotto di due automi

andare in nodi diversi.

Troviamo un cammino che finisce in un loop (6.2, cammino rosso), questo è quello che non va proprio bene. Quindi questo cammino rappresenta un controesempio per la formula che si voleva verificare, quindi si ottiene un cammino che non è un **run** della formula che volevamo verificare

6.1.4 Conclusioni

Per concludere l'argomento, possiamo dire che con il model checking esiste un metodo per verificare la proprietà dei sistemi. Quello interessante è usare lo stesso approccio per fare Planning, ovvero trovare sempre nell'infinito cammini che modellino la pianificazione. Se prendiamo il solito automa π e la formula φ che rappresenta il goal, tutti i run infiniti dell'automa prodotto di $B(\pi)$ e $B(\varphi)$ soddisfano φ e quindi rappresentano un piano sequenziale.

Conclusione che l'algoritmo di prima può essere utilizzato sia per trovare dei controesempi, in modo da verificare che qualche proprietà non è valida, ma essere usato anche dando la formula in positivo (anzichè in negativo) per generare dei piani infiniti-

- ad esempio, la formula $\mathbf{F} \text{ on}(a,b)$ rappresenta un **achievement goal**: esiste un cammino con uno stato in cui vale $\text{on}(a,b)$

Se si vuola ottenere un piano di lunghezza finita con questo approccio, è **sufficiente** aggiungere nello stato finale un loop fittizio.

6.2 Model Checkin in SPIN

SPIN, è un tool per la *verifica dei sistemi concorrenti* basato su tecniche di model checking:

- la specifica del modello è data nel linguaggio **Promela** che consente di definire sistemi distribuiti mediante un insieme di processi formulati in un pseudo codice C, che consente l'uso di primitive di sincronizzazione e scambio di messaggi
- la proprietà da verificare è una formula in linear time temporal logic (LTL)

6.2.1 Model Checker in CTL

In realtà, molti model checker sono fatti in modo diversi, basati su criteri diversi. Visto che abbiamo parlato di logiche temporali CTL, accenniamo al fatto che ci sono dei model check che verificano le formule utilizzando CTL. È un approccio interessante che si basa sul ragionare direttamente sul transition system, quindi non costruisce grafi complicati e molto pesanti dal punto di vista computazionale come succedeva con LTL. In questo caso il model checker ragiona sull'automa (transition system) e il problema principale per fare la verifica è decomporre questa struttura di Kripke in componenti strettamente collegati tra loro). Approccio più efficienti di quello descritto in precedenza. In realtà, più efficiente non significa niente. Quando abbiamo parlato di queste logiche temporali, avevamo osservato che le formule trattate da CTL e LTL non erano confrontabili.

In ogni caso, il model checkin è un buon esempio di come strumenti basati sulla logica possano essere usati con successo per verificare proprietà di sistemi reali usati in ambiente industriale (SPIN).

Chapter 7

Agenti Ibridi e Reattivi

Agenti reattivi, agenti visti quando sono state illustrate le varie proprietà (con Baldoni, sono le primissime lezioni). In questo capitolo tratteremo la **gestione** e **realizzazione** di agenti reattivi e ibridi

7.1 Architetture Reattive

- Ci sono molti problemi irrisolti associati all'**IA simbolica**. Quindi problemi riscontrati nelle architetture ad agenti che sono difficili da realizzare utilizzando l'approccio classico.
- molti ricercatori sono arrivati a discutere e criticare questo paradigma di IA simbolica facendo vedere, quindi sviluppando tecniche o approcci diversi alla realizzazione di agenti, sviluppand **architetture reattive**.
- in questa presentazione (capitolo) iniziamo discutendo l'attività di uno dei più comunicativi ricercatori nel settore: Rodney Brooks. Il mitico Rodney a fine anni 80 inizio anni 90 si è occupato di problemi di questo genere.

7.1.1 L'approccio di Brooks

Brooks sosteneva che un comportamento intelligente può:

1. essere generato **senza rappresentazione** esplicita del tipo di ciò che è proposto dall'IA simbolica
2. essere generato **senza ragionamento** esplicito del tipo di ciò che è proposto dall'IA simbolica
3. l'intelligenza è una proprietà **emergente** di certi sistemi complessi

L'approccio classico alle architetture degli agenti è che la logica può essere usato per rappresentare *conoscenza*, attraverso l'uso delle sue formule e dall'altro a ragionare attraverso dei meccanismi di inferenza definiti per quella particolare logica.

Brooks identifica due idee chiave

Brooks dice che:

- per definire l'intelligenza, questa non è che venga fuori definendo un *theorem prover* o sistemi esperti, ma viene fuori dall'essere situati in un mondo, quindi avere un ambiente in cui l'agente viene situato;

- il comportamento di attribuire o no intelligenza ad un agente non è legato al fatto che ciascuno di questi agenti abbia una sua architettura particolare, ma bisogna guarda cosa fa. L'intelligenza sta negli occhi di chi guarda e quindi non è una proprietà innata e isolata.

Nel suo articolo viene riportato un esempio.

Viene preso un problema classico dell'AI. Problema che va sotto il titolo di **scimmia e la banana**. Supponiamo di sapere cosa sia una sedia e questa è appunto una sedia dove una persona si può sedere e mettersi su. Possiamo usare questa conoscenza per ragionare. Il ragionare è che una persona è seduta su una di queste sedie e vede sulla propria testa un casco di banane e per risolvere tale problema, trovo un piano che consiste nel *salire su una sedia, prendere la banana e mangiarla*. In sostanza dice che per capire se una persona (agente) è intelligente bisogna mettersi la e guardare cosa fa.

7.1.2 Architettura di Brooks

Brooks per illustrare le sue idee ha definito un'architettura particolare che serve per modellare e realizzare agenti reattivi. Tale architettura prende il nome di **subsumption architecture**.

- Architettura adeguata per eseguire tasks sui robot, ma che magari con altri problemi tratta con metodo simboli dell'intelligenza artificiale non siano tanto gestibili con il suo approccio.

Non è detto quindi che qualunque agente in qualunque modo debba essere realizzato con un'architettura anzichè un'altra.

- la sua architettura è una gerarchia di **task** che eseguono dei **behaviors**. Queste sono regole semplici che **eseguono** dei task (compiti da realizzare).
- i **behaviors**, competono tra loro. Sono essenzialmente regole messa a strati uno sopra l'altro, dove gli stati più bassi rappresentano tipi di behaviors più primitivi (come evitare ostacoli. Classico esempio dell'agente reattivo) e man mano avendo questa stratificazione di livelli, i vari behaviors partendo dal basso, hanno precedenza l'uno sull'altro (priorità alta livello basso).

I sistemi risultanti sono estremamente semplici.

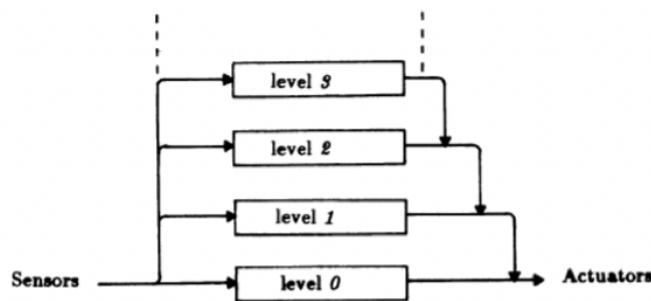


Figure 7.1: Architettura Reattiva

- per ogni passo bisogna decidere, in base a quello che si riceve dall'esterno (*Sensors*), quale dei livelli (*Level*) deve scattare, quindi quale regola deve scattare.

7.1.3 Selezione delle azioni

La scelta di un'azione è realizzata attraverso un insieme di behaviors. Un **behavior** non è altro che una coppia (c,a) , dove

- c è un insieme di percezioni dette **condizioni**
- a è un'azione

Un behaviors (c,a) scatta quando la funzione sse restituisce una percezione p , tale che $p \in c$.

- associata a un insieme di behavior rules c'è una **relazione di inhibition** $<$
- leggiamo $b_1 < b_2$ come " b_1 inibisce b_2 ", cioè b_1 è più basso nella gerarchia di b_2 , e quindi ha la priorità su b_2

7.1.4 Un esempio: Steels' Mars Explorer (1990)

Il sistema di esplorazione di Marte proposto da Steels usando la subsumption architecture, ottiene una performance cooperativa quasi ottima nel dominio raccolta di rocce su Marte': L'obiettivo è di esplorare un pianeta distante, e in particolare, di raccogliere campioni di una roccia preziosa. La posizione dei campioni non è conosciuta in anticipo, ma si sa che tendono ad essere raggruppati. La soluzione fa uso di due meccanismi:

- un campo **gradiente**. La postazione di base genera un segnale radio, che si riduce man mano che la distanza dalla sorgente cresce. Per trovare la direzione verso la base, un agente deve solo muoversi seguendo il gradiente del segnale.
- Per permettere a piu agenti di comunicare fra di loro, gli agenti usano "**briciole radioattive**", che possono essere lasciate cadere o raccolte. Pollicino vi dice qualcosa?

Steels' Mars Explorer Rules

Per agenti individuali (non-cooperativi), il behaviour di livello piu basso (e quindi con la priorita piu alta) è di evitare ostacoli:

1. *if detect an obstacle then change direction*

I campioni portati da agenti sono depositati alla base:

2. *if carrying samples and at the base then drop samples*

Gli agents che portano campioni ritornano alla base:

3. *if carrying samples and not at the base then travel up gradient*

Gli agenti raccolgono i campioni che trovano:

4. *if detect a sample then pick sample up*

Un agente che non sa cosa fare esplora a caso:

5. *if true then move randomly*

Ordine dei behavior:

- $1 < 2 < 3 < 4 < 5$

7.1.5 Steels' Mars Explorer: cooperation

- L'idea e che un agente crei un sentiero di briciole radioattive ogni volta che trova un campione di roccia, per comunicarlo ad altri agenti.
- Il sentiero sara creato quando l'agente riporta le rocce alla base.
- Mentre un agente segue un sentiero verso la sorgente delle rocce, raccoglie le briciole che trova, rendendo il sentiero piu sottile.
- Dopo che un po' di agenti hanno seguito il sentiero senza trovare campioni, il sentiero sara rimosso.

Le regole 1, 2, 4 e 5 rimangono inalterate. Le seguenti regole sono aggiunte:

6. *if carrying samples and not at the base then drop 2 crumbs and travel up gradient*
7. *if sense crumbs then pick up 1 crumb and travel down gradient*

Ordine dei behavior:

- $1 < 2 < 6 < 4 < 7 < 5$

7.1.6 Vantaggi degli agenti reatti

- Semplicita
- Economia
- Trattabilita computazionale
- Robustezza verso i fallimenti
- Eleganza

Come abbiamo visto vengono fuori (esempio di Steels) sorprendenti. Soluzione abbastanza elegante.

7.1.7 Limiti degli Agenti Reattivi

- Agenti senza modello dell'ambiente devono avere sufficienti informazioni disponibili localmente
- Se le decisioni sono basate sull'ambiente locale, come puo l'agente tenere conto dell'informazione non-locale (ha una visione a breve termine)
- Difficile realizzare agenti reattivi che apprendono
- Visto che i behaviour emergono da interazioni fra componenti piu ambiente, e difficile vedere come ingegnerizzare agenti specifici (non esistono metodologie generali)
- È difficile definire agenti con un gran numero di behaviour (le dinamiche delle interazioni diventano troppo complesse da capire)

7.2 Architetture Ibride

Molti ricercatori hanno cercato di fondere assieme sia architetture di agenti reattivi che aspetti che facciano riferimento ad aspetti di intelligenza artificiale classica. Questo approccio è noto come **architettura ibrida**. L'approccio è quello di costruire un agente con due o più sottosistemi

- uno **deliberativo**, contenente un modello simbolico del mondo, che sviluppa piani e prende decisioni nel modo proposto dall'IA simbolica;
- uno **reattivo**, capace di reagire ad eventi senza ragionamenti complessi

Spesso, a componenti *reattivi* viene data precedenza rispetto a quelli *deliberativi*. Questo tipo di struttura (quella ibrida) porta naturalmente all'idea di un'architettura a strati (*layered*) di cui **TOURINGMACHINES** e **INTERRAP** sono esempi. In questa architettura, i sottosistemi di controllo di un agente sono organizzati in una gerarchia, con livelli più alti che trattano l'informazione a crescenti livelli di astrazione.

Un problema importante in queste architetture è in che tipo di schema di controllo inserire i sottosistemi dell'agente per tratte le interazioni fra i vari stati. Sono stati proposti:

- **horizontal layering**, dove ognuno di questi livelli è gestito per conto suo. Nella parte sinistra troviamo le percezioni *perceptual input* che entrano e vanno a finire in tutti i livelli (*layers*). Da questi escono le azioni (*action output*). Una cosa fatta in questo modo non funziona, perché in base all'input dovremmo decidere quale sia il livello che scatta per primo e poi non è che possiamo eseguire tutte le azioni in un colpo solo. Comunque, si noti come in questo tipo di architettura deve esserci un **controllore** che controlli appunto cosa entra e cosa esce dai vari livelli;
- **vertical layer**, i livelli vengono tutti attraversati. Input dei sensori e output delle azioni sono gestiti al massimo da uno strato ciascuno. Una versione variante potrebbe essere quella dove ci sono due passate dal basso per poi ritornare indietro.

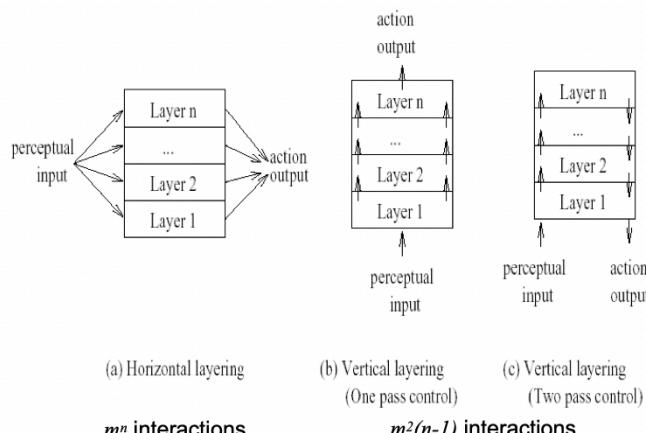


Figure 7.2: Architettura Ibrida

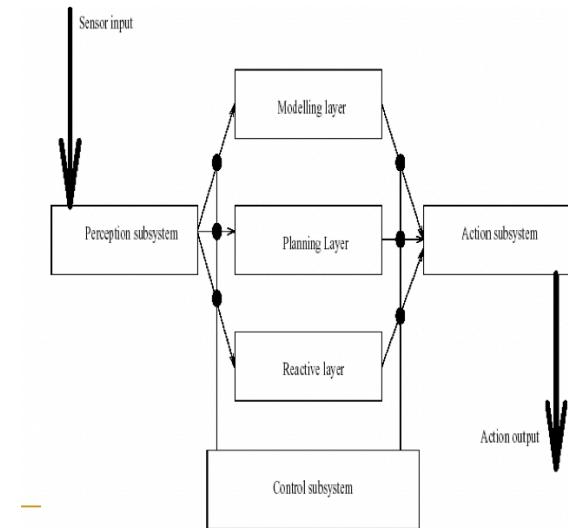


Figure 7.3: Esempio di architettura TOURINGMACHINES

7.2.1 Ferguson - TOURINGMACHINES

L'architettura di TOURINGMACHINES consiste di sottosistemi di **percezione** e **azione**, che interfacciano direttamente con l'ambiente dell'agente, e tre **control layers**, inseriti in un **control framework**, che media fra gli strati.

- Lo strato **reactive** è implementato come un insieme di regole situation-action, a la subsumption architecture
- the **planning layer** costruisce piani e sceglie azioni da eseguire, in modo da raggiungere i goal dell'agente;
- il **modelling layer** contiene la rappresentazione simbolica degli "stati cognitivi" di altre entità nell'ambiente dell'agente
- i tre layer comunicano tra di loro e sono inseriti in un "control framework", che usa **"control rules"**

Per un sunto di quella che è l'architettura TOURINGMACHINES

Facendo riferimento a 8.2,

- il **sistema di controllo** (*Control system*) serve per gestire come detto prima cosa entra e cosa esce;
- **reactive layer**, quello più in basso, è implementato come un insieme di regole *situation-action*;
- sopra a questi ci sono livelli che sono più legati all'approccio standard. Abbiamo il **planning** che esegue piani e **modelling** che contiene una rappresentazione simbolica di quello che è l'ambiente.

7.2.2 Muller InteRRaP

Altro agente ibrido. È di tipi vertically layered, dove le frecce salgono e scendono. Il contenuto non è tanto diverso da quello di prima.

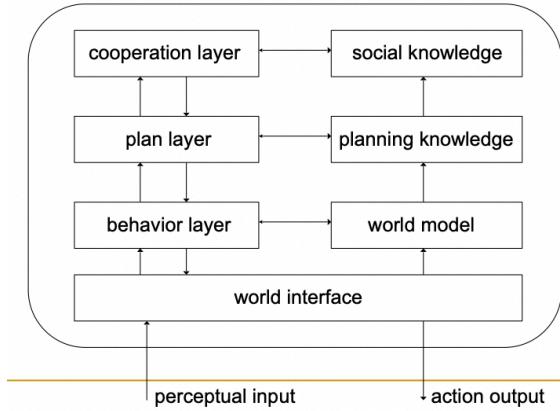


Figure 7.4: Esempio di architettura Muller InteRRaP

7.3 Conclusioni

Approccio basato su agenti reattivi abbiamo visto che ha degli effetti interessanti, può consentire (almeno negli esempi visti) di realizzare un esempio interessante dove con poca spesa da una soluzione interessante perché non ci sono tutti gli eventuali componenti che possono costituire un sistema classico dell'intelligenza artificiale. Quindi ci sono sia aspetti negativi e aspetti positivi. L'esempio di Marte è molto lontano dalla realtà anche se in effetti, il problema del Rover che gira su Marte è reale, ed effettivamente l'AI è utilizzata proprio per problemi di questo genere. Per comunicare dalla Terra a Marte ci vuole un fottio, proprio per qui è inevitabile che la costruzione dei robot sia ottimizzata.

Un esempio di agente reattivo è l'aspirapolvere. È agente apparentemente intelligente che soddisfa l'idea di Brooks. Se vediamo l'aspirapolvere che gira in un ambiente dove ci sono ostacoli naturalmente siamo in un contesto di questo tipo e possiamo dire quindi che certi oggetti che vengono utilizzati comunemente sono realizzati con architetture presentate in questo capitolo. Proprio per cui possiamo dire che certi oggetti che vengono usati comunemente, sono realizzati tramite architetture di questo tipo. Un robot funziona bene perché ha tanti sensori, sensori che è in grado di gestirli. Che poi al suo interno abbia qualche componente che ha atteggiamento/comportamento che possa essere ritenuto intelligente può anche esserlo, però in un oggetto (agente) di questo tipo è difficile vedere intelligenza o vedere dove vengono usati aspetti intelligenti.

Nell'esempio della macchina il problema non è tanto eseguire azioni ma è la percezione.

Chapter 8

Agenti Ibridi e Reattivi

Agenti reattivi, agenti visti quando sono state illustrate le varie proprietà (con Baldoni, sono le primissime lezioni). In questo capitolo tratteremo la **gestione** e **realizzazione** di agenti reattivi e ibridi

8.1 Architetture Reattive

- Ci sono molti problemi irrisolti associati all'**IA simbolica**. Quindi problemi riscontrati nelle architetture ad agenti che sono difficili da realizzare utilizzando l'approccio classico.
- molti ricercatori sono arrivati a discutere e criticare questo paradigma di IA simbolica facendo vedere, quindi sviluppando tecniche o approcci diversi alla realizzazione di agenti, sviluppand **architetture reattive**.
- in questa presentazione (capitolo) iniziamo discutendo l'attività di uno dei più comunicativi ricercatori nel settore: Rodney Brooks. Il mitico Rodney a fine anni 80 inizio anni 90 si è occupato di problemi di questo genere.

8.1.1 L'approccio di Brooks

Brooks sosteneva che un comportamento intelligente può:

1. essere generato **senza rappresentazione** esplicita del tipo di ciò che è proposto dall'IA simbolica
2. essere generato **senza ragionamento** esplicito del tipo di ciò che è proposto dall'IA simbolica
3. l'intelligenza è una proprietà **emergente** di certi sistemi complessi

L'approccio classico alle architetture degli agenti è che la logica può essere usato per rappresentare *conoscenza*, attraverso l'uso delle sue formule e dall'altro a ragionare attraverso dei meccanismi di inferenza definiti per quella particolare logica.

Brooks identifica due idee chiave

Brooks dice che:

- per definire l'intelligenza, questa non è che venga fuori definendo un *theorem prover* o sistemi esperti, ma viene fuori dall'essere situati in un mondo, quindi avere un ambiente in cui l'agente viene situato;

- il comportamento di attribuire o no intelligenza ad un agente non è legato al fatto che ciascuno di questi agenti abbia una sua architettura particolare, ma bisogna guarda cosa fa. L'intelligenza sta negli occhi di chi guarda e quindi non è una proprietà innata e isolata.

Nel suo articolo viene riportato un esempio.

Viene preso un problema classico dell'AI. Problema che va sotto il titolo di **scimmia e la banana**. Supponiamo di sapere cosa sia una sedia e questa è appunto una sedia dove una persona si può sedere e mettersi su. Possiamo usare questa conoscenza per ragionare. Il ragionare è che una persona è seduta su una di queste sedie e vede sulla propria testa un casco di banane e per risolvere tale problema, trovo un piano che consiste nel *salire su una sedia, prendere la banana e mangiarla*. In sostanza dice che per capire se una persona (agente) è intelligente bisogna mettersi la e guardare cosa fa.

8.1.2 Architettura di Brooks

Brooks per illustrare le sue idee ha definito un'architettura particolare che serve per modellare e realizzare agenti reattivi. Tale architettura prende il nome di **subsumption architecture**.

- Architettura adeguata per eseguire tasks sui robot, ma che magari con altri problemi tratta con metodo simboli dell'intelligenza artificiale non siano tanto gestibili con il suo approccio.

Non è detto quindi che qualunque agente in qualunque modo debba essere realizzato con un'architettura anzichè un'altra.

- la sua architettura è una gerarchia di **task** che eseguono dei **behaviors**. Queste sono regole semplici che **eseguono** dei task (compiti da realizzare).
- i **behaviors**, competono tra loro. Sono essenzialmente regole messa a strati uno sopra l'altro, dove gli stati più bassi rappresentano tipi di behaviors più primitivi (come evitare ostacoli. Classico esempio dell'agente reattivo) e man mano avendo questa stratificazione di livelli, i vari behaviors partendo dal basso, hanno precedenza l'uno sull'altro (priorità alta livello basso).

I sistemi risultanti sono estremamente semplici.

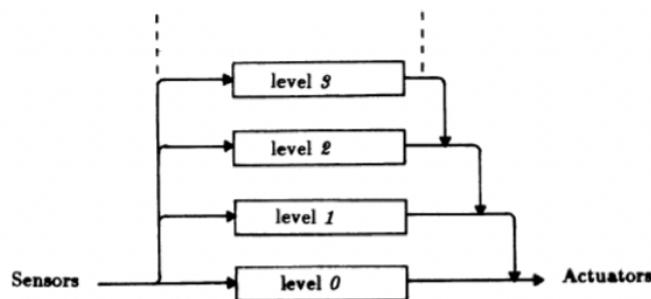


Figure 8.1: Architettura Reattiva

- per ogni passo bisogna decidere, in base a quello che si riceve dall'esterno (*Sensors*), quale dei livelli (*Level*) deve scattare, quindi quale regola deve scattare.

8.1.3 Selezione delle azioni

La scelta di un'azione è realizzata attraverso un insieme di behaviors. Un **behavior** non è altro che una coppia (c,a) , dove

- c è un insieme di percezioni dette **condizioni**
- a è un'azione

Un behaviors (c,a) scatta quando la funzione sse restituisce una percezione p , tale che $p \in c$.

- associata a un insieme di behavior rules c'è una **relazione di inhibition** $<$
- leggiamo $b_1 < b_2$ come " b_1 inibisce b_2 ", cioè b_1 è più basso nella gerarchia di b_2 , e quindi ha la priorità su b_2

8.1.4 Un esempio: Steels' Mars Explorer (1990)

Il sistema di esplorazione di Marte proposto da Steels usando la subsumption architecture, ottiene una performance cooperativa quasi ottima nel dominio raccolta di rocce su Marte': L'obiettivo è di esplorare un pianeta distante, e in particolare, di raccogliere campioni di una roccia preziosa. La posizione dei campioni non è conosciuta in anticipo, ma si sa che tendono ad essere raggruppati. La soluzione fa uso di due meccanismi:

- un campo **gradiente**. La postazione di base genera un segnale radio, che si riduce man mano che la distanza dalla sorgente cresce. Per trovare la direzione verso la base, un agente deve solo muoversi seguendo il gradiente del segnale.
- Per permettere a piu agenti di comunicare fra di loro, gli agenti usano "**briciole radioattive**", che possono essere lasciate cadere o raccolte. Pollicino vi dice qualcosa?

Steels' Mars Explorer Rules

Per agenti individuali (non-cooperativi), il behaviour di livello piu basso (e quindi con la priorita piu alta) è di evitare ostacoli:

1. *if detect an obstacle then change direction*

I campioni portati da agenti sono depositati alla base:

2. *if carrying samples and at the base then drop samples*

Gli agents che portano campioni ritornano alla base:

3. *if carrying samples and not at the base then travel up gradient*

Gli agenti raccolgono i campioni che trovano:

4. *if detect a sample then pick sample up*

Un agente che non sa cosa fare esplora a caso:

5. *if true then move randomly*

Ordine dei behavior:

- $1 < 2 < 3 < 4 < 5$

8.1.5 Steels' Mars Explorer: cooperation

- L'idea e che un agente crei un sentiero di briciole radioattive ogni volta che trova un campione di roccia, per comunicarlo ad altri agenti.
- Il sentiero sara creato quando l'agente riporta le rocce alla base.
- Mentre un agente segue un sentiero verso la sorgente delle rocce, raccoglie le briciole che trova, rendendo il sentiero piu sottile.
- Dopo che un po' di agenti hanno seguito il sentiero senza trovare campioni, il sentiero sara rimosso.

Le regole 1, 2, 4 e 5 rimangono inalterate. Le seguenti regole sono aggiunte:

6. *if carrying samples and not at the base then drop 2 crumbs and travel up gradient*
7. *if sense crumbs then pick up 1 crumb and travel down gradient*

Ordine dei behavior:

- $1 < 2 < 6 < 4 < 7 < 5$

8.1.6 Vantaggi degli agenti reatti

- Semplicita
- Economia
- Trattabilita computazionale
- Robustezza verso i fallimenti
- Eleganza

Come abbiamo visto vengono fuori (esempio di Steels) sorprendenti. Soluzione abbastanza elegante.

8.1.7 Limiti degli Agenti Reattivi

- Agenti senza modello dell'ambiente devono avere sufficienti informazioni disponibili localmente
- Se le decisioni sono basate sull'ambiente locale, come puo l'agente tenere conto dell'informazione non-locale (ha una visione a breve termine)
- Difficile realizzare agenti reattivi che apprendono
- Visto che i behaviour emergono da interazioni fra componenti piu ambiente, e difficile vedere come ingegnerizzare agenti specifici (non esistono metodologie generali)
- È difficile definire agenti con un gran numero di behaviour (le dinamiche delle interazioni diventano troppo complesse da capire)

8.2 Architetture Ibride

Molti ricercatori hanno cercato di fondere assieme sia architetture di agenti reattivi che aspetti che facciano riferimento ad aspetti di intelligenza artificiale classica. Questo approccio è noto come **architettura ibrida**. L'approccio è quello di costruire un agente con due o più sottosistemi

- uno **deliberativo**, contenente un modello simbolico del mondo, che sviluppa piani e prende decisioni nel modo proposto dall'IA simbolica;
- uno **reattivo**, capace di reagire ad eventi senza ragionamenti complessi

Spesso, a componenti *reattivi* viene data precedenza rispetto a quelli *deliberativi*. Questo tipo di struttura (quella ibrida) porta naturalmente all'idea di un'architettura a strati (*layered*) di cui **TOURINGMACHINES** e **INTERRAP** sono esempi. In questa architettura, i sottosistemi di controllo di un agente sono organizzati in una gerarchia, con livelli più alti che trattano l'informazione a crescenti livelli di astrazione.

Un problema importante in queste architetture è in che tipo di schema di controllo inserire i sottosistemi dell'agente per tratte le interazioni fra i vari stati. Sono stati proposti:

- **horizontal layering**, dove ognuno di questi livelli è gestito per conto suo. Nella parte sinistra troviamo le percezioni *perceptual input* che entrano e vanno a finire in tutti i livelli (*layers*). Da questi escono le azioni (*action output*). Una cosa fatta in questo modo non funziona, perché in base all'input dovremmo decidere quale sia il livello che scatta per primo e poi non è che possiamo eseguire tutte le azioni in un colpo solo. Comunque, si noti come in questo tipo di architettura deve esserci un **controllore** che controlli appunto cosa entra e cosa esce dai vari livelli;
- **vertical layer**, i livelli vengono tutti attraversati. Input dei sensori e output delle azioni sono gestiti al massimo da uno strato ciascuno. Una versione variante potrebbe essere quella dove ci sono due passate dal basso per poi ritornare indietro.

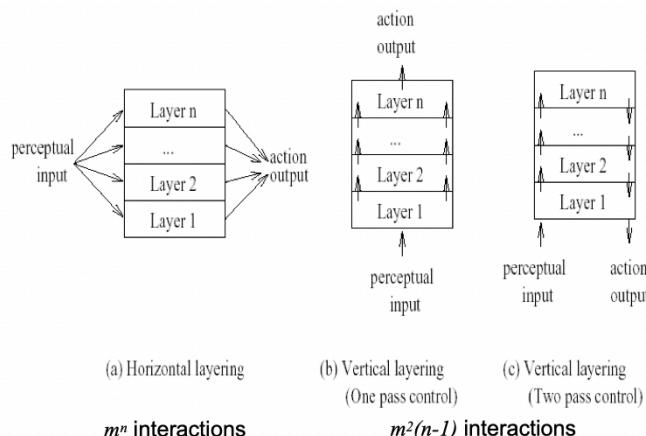


Figure 8.2: Architettura Ibrida

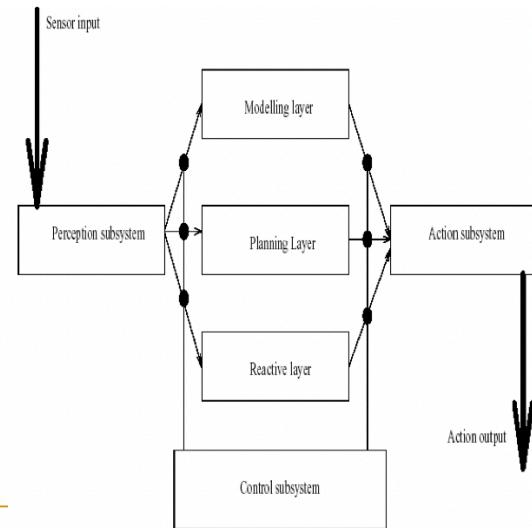


Figure 8.3: Esempio di architettura TOURINGMACHINES

8.2.1 Ferguson - TOURINGMACHINES

L'architettura di TOURINGMACHINES consiste di sottosistemi di **percezione** e **azione**, che interfacciano direttamente con l'ambiente dell'agente, e tre **control layers**, inseriti in un **control framework**, che media fra gli strati.

- Lo strato **reactive** è implementato come un insieme di regole situation-action, a la subsumption architecture
- the **planning layer** costruisce piani e sceglie azioni da eseguire, in modo da raggiungere i goal dell'agente;
- il **modelling layer** contiene la rappresentazione simbolica degli "stati cognitivi" di altre entità nell'ambiente dell'agente
- i tre layer comunicano tra di loro e sono inseriti in un "control framework", che usa **"control rules"**

Per un sunto di quella che è l'architettura TOURINGMACHINES

Facendo riferimento a 8.2,

- il **sistema di controllo** (*Control system*) serve per gestire come detto prima cosa entra e cosa esce;
- **reactive layer**, quello più in basso, è implementato come un insieme di regole *situation-action*;
- sopra a questi ci sono livelli che sono più legati all'approccio standard. Abbiamo il **planning** che esegue piani e **modelling** che contiene una rappresentazione simbolica di quello che è l'ambiente.

8.2.2 Muller InteRRaP

Altro agente ibrido. È di tipi vertically layered, dove le frecce salgono e scendono. Il contenuto non è tanto diverso da quello di prima.

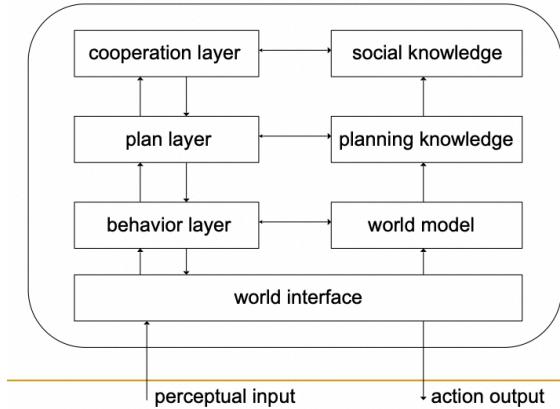


Figure 8.4: Esempio di architettura Muller InteRRaP

8.3 Conclusioni

Approccio basato su agenti reattivi abbiamo visto che ha degli effetti interessanti, può consentire (almeno negli esempi visti) di realizzare un esempio interessante dove con poca spesa da una soluzione interessante perché non ci sono tutti gli eventuali componenti che possono costituire un sistema classico dell'intelligenza artificiale. Quindi ci sono sia aspetti negativi e aspetti positivi. L'esempio di Marte è molto lontano dalla realtà anche se in effetti, il problema del Rover che gira su Marte è reale, ed effettivamente l'AI è utilizzata proprio per problemi di questo genere. Per comunicare dalla Terra a Marte ci vuole un fottio, proprio per qui è inevitabile che la costruzione dei robot sia ottimizzata.

Un esempio di agente reattivo è l'aspirapolvere. È agente apparentemente intelligente che soddisfa l'idea di Brooks. Se vediamo l'aspirapolvere che gira in un ambiente dove ci sono ostacoli naturalmente siamo in un contesto di questo tipo e possiamo dire quindi che certi oggetti che vengono utilizzati comunemente sono realizzati con architetture presentate in questo capitolo. Proprio per cui possiamo dire che certi oggetti che vengono usati comunemente, sono realizzati tramite architetture di questo tipo. Un robot funziona bene perché ha tanti sensori, sensori che è in grado di gestirli. Che poi al suo interno abbia qualche componente che ha atteggiamento/comportamento che possa essere ritenuto intelligente può anche esserlo, però in un oggetto (agente) di questo tipo è difficile vedere intelligenza o vedere dove vengono usati aspetti intelligenti.

Nell'esempio della macchina il problema non è tanto eseguire azioni ma è la percezione.

Chapter 9

Linguaggi e architetture per agenti BDI

In questo capitolo affronteremo linguaggi e architetture per agenti BDI. Linguaggi che verranno descritti attraverso le loro proprietà principali, legati, almeno in parte, ai formalismi presentati nei capitoli precedenti. Non tutti i linguaggi che consideremo sono utilizzabili in questo momento (ai tempi d'oggi). Alcuni stono stati proposti e progettati ma non sono stati molto utilizzati nel seguito. Noi ci focalizzeremo verso alcuni strumenti di programmazione che consentono di costruire e modellare agenti BDI.

9.1 Procedural Reasoning System (PRS)

Tanto per incominciare parliamo del PRS (Procedural Reasoning System) proposto da Georgeff e colleghi. È un architettura che può essere adottata da diversi tipi di linguaggi. Da questa architettura sono nati parecchi linguaggi (DMARS, Jam, Jack, ...) che sono stati utilizzati nella realtà con successo in molte applicazioni multiagenti con rilevanza particolare (come il controllo del traffico aereo). Noi daremo un'idea di quelle che sono le caratteristiche principali in modo da capire il suo funzionamento.

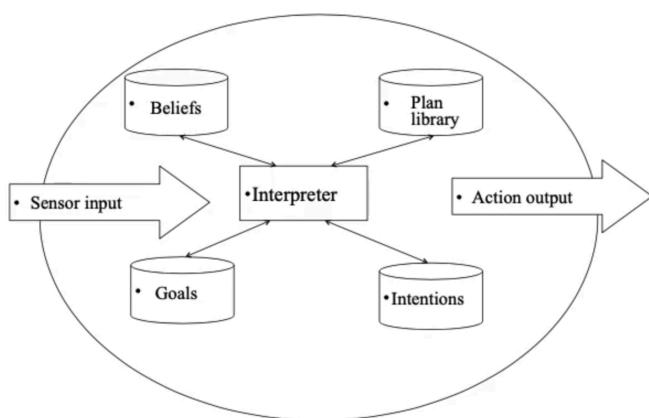


Figure 9.1: Architettura di un Procedural Reasoning System

Architettura 9.2 che un interprete che riceve in input dall'ambiente esterno, dove può eseguire delle azioni sull'ambiente esterno. Siccome stiamo parlando di architetture BDI, vediamo che all'interno ci sono dei contenitori che contengono *belief*, *goal*, *intention* e in più troviamo un Plan Library. Gli input del sistema sono **eventi**, quindi il modo in cui funziona questa architettura prevede che l'interprete lavori su una **coda di eventi** dove questi possono essere di due tipi:

- **esterni**, percepiti dall'ambiente

- **interni**, come aggiunta o cancellazione di belief o goal. Ad esempio possono modificare i belief e i goals.

Gli output sono azioni esterne o interne che sono eseguite dall'interprete. I belief sono rappresentati come i fatti nel Prolog: in pratica come atomi della logica del primo ordine.

9.1.1 I piani del PRS

I piani che sono contenuti nella libreria dei piani, sono pronti per essere usati quando si comincia ad eseguire un programma in un agente PRS. Gli agenti, non pianificano, i piani ci sono già e quindi forniti dal programmatore prima di iniziare l'esecuzione e vengono utilizzati nella computazione. La cosa principale che non esiste è un meccanismo per pianificare, quindi per costruire piani. L'esecuzione di un programma PRS utilizza/fa riferimento a piani *preesistenti*. Senza entrare troppo nel dettaglio, i piani hanno:

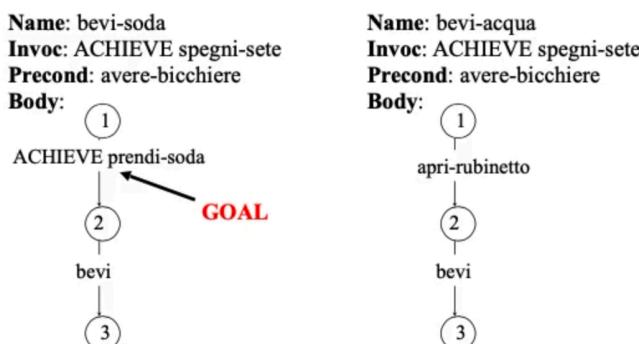
- un *nome*;
- una *condizione di evocazione*, conosciuta come *triggering event*;
- *precondizioni*, che devono valere prima che un piano venga eseguito;
- *body*, corpo. È una sequenza di espressioni (*simple plan expression*) che possono essere:
 - un'azione atomica o
 - sottogoal.

L'invocazione del piano dipende dal *triggering event*, cioè un qualche che fa scattare l'esecuzione del piano.

PRS: Esempio

Per spegnere la sete abbiamo due piani e questo si traduce nell'avere due possibili soluzioni. Il trigger del piano (*trigger-event*) (**Invoc**) è un goal di tipo ACHIEVE *spegni-sete* e questo

Due piani per spegnere la sete.



bevi e *apri-rubinetto* sono azioni.

Figure 9.2: Esempio di piano PRS

ACHIEVE è uguale per entrambi i piani. La pre-condizione è quella di avere un bicchiere, quindi per poter eseguire un piano quello che deve essere **vero** è avere un bicchiere.

Nei primi lavori di Georgeff, il piano era descritto come un grafo orientato e aciclico.¹. Nelle varie versioni di PRS di solito i piani erano in sequenza (in questo caso verticali) di azioni.

¹Vedere il body di un programma fatto come un grafo, risulta essere una soluzione pochettima complessa

1. per arrivare a prendere la soda, bisogna andare a prenderla da qualche altra parte. Questo piano inizia con un goal (ACHIEVE prendi-soda 9.3). Una volta presa viene eseguita l'azione bevi. Prima di bere comunque bisogna, come già detto, di procurarsi la soda e poi bere

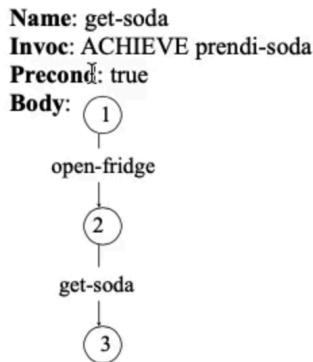


Figure 9.3: Un piano in PRS per prendere la soda

Se scegliessimo la soluzione bevi acqua finisce tutto lì, se invece scegliessimo il primo piano bisogna attivare prima il goal e quindi da qualche altra parte ci deve essere la descrizione di quel goal. In effetti, ci sarà un altro piano 9.3 che permetterà di ottenere la soda. L'esecuzione del piano è quella di far eseguire due azioni e se tutto va bene, allora la soda viene restituita. Quello che può succedere è che l'agente scelga di prendere la soda e quindi eseguire il primo goal (ACHIEVE prendi-soda), arriva al frigorifero e la soda non c'è e allora il piano si **pianta** senza andare avanti. Questo implica a far scattare questo piano dall'esecutore del piano che deve permettere di percorrere un'altra strada.

- Per raggiungere un dato *goal*, l'agente formula l'intenzione di raggiungere questo obiettivo, cioè sceglie un *piano* applicabile "triggered" dal *goal*. Questo piano diventa un'*intenzione*, ed è aggiunto alla *intention structure*.
- Ad ogni passo del loop principale, l'interprete sceglie un piano (parzialmente eseguito) nella intention structure, e ne esegue un passo. Questi piani che sono nell'intention structure che sono diventati intenzioni, vengono eseguiti un passo per volta. Ogni volta che l'interprete arriva sull'intention structure, prende un piano ed esegue un passo alla volta di quel piano.
- Se ci sono molto opzioni disponibili, l'interprete può scegliere quella con massima utilità, o entrare in un ragionamento di metalivello usando "metalevels plans". In generale però ci possono essere più intenzioni fra cui scegliere per cui l'interprete deve andare a vedere quali sono le possibili intenzioni e ne sceglie una mediante meccanismi di ragionamento che possono variare

9.1.2 Control loop del PRS

I passi principali del *control loop* del PRS sono i seguenti.

- all'inizio del ciclo vengono aggiornati *belief* e *goal* secondo gli eventi che ci sono nell'*event queue*;
- *goal* e *belief* possono cambiare;
- l'interprete va avanti, andando a cercare un piano applicabile e nel far questo va a vedere tutti i piani che può eseguire in modo da farci dei ragionamenti sopra (che vedremo più in là). Tra tutti i piani ne sceglie uno che va a finire nell'*intention structure*

- a questo punto nell'*intention structure* viene scelta un'intenzione e quindi un task da eseguire
- si esegue un passo di quel dove questo può essere
 - un'azione primitiva
 - scelta di un sottogoal che viene messo nella coda ed entra in ciclo insieme a tutti gli altri eventi che sono disponibili

La cosa principale che ogni volta che l'interprete arriva a selezionare qualche intenzione, ne esegue un Passo.

Esempio di esecuzione di 9.2

- supponiamo che l'evento "ACHIEVE spegni-sete" sia stato appena aggiunto all'*event queue*. Le *invocation conditions* dei piani *bevi-soda* e *bevi-acqua* fanno scattare questo evento, e quindi il generatore di opzioni restituisce ambedue i piani (assumiamo che *avere-bicchiere* sia valida);
- assumiamo che il *deliberator* scelga l'opzione *bevi-soda*, creano una nuova intenzione per questo. Il primo passo di questo piano è il goal "ACHIEVE prendi-soda". Quando questo passo è eseguit, il goal è "posted" nell'*event queue*.
- Nel ciclo successivo, l'*option generator* sceglie il piano per ottenere la soda e il suo frame è aggiunto in cima allo stack del piano precedente. L'agente ora esegue l'azione *open-fridge* e scopre che il frigo non contiene soda. Il piano fallisce e l'agente è costretto ad abbandonare la sua intenzione di bere soda e ripartire con il goal iniziale
- al prossimo ciclo, viene scelta l'opzione di bere acqua e il piano è completato con successo nei cicli successivi.

9.1.3 AgentSpeak(L)

Un linguaggio derivato dal PRS può avere dei problemi dal punto di vista formale e quello che Rao ha voluto fare è dare anche della semantica. Ha definito l'AgentSpeak(L) un linguaggio tipo il Prolog (quindi basato su clausole di Horn) che consente di dare una semantica operazionale. Dopodiché procedendo nell'evoluzione di questo approccio è nato il JASON che consente di realizzare sistemi multiagenti.

9.2 Agent-oriented programming (AOP)

Linguaggio proposto all'inizio degli anni 90 da Shoham. Con AOP si propone un nuovo paradigma di programmazione, che promuove una visione sociale della computazione, in cui più agenti interagiscono uno con l'altro. L'articolo presenta:

- una descrizione di quelli che sono gli stati mentali diversi da quella che normalmente veniva data nei linguaggi BDI.
- un linguaggio di programmazione AGENT-0 per definire agenti.

9.2.1 AOP: Categorie mentali

Ci sono due *categories mentali* principali:

- *belief*
- *obligation* (o *commitment*)

In aggiunta abbiamo la *decision*, trattata come una *obligation*, quindi un agente può prendere un commitment verso se stesso e infine un ulteriore categoria sono le *capability*, ovvero cosa sa fare quel particolare agente.

Le formule fanno riferimento esplicitamente al *tempo*. Cosa che non avviene in diversi altri approcci incontranti nei capitoli precedenti.

- le **formule** come detto, sono etichettate con un tempo dove quest'ultimo avviene con un'enumerazione (tempo1, tempo2,...,tempoK). È un linguaggio basato su istanti di tempo e questo ricorda un po la logic linear time.
- esempio di robot che tiene una tazza al tempo t. $\text{holding}(\text{robot}, \text{cup})^t$

Le azioni non sono distinte dai fatti: la presenza di un azione è rappresentata dal fatto corrispondente.

- **Belief** $B_a^t \varphi$. L'agente a crede al tempo t φ
- **Obligation** $\text{OBL}_{a,b}^t \varphi$. Significa che l'agente a all'istante t è obbligato, o impegnato, verso l'agente b circa φ , dove φ può rappresentare un'azione. Quindi a è commitment ad eseguire quell'azione rispetto a b
- **Decision**, $\text{DEC}_a^t \varphi =_{\text{def}} \text{OBL}_{a,a}^t \varphi$. La decisione è definita come commitment verso se stesso
- **Capability**, $\text{CAN}_a^t \varphi$. La capacità che all'istante t l'agente a sia capace di φ

9.2.2 AOP: AGENT-0

Motivo per cui l'approccio è da etichettare come BDI è perchè ci sono tutti gli stati mentali BDI che sono stati proposti anche per questo approccio. Nell'articolo citato, viene proposto un linguaggio di programmazione, linguaggio che contiene un certo numero di costrutti:

- **facts** (t atoms), (un atomo vale al tempo t)
- **private actions**, azioni private da eseguire al tempo t (DO t p-actions)
- **Communicative actions**, aspetto non particolarmente rilevante. Il linguaggio è previsto anche per realizzare sistemi multiagenti, quindi ci sono delle azioni di **comunicazione** che consentono agli agenti di comunicare tra loro.
 - (INFORM t a fact), al tempo t informa l'agente a che un *fact* vale
 - (REQUEST t a action), richiedi al tempo t all'agente a un'azione

Implementato come estensione del LISP. È un linguaggio basato sulle liste. Ogni agente ha quattro componenti

- un set di **capabilities**, cose che l'agente può fare
- un set di **belief** iniziali
- un set di **commitments**, cose che l'agente dovrà fare
- un set di **commitment rules**, questo è il programma che determina come agisce l'agente

9.2.3 Commitment Rule e Interprete: AGENT-0

Si faceva riferimento al linguaggio LISP, poichè tutte le espressioni sono rappresentate da liste e quindi racchiuse da () .

COMMITMENT RULES
Commitment rules hanno la seguente struttura:
 $(\text{COMMIT } \text{message-cond } \text{mental-cond } (\text{agent action})^*)$

- A **mental condition** è una combinazione logica di **mental patterns**. A mental pattern is one of: (B fact) or ((CMT a) action)
 (CMT means "committed").
- A **message condition** è una combinazione logica di **message patterns**: (From Type Content) dove *From* è il nome del mittente, *Type* è il tipo della azione comunicativa (INFORM, ...) e *Content* è un fatto o una azione che dipende dal tipo.
- (agent action)* è una lista di coppie

Figure 9.4

AOP: AGENT-0 commitment rules

```
(COMMIT
  (?agent REQUEST (DO ?time ?action))           messagecond
  (AND (B ?now (Friend ?agent))
        (CAN myself ?action)
        (NOT ((CMT ?anyone)
               (DO ?time ?any_action))))           mentalcond
  (myself (DO ?time ?action)))                  commitment
```

Se io ricevo un messaggio da *agent* che mi richiede di eseguire *action* al *time*,
 e io credo che

agent attualmente è un amico
 io so eseguire *action*
 al tempo *time* non sono impegnato a eseguire qualunque altra azione,
 allora mi impegno (commit) ad eseguire *action* at tempo *time*.

Figure 9.5

Quando eseguiamo questa commitment rules (9.5), quello che deve essere eseguito è un COMMITMENT di fare quello richiesto ad un certo TIME. Il calcolo della regola del commit, crea appunto questo commitment dell'agente a fare al tempo time una certa ACTION. Nella computazione rimangono i commitment, perchè ai commitment è associato il tempo a cui l'azione deve essere eseguita. Quindi i commitment calcolati da una commitment rules vengono messi da parte fino a quando non scatta il tempo previsto e solo quando si arriva al TIME allora scatta. La cosa interessante è che se un agente ha un commitment non è che lo esegue subito, ma aspetta che scatti il tempo in cui è stato previsto che questo commitment/richiesta venisse soddisfatta.

AOP: AGENT-0 interprete

Beliefs, commitments, e capabilities di un agente sono rappresentati ciascuno da un database. Beliefs e commitments possono cambiare ad ogni passo dell'esecuzione di un programma, mentre le capabilities sono fisse. L'interprete esegue il seguente ciclo:

- leggi i messaggi correnti, e aggiorna i belief e commitment (valutando le commitment rules);

- esegui i commitment per il tempo attuale, eventualmente risultando in un cambiamento di belief.

AOP

- I beliefs sono aggiornati o come risultato di essere informati, o come risultato di eseguire una azione privata.
- I commitments possono essere aggiunti come segue: Per ogni commitment rule:

(COMMIT messagecond mentalcond (agent_i; action_i)*)

- se la message condition vale per il nuovo messaggio in arrivo;
- se la mental condition vale per il nuovo stato mentale;

per tutti gli i , l'agente è capace di eseguire l'azione_i, allora per tutti gli i , prendere un commitment con agent_i per eseguire action_i al tempo specificato.

Chapter 10

Linguaggi logici per agenti

Introdurremo quelli che sono i linguaggi basati sulla logica per programmare agenti.

10.1 Linguaggi Logici

Secondo Wooldrige e Ciancarini i metodi formali giocano tre ruoli:

- **specificare** sistemi, ed è quello che abbiamo fatto descrivendo gli agenti BDI attraverso la logica. Specificare un agente non vuol dire avere un linguaggio di programmazione. Quello visto riguardava essenzialmente come modellare certi aspetti degli agenti usando una logica;
- **verificare** sistemi. Quindi altro uso che possiamo fare dei metodi formali è quello di verificare proprietà di sistemi. Fatto quando abbiamo parlato del **model checking**;
- **programmare direttamente** sistemi. Avere una descrizione formale di un sistema o meglio detto avere una definizione formale di programmazione basata su qualche formalismo (es. su qualche logica) e quindi formulare direttamente quelle che sono le proprietà dell'agente in una qualche logica. Approccio che richiede di usare un linguaggio logico che possa essere eseguito (computational logic). Esempio tipico è il Prolog. Linguaggio basato sulla logica classica, dove i programmi non sono nient'altro che insiemi basati su clausole di Horn. Un qualcosa di analogo può essere fatto riferendosi ad altre logiche. Abbiamo due formalismi che intorduremo in questo capitolo

10.1.1 GOLOG

Linguaggio di programmazione basato sul calcolo delle situazioni, ovvero basato su linguaggi formali, quindi su formalismi. Linguaggio abbastanza utilizzato. È stato progettato e discusso nel 1994 ed utilizzato tutt'ora in certi ambienti. Linguaggio usato per programmare robot di alto livello (*cognitive robots*) e agent software intelligenti. In GOLOG, le **azioni primitive** sono specificate dandone le *precondizioni* e *effetti* (rappresentati come *successor state axioms*). GOLOG formula i programmi basandosi su delle formule logiche basate su precondizioni ed effetti delle azioni.

- possibile definire delle precondizioni
- definire degli effetti dell'esecuzione delle azioni
- punto importante **success state axioms**. È una soluzione proposta dagli autori del GOLOG per risolvere/rendere più fattibile ed efficiente il problema dei frame che avevamo leggermente discusso nella lezione che riguardava proprio il GOLOG (non ricordo quale. Trovatelo con ctrl+f). È quindi possibile, per risparmiare sulla complessità del calcolo delle computazioni da fare, definire su ogni fluente una regola

Esempio:
Preconditions

$$\text{Poss}(\text{pickup}(x), s) \equiv \exists r \{ a = \text{drop}(r, x) \wedge \text{fragile}(x, s) \} \wedge \text{nexto}(x, s) \wedge \neg \text{heavy}(x)$$

Successor state axioms (uno per ogni fluente)

$$\begin{aligned} \text{Poss}(a, s) \Rightarrow & \{ \text{broken}(x, do(a, s)) \equiv \\ & \exists r \{ a = \text{drop}(r, x) \wedge \text{fragile}(x, s) \} \wedge \\ & \neg \text{broken}(x, s) \wedge \neg \exists r \{ a = \text{repair}(r, x) \} \} \end{aligned}$$

Figure 10.1

10.1.2 Azioni complesse

Il GOLOG permette di definire delle **azioni complesse** usando l'abbreviazione $\text{Do}(\delta, s, s')$, dove:

- δ è un'espressione di azione complessa;
- s, s' sono due stati

Intuitivamente $\text{Do}(\delta, s, s')$ è vera quando la situazione s' è una situazione terminante dopo l'esecuzione di δ iniziata nella situazione s . Ci sono comunque varie tipi di azioni che vengono fornite da questo linguaggio

- LISTE DI AZIONI CHE ANDREMO A SCRIVERE DOPO

La formalizzazione delle azioni complesse che abbiamo visto nelle slide precedenti si basa sulla logica dinamica, questa è quella logica delle azioni su cui si sono basati per le azioni del GOLOG.

GOLOG(alGol in LOGic) è stato progettato come un linguaggio di programmazione logica per domini logici, cercando di mescolare lo stile di programmazione dell'Algol con la logica. Prende a prestito dall'Algol molti costrutti standard della programmazione, come *sequenza, condizionale, procedure ricorsive e loop*. Per esempio

if ϕ **then** δ_1 **else** δ_2 $=_{\text{def}} [\phi?; \delta_1] \mid [\neg\phi?; \delta_2]$

Eredita quindi dall'Algol i costrutti standard come *if, then, else*. GOLOG usa un costrutto IF THEN ELSE che è costruito con azioni complesse. In questo caso facciamo un test su ϕ allora δ_1 altrimenti δ_2

10.1.3 GOLOG: un elevator controller

Esempio dell'ascensore per far capire un po cosa può maneggiare GOLOG

- **azioni primitive**, $up(n)$, $down(n)$, $open, \dots$
- **fluents**, $currentFloor(s)=n$, $nextFloor(n, s)$. Questi descrivono lo stato
- **Successor state axioms**
- **procedures**. In particolare è possibile definire procedure di calcolo. In questo caso le procedure dell'ascensore vengono introdotte con del WHILE. *Esiste un n tale che vale on(n) allora esegui la procedura serveAFloor*.
- la procedura *serveAFloor* ad esempio prende a caso un piano n da π
- procedura che serve il piano.

Un programma in GOLOG utilizza molto costrutti che sono presenti nei linguaggi di programmazione classici

10.1.4 Come eseguire un programma

Per poter eseguire un programma in GOLOG si procede in questo modo. Ci sono degli assiomi che vengono utilizzati e poi c'è il programma. La formula logica viene eseguita in questo modo. Dall'insieme degli assiomi si deriva quello che si ottiene dalla situazione S_0 e finisce in s

- $Axioms \models \exists s. Do(program, S_0, s)$
- S_0 è la situazione iniziale

Alla fine il risultato è una sequenza di stati e azioni che portano fino ad s , così facendo si ottiene un piano, ovvero una sequenza di azioni che portano dallo stato iniziale allo stato finale.

- $s = do(a_n, \dots, do(a_2, do(a_1, S_0))\dots)$

L'interprete GOLOG è un **theorem prover general-purpose** (in generale per la logica del secondo ordine). Così come succede per il Prolog, i programmi GOLOG sono eseguiti per i loro *side effects*, ossia per ottenere legami per le variabili quantificate esistenzialmente.

10.1.5 Estensioni del GOLOG

Dal 1994, anno in cui è stata presentata un'implementazione prototipale del GOLOG, sono state presentate altre versioni.

- GONGOLOG incorpora la concorrenza, trattando processi concorrenti con priorità diverse, interrupts, azioni esogene
- IndiGolog, i programmi possono essere eseguiti incrementalmente per permettere azioni interleaved, planning, sensing and exogenous events

10.2 Concurrent METATEM

Concurrent METATEM è un linguaggio basato sull'esecuzione diretta (avere delle formule in una qualche logica che possono essere eseguite direttamente dall'interprete o compilatore del linguaggio di programmazione) di *formule temporali*

- un sistema Concurrent METATEM contiene un insieme di agenti che possono comunicare fra di loro via *asynchronous broadcast message passing*. METATEM è un linguaggio concorrente, c'è un meccanismo di comunicazione tra agenti di tipo broadcast
- ogni agente è programmato dando una specifica del suo comportamento mediante logica temporale. Questa specifica può essere eseguita direttamente. Quindi un programma che essendo basato su logica temporale, si basa su un insieme di formule della logica temporale (che vedremo poco più avanti)
- la logica usata dal METATEM è la logica temporale linear time LTL estesa nel passato oltre che nel futuro. In questo modo viene possibile eseguire direttamente la specifica.

Non è un linguaggio molto diffuso, ma è interessante perché l'idea è quella di utilizzare la **logica temporale LTL**.

In quella discussa da noi è possibile partire da uno stato iniziale e andare fino all'infinito. Qui invece, nella logica utilizzata da METATEM (come detto) è possibile anche andare nel passato.

10.2.1 Operatori utilizzati in METATEM

- operatori che tengono conto del futuro

1. $\bigcirc \varphi$, φ deve essere soddisfatta nello stato successivo
2. $\varphi \mathbf{U} \psi$, φ sarà vera fino a ψ
3. $\diamond \varphi$, φ deve essere soddisfatta in qualche stato futuro
4. $\Box \varphi$, φ deve essere soddisfatta in tutti gli stati futuri

\diamond e \Box sono quelli che abbiamo chiamato nella logica LTL **F** e **G**, quindi rispettivamente prima o poi φ sarà soddisfatta nel futuro, l'altra sarà sempre soddisfatta nel futuro.

- operatore che tengono conto del passato

1. $\varphi \mathbf{S} \psi$, φ è stata vera fino a ψ
2. ¹ $\bullet \varphi$, φ è stata soddisfatta nello stato precedente.

- Per esempio...
 - $\Box \text{important(agents)}$
“ora e sempre sarà vero che gli agenti sono importanti”
 - $\Diamond \text{important(ConcurrentMetateM)}$
“in qualche momento nel futuro, ConcurrentMetateM sarà importante”
 - $\blacklozenge \text{important(Prolog)}$
“in qualche momento nel passato era vero che il Prolog era importante”
 - $(\neg \text{friends(us)} \cup \text{apologize(you)})$
“non siamo amici finché non chiedi scusa”
 - $\bigcirc \text{apologize(you)}$
“domani (nel prossimo stato), chiederai scusa”

METATEM

Un programma METATEM consiste di un insieme di regole con la forma:

$\Box(past \ and \ present \ formula \Rightarrow present \ or \ future \ formula)$

dove la parte destra è vincolata a essere una *disgiunzione* o una formula *sometime*.

Il linguaggio fornisce due meccanismi ortogonali per rappresentare la scelta:

indeterminatezza statica, con l'operatore classico \vee

indeterminatezza temporale, con l'operatore sometime \Diamond (però, dato $\Diamond \varphi$, l'esecuzione cerca di soddisfare φ appena possibile).

Figure 10.2

Esempi di regole

- **start** \Rightarrow popped(a), **start** vuol dire che stiamo nello stato iniziale sia verso il futuro che verso il passato (può essere usato per entrambi i modi). Una regola di questo tipo vuol dire che nello stato iniziale è soddisfatto popped(a)

¹Il cerchio dovrebbe essere solo con contorno **bold** e interno vuoto, vedere slide Martelli per capire

- $\bullet \text{pop}(X) \Rightarrow \diamond \text{popped}(X)$. Abbiamo detto che nella parte sinistra di una regola possiamo avere anche formule che facciano riferimento all'istante precedente, quindi, ogni volta che pop è soddisfatta all'istante precedente allora ci sarà questo commitment che prima o poi popped(X) sarà vera. Se invece di utilizzare la notazione $\diamond \text{popped}(X)$ possiamo usare la disgiunzione che dice che lo stack è vuoto oppure valga popped(X)

L'interprete prende tutte le regole e trova quelle che possono essere eseguite se hanno l'antecedente soddisfatto. Le regole possono essere davvero tante e da queste regole prende i conseguenti e li mette tutti assieme e li trasforma in forma disgiuntiva se già non lo fossero e tra questi ne sceglie una da eseguire. Nel caso in cui non riesce a fare niente di utile eseguendo una delle scelte, può fare sempre **backtracking** e sceglierne un'altra.

10.2.2 Scambio di messaggi tra agenti

Abbiamo detto che **METATEM** è un linguaggio concorrente e quindi ha una parte dedicata allo scambio di messaggi fra agenti. Questo viene modellato dicendo che vi sono tre categorie di prediciati

1. *environments* predicates. Sono prediciati che rappresentano messaggi che arrivano. Per esempio, per l'instance $\text{push}(X)$ questa è vera se un messaggio $\text{push}(b)$ è appena stato ricevuto
2. *Component* predicates, corrisponde ai messaggi che escono dall'agente. Quando $\text{popped}(c)$ è vera allora significa che il messaggio $\text{popped}(c)$ è stato inviato
3. *Internals* predicates.

Esempio di comunicazione

stack (pop,push) [popped, stack-full]

L'agente stack può ricevere messaggi che si chiamano o *pop*, o *push* e può inviare messaggi ad altri agenti che sono etichettati o come *popped* o come *stack-full*. Quindi, diciamo semplicemente che

- (), sono rappresentati i messaggi che può ricevere l'agente, ovvero una lista di messaggi che l'agente può ricevere;
- , una lista di messaggi che l'agente può inviare

Produttore-Consumatore

- il produttore *rp* può ricevere delle richieste o da *ask1,ask2* e i messaggi che *rp* può inviare sono *give1,give2*. La formula che definisce il comportamento dell'agente è riportata al punto successivo
- $\bullet \text{ask1} \Rightarrow \diamond \text{give1}$, se al passo precedente è arrivato l'*ask1* (quindi dal consumatore *rc1*), allora il produttore farà un commitment a dare prima o poi *give1*. Lo stesso per il messaggio *ask2*
- visto che **non è** previsto che entrambi i consumatori contemporaneamente occupino la risorsa che dovrebbe esserli fornita allora ci vuole un vincolo di questo tipo
start $\Rightarrow \square \neg (\text{give1} \wedge \text{give2})$

Lato consumatore

- riceve *give1* e manda via richiesta con *ask1*.

```

rp (ask1, ask2) [give1, give2]
Oask1  $\Rightarrow \Diamond$ give1  commits to eventually give to any agent that asks
Oask2  $\Rightarrow \Diamond$ give2
start  $\Rightarrow \Box \neg(give1 \wedge give2)$   can give to only one agent at a time

Two consumers:
rc1 (give1) [ask1]
start  $\Rightarrow$  ask1
Oask1  $\Rightarrow$  ask1  ask on every cycle
rc2 (ask1, give2) [ask2]
O( ask1  $\wedge \neg$ ask2)  $\Rightarrow$  ask2  an ask2 message is sent on every
                                cycle where, on its previous cycle, it
                                did not send ask2 but received ask1.

```

Linguaggi logici per agenti

20

Figure 10.3: Produttore Consumatore

- il programma di questo consumatore1 è quello che inizialmente fa un ask1 (in modo da dire che è lui) e la seconda regola è che allo stato precedente aveva fatto un ask1 e allora lo fa anche nello stato corrente.
- secondo consumatore. Può ricevere informazione che uno abbia fatto una richiesta oppure che sia arrivato un give2 per lui. La regola dice, se allo stato precedente, 1 ha chiesto e 2 non ha chiesto allora \Rightarrow chiede. Quindi ask2 mandato in ogni ciclo in cui quello precedente non aveva chiesto.

L'esempio non è particolarmente interessante ma il meccanismo SI! METATEMP è stato un tentativo di utilizzare la logica temporale anzichè la logica classica, ma in pratica non ha avuto un grandissimo successo perchè richiederebbe argomento molto ben definito.

Chapter 11

La teoria dei giochi

11.1 La teoria dei giochi

Che caratteristiche ha una **teoria dei giochi**? A differenza di quello che abbiamo visto fino ad adesso, siamo in un ambiente multiagente che ha come obiettivo quello di trovare in qualche modo un accordo fra tutti gli agenti che partecipano al gioco. Tali agenti sono **indipendenti** l'uno dall'altro, quindi ognuno ha i propri obiettivi

- insieme di agenti dove, basandoci su questa teoria, dobbiamo trovare un risultato che sia il migliore per tutti gli agenti partecipanti. Ogni agente partecipa per conto suo;

Uno dei problemi principali è quello di trovare un meccanismo con cui dare/ottenere decisioni razionali.

Piccola Nota

Il titolo *teoria dei giochi* è in qualche modo affine all'argomento di giochi che di solito viene presentato nei libri di AI. I giochi sono ad esempi quelli degli scacchi, dove i giocatori hanno un'informazione perfetta, (sanno tutti quello che succede sulla schacchiera) e hanno delle tecniche per trovare le migliori soluzioni.

Per quanto riguarda la teoria dei giochi (al contrario dei giochi normali dove abbiamo detto che i giocatori oltre che a giocare a turno sanno l'informazione perfetta di quello che sta succedendo),

- le mosse sono **simultanee**, dove i giocatori eseguono mosse tutte assieme. Ogni giocatore si muove per conto suo (concetto di agente **egoista**).
- i giochi che vedremo (sono molto semplificati rispetto a quello che viene studiato nella teoria dei giochi), consistono in una sola mossa e ciò li rende molto diversi dai giochi che conosciamo

La parola "gioco" è molto ingannevole perché fa pensare ai giochi classici, quando in realtà questa teoria si occupa di argomenti molto complessi. Più che giochi, si occupa di studiare situazioni economiche, tecniche e anche molto complesse. È un settore dove molti ricercatori hanno e lavorano per risolvere problemi molto complessi.

La teoria dei giochi può essere usata in vari modi:

- **progettazione di agenti**, ovvero teoria che ha delle proprie regole per cui la prima cosa che si può pensare di fare è quella di gestire non solo la decisione degli agenti ma anche quelle che sono le regole per partecipare al giochi. (*da slide*: la teoria dei giochi può analizzare le decisioni dell'agente e calcolare l'utilità attesa per ognuna di esse)

- **progettazione di meccanismi**, trovare protocolli che descrivono il modo con cui l'agente agisce, quindi quali sono le regole che possono essere usate per lo svolgimento di un gioco. (*da slide*: quando un ambiente è popolato da più agenti potrebbe essere possibile definirne le regole (ovvero il gioco a cui gli agenti devono partecipare) in modo da massimizzare il bene comune. Ad esempio la teoria dei giochi può aiutarci a progettare un protocollo per una collezione di router su Internet in modo che ciascuno di essi sia incentivato ad agire per ottimizzare l'uso globale della rete.)

11.1.1 Proprietà principali della teoria dei giochi

- le mosse sono simultanee;
- **giocatori** o agenti che prenderanno decisioni. Noi vedremo solo giochi a due giocatori;
- **azioni** (strategie) eseguite dai giocatori. Queste azioni (o strategie) non sono nient'altro che mosse adottate dai giocatori per eseguirle. Nei casi semplici un giocatore avrà un insieme di azioni disponibili per scegliere quale strada seguire. Rispetto ai giochi che conosciamo, i giochi di questo tipo, almeno negli esempi che vedremo, si basano sul fatto di eseguire una sola azione. Nel corso del gioco, un agente ha a disposizione delle strategie da adottare, ragionando su quella che sembra la migliore. Noi adottiamo che ogni giocatore sceglie una sola strategia, quindi strategie composte, nei casi più banali, composte da una sola azione.
- **strategie pure**, azioni che il giocatore deve intraprendere. Pure da non confonderle con **miste**;
- il risultato di un gioco è determinato dalle strategie che ogni partecipante ha deciso di adottare. Parliamo quindi di **strategy profile**, ovvero la specifica di quale strategia è stata adottata da ogni giocatore.

11.1.2 Un esempio: Il gioco Morra a due-dita

- abbiamo due giocatori P e D. Le azioni che vengono svolte da questi giocatori è quello di buttar "giu" (quindi mostrare simultaneamente) un dito oppure due.

Funzionamento del gioco:

- se f è il numero totale delle dita che vengono mostrate e questo risulta **pari**, P vince f dollari da D, altrimenti caso contrario

Ci sono quattro strategy profiles:

- $s_1(P:\text{uno}, D:\text{uno})$, $s_2(P:\text{uno}, D:\text{due})$, $s_3(P:\text{due}, D:\text{uno})$, $s_4(P:\text{due}, D:\text{due})$

Oltre a questo, questione fondamentale, è quella di descrivere quelle che sono le preferenze degli agenti. Preferenze, descritte mediante queste **utility functions**. Queste non fanno altro che associare numeri in base all'azione (strategia) che l'agente fa. Tale **numero**, indica quanto è "buono" il risultato per quell'agente

- Nel nostro caso abbiamo due utility functions u_P , u_D tali che, per esempio $u_P(s_1) = 2$ e $u_D(s_1) = -2$, perché il valore di f è 2, e quindi P guadagna 2 dollari mentre D li perde.
- analogamente per tutti gli altri strategy profiles

	D:uno	D:due
P: uno	P = 2, D = -2	P = -3, D = 3
P: due	P = -3, D = 3	P = 4, D = -4

Figure 11.1: Matrice di payoff - Morra a due-dita

Le utilità quindi vengono associate alle azioni eseguite (strategie utilizzate) Normalmente, anzichè scrivere tutte le strategy profiles, descriveremo le preferenze mediante una **matrice di payoff** che dà l'utilità a ogni giocatore per ogni combinazione di azioni di tutti i giocatori. Per esempio, la matrice di payoff per la Morra a due-dita è la seguente

Una soluzione al gioco è uno *strategy profile* in cui ogni giocatore adotta una **strategia razionale**, quindi strategia che porti ad una soluzione che sia migliore per l'agente che la esegue.

11.1.3 Un esempio: Dilemma del Prigioniero

Due presunti scassinatori, Alice e Bob, sono catturati con le mani nel sacco e interrogati separatamente dalla polizia. Entrambi sanno che, **se confessano il crimine**, verranno condannati a **5 anni** di prigione per furto con scasso, ma se **entrambi rifiutano di confessare** potranno essere condannati solo a **1 anno** per il crimine minore di possesso di oggetti rubati. Tuttavia, la polizia offre a ognuno di loro separatamente la possibilità di **testimoniare contro il complice**: in tal caso il testimone **se ne andrebbe libero**, mentre l'altro sarebbe condannato a **10 anni**.

Ora Alice e Bob sono posti di fronte al cosiddetto **dilemma dei prigionieri**: devono testimoniare o no? Essendo agenti razionali, Alice e Bob vogliono massimizzare la propria utilità. Supponiamo che Alice sia totalmente disinteressata del destino del complice: in questo caso la sua utilità decresce con il numero di anni che ella dovrà passare in prigione, ma quel che accade a Bob è del tutto ininfluente. Bob contraccambia tale sentimento.

Figure 11.2: Il dilemma del prigioniero

- sia Alice che Bob dovranno scegliere in modo **razionale** quale sia la soluzione migliore per loro. Le mosse si che sono simultanee, ma ognuno esegue la propria strategia;
- ognugno è **indipendente** dall'altro e nessuno dei due sa cosa sceglie l'altro

	Alice: testify	Alice: refuse
Bob: testify	A = -5, B = -5	A = -10, B = 0
Bob: refuse	A = 0, B = -10	A = -1, B = -1

Figure 11.3: Matrice di payoff - Il dilemma del prigioniero

- i numeri che si vedono corrispondono agli anni di prigione;
- l'utilità dipende dagli anni di prigione: più breve è il periodo di prigione, più alta è l'utilità

- qualunque cosa faccia Bob, per Alice risulta sempre meglio testimoniare (*testify*). Infatti si nota un -5 e uno 0 contro un -10 e -1

Alice analizza la matrice come segue:

- supponiamo che Bob testimoni. In tal caso io mi prendo 5 anni se testimonio e 10 se non lo faccio, quindi in questo caso è meglio **testimoniare**;
- d'altra parte, se Bob si rifiuta, io prendo 0 anni se testimonio e 1 anno se rifiuto. Quindi anche in questo caso **testimoniare** è meglio.

In definitiva, come detto, in entrambi i casi è meglio **testimoniare**, quindi io Alice testimonierò.

Alice ha scoperto che *testimoniare* è una **strategia dominante** per il gioco. Diciamo che una strategia s per il giocatore p **domina fortemente** la strategia s' se il risultato di s è meglio per p di quello di s' per ogni scelta di strategie da parte degli altri giocatori. Una **strategia dominante** è una che domina tutte le altre. È irrazionale scegliere una strategia fortemente dominata, ed è irrazionale non scegliere una strategia dominante quanto ne esiste una. Essendo razionale, Alice sceglie la strategia dominante.

Verifichiamo sulla matrice di payoff 11.3 che la **strategia dominante** per Alice sia *testify*

- supponiamo che Bob esegua *testify* (prima riga della tabella): per Alice è meglio *testify* che *refuse*, perché nel primo caso avrebbe un'utilità di -5, mentre nel secondo caso avrebbe un'utilità di -10;
- supponiamo adesso che Bob esagua *refuse* (seconda riga della tabella): per Alice è ancora meglio *testify* che *refuse*, perché nel primo caso avrebbe un'utilità di 0, mentre nel secondo caso avrebbe un'utilità di -1;
- si conclude quindi che per Alice è sempre meglio **testify**, qualunque cosa faccia Bob

11.1.4 La strategia dominante

Per semplicità consideriamo i giochi con due giocatori. Dato un giocatore A e due strategie s_{A1} e s_{A2} di questo giocatore diciamo:

- s_{A1} **domina strettamente** s_{A2} se, qualunque cosa faccia l'altro giocatore, s_{A1} dà al giocatore A un payoff maggiore di quello che s_{A2} può dare;
- s_{A1} **domina debolmente** s_{A2} se, qualunque cosa faccia l'altro giocatore, s_{A1} dà al giocatore A un payoff maggiore o uguale a quello che s_{A2} può dare e, in almeno un caso s_{A1} dà un payoff maggiore di quello che s_{A2} dà;
- s_{A1} è una **strategia fortemente dominante** se s_{A1} domina fortemente ogni altra strategia del giocatore A
- analogamente per **debolmente dominante**

Il dilemma

Se Alice è furba oltre che razionale, potrà continuare a ragionare come segue: anche per Bob la strategia dominante è testimoniare. Quindi, lui testimonierà ed entrambi saremo condannati a 5 anni. Il dilemma è che il risultato è peggiore, -5 per ciascun giocatore, del risultato che entrambi otterebbero se si rifiutassero di testimoniare, cioè -1 per entrambi.

- proprio per cui diremo che lo strategy profiles (*testify, testify*) non è **Pareto Optimal**, ma il profilo (*refuse, refuse*), lo è
- in generale un risultato è **Pareto Optimal** se non c'è un altro risultato preferibile da tutti i giocatori. Quindi in termini più terra terra è Pareto Optimal quando abbiamo uno strategy profiles meglio di un altro (quindi nel nostro caso numeri più piccolo). N.B. NON VUOL DIRE SCEGLIERE PER FORZA IL PARETO OPTIMAL

Pareto Optimal

Consideriamo un gioco a due giocatori. La definizione di **Pareto Optimal** si basa sulle preferenze: abbiamo visto che strategy profile è una coppia di strategie, una per ciascun giocatore, con un valore numero (utilità) associato a ciascun giocatore.

- dati due risultati w_1 e w_2 , un giocatore A preferisce w_1 a w_2 se l'utilità per A in w_1 sia maggiore di quella in w_2 ;
- w_1 è **strettamente Pareto** superiore a w_2 se ogni giocatore preferisce w_1 a w_2 ;
- w_1 è **debolmente Pareto** superiore a w_2 se ogni giocatore considera w_1 migliore o uguale a w_2 e almeno un giocatore preferisce w_1 a w_2
- un risultato è **Pareto Optimal** se non ce n'è nessuno superiore a lui

Se la soluzione $(-1, -1)$ è meglio di $(-5, -5)$ allora perchè non sceglierla come soluzione? Quello che potrebbe succedere è che uno dei due decide (potrebbe ingannare l'altro e prendersi una soluzione che sia meglio per lui. Ultima considerazione che possiamo fare è la seguente.

- la strategia che entrambi scelgano *refuse* è poco probabile perchè appunto avendo ciascuno dei due la possibilità di "fare lo scherzetto" e prendersi la cosa migliore non è una cosa razionale.
- Quindi la soluzione più studiata e più importante è che caratterizza un po tutto è che la situazione migliore si avrebbe con un punto di **equilibrio**, noto come

C'è un modo per Alice e Bob di arrivare al risultato $(-1, -1)$? Certamente è *possibile* che entrambi si rifiutino di testimoniare, ma è molto *improbabile*. Ciascuno avrebbe la tentazione di testimoniare, perchè ci guadagnerebbe 1 anno di prigione mentre l'altro si troverebbe con 10 anni, ma non c'è nessun motivo di ritenere che un giocatore testimoni e l'altro no.

- la ragione per cui la soluzione Pareto Optimal è improbabile è che non è un punto di equilibrio. Di solito si parla di quilibrio di Nash in onore del matematico Jhon Nash

11.1.5 L'equilibrio di Nash

¹ Uno strategy profile $S=(P1:s1, P2:s2)$ è un **equilibrio di Nash** se ogni giocatore P_i in S farebbe peggio se deviasse da S , assumendo che tutti gli altri giocatori si attengano a S . Precisamente:

- assumendo che il primo agente giochi s_1 , il secondo agente non può fare niente di meglio che giocare s_2 , and
- assumendo che il secondo agente giochi s_2 , il primo agente non può fare niente di meglio che giocare s_1
- **È ampiamente riconosciuto dagli studiosi della teoria dei giochi che lequilibrio è una condizione necessaria per qualunque soluzione a un gioco.**

¹Se ci sono strategie dominanti, queste possono descrivere un punto di equilibrio, non il viceversa

Per il dilemma dei prigionieri, solo lo strategy profiles (A:testify, B: testify) è un equilibrio di Nash. È difficile vedere come giocatori razionali possono evitare questo risultato, a meno che si cambi il gioco (o gli agenti) in qualche modo. Si può dimostrare che un equilibrio con strategie dominanti è un equilibrio di Nash, ma non tutti i giochi hanno strategie dominanti.

11.1.6 Un esempio: Costruttore hardware

Abbiamo detto che non tutti i giochi hanno strategie dominanti.

Vediamo un esempio.

Acme, un costruttore di hardware per videogiochi, deve decidere se il suo prossimo gioco userà DVD o CD. Intanto Best, produttore di software per i giochi, deve decidere se produrre il suo prossimo gioco su DVD o CD. I profitti saranno positivi se entrambi sono d'accordo, e negativi se non lo sono, come mostrato nella matrice seguente:

	Acme: DVD	Acme: CD
Best: DVD	A = 9, B = 9	A = -4, B = -1
Best: CD	A = -3, B = -1	A = 5, B = 5

Figure 11.4: Ulteriore esempio - Due equilibri di Nash

- in questo caso non c'è nessuna strategia dominante.
- se vogliamo vedere se A ha strategia dominante andiamo a vedere cosa può fare l'altra parte. Se B gioca DVD allora A guadagnerebbe 9 o perderebbe -4. Altrimenti, se B giocasse CD, allora A guadagnerebbe 0 5 o perderebbe -3. Si nota come c'è un cambiamento per la situazione di A. Non esiste, non c'è la possibilità che qualunque cosa faccia B questa determina una strategia dominante per A;

Però si notano due equilibri di Nash. (DVD, DVD) e (CD, CD). Questo corrisponde a scegliere la stessa strategia per i due giocatori. Punto di equilibrio perché se A gioca DVD, il B non può fare niente di meglio che giocare DVD e analogamente con CD, CD. Abbiamo quindi due punti di equilibrio. **Se ci sono equilibri di Nash i giocatori tendono ad accettare la soluzione che viene fuori dallequilibrio** Da come è stata strutturata la teoria dei giochi a noi interessa trovare una soluzione per ogni giocatore, ovvero l'indicazione di quale strategia serve ad ogni giocatore. In questo modo, se ci sono diverse soluzioni, allora diventa problematico stabilire quale delle due sia la soluzione giusta. Come procediamo?

Non possiamo dire che DVD, DVD sia la soluzione

- una possibilità visto che ci sono due punti di equilibrio diversi, potrebbe essere quella di scegliere la soluzione Pareto Optimal (DVD, DVD). Un modo di risolvere quindi + quello di prendere il punto di equilibrio che sia Pareto Optimal
- se dovessimo cambiare l'esempio, dove al posto (9, 9) → (5, 5), ci troviamo due punti di equilibrio con la stessa utilità sia per DVD,DVD che per CD,CD e nessuno dei due sarà meglio dell'altro. I meccanismi che abbiamo descritto non funzionano e per poter trattare un esempio di questo genere, gli agenti devono essere in grado di **comunicare** e questo vuol dire scambiarsi informazioni/negoziare con strumenti che non abbiamo descritto.

- se da una parte abbiamo due equilibri, ci potrebbero essere soluzioni che non presentono equilibri. Un esempio lampante è il gioco Morra a due-dita. Se prendiamo una strategy profile in cui ci sono questi due giocatori, dove un giocatore guadagna una cifra che è esattamente l'opposto di quello che perde l'altro giocatore (2, -2) allora diremo che il gioco in questione sarà a **somma zero**, ossia un gioco un cui la somma dei payoff in ogni cella della matrice è zero (non ci sono equilibri). Nei giochi a somma 0 a due giocatori, le vincite devono essere uguali ed opposte. Quindi nessun profilo di strategie può essere un equilibrio e dovrebbe considerare **strategie miste**.

11.1.7 Le strategie miste

	A: destra	A: sinistra
B: destra	A = -1, B = 1	A = 1, B = -1
B: sinistra	A = 1, B = -1	A = -1, B = 1

Figure 11.5: Gioco a somma zero - I calci di rigore

In questo gioco ci si aspetta che il giocatore A tiri un po a sinistra e un po a destra e lo stesso lo fa con uguale probabilità. Un giocatore non ha una sola strategia da scegliere ma ne ha k . In questo esempio è facile vedere che nessuna coppia di strategie pure forma un equilibrio. In pratica, come detto, ci si aspetta che A tiri un po a destra e un po a sinistra con uguale probabilità e lo stesso farà il giocatore (portiere) B. Il tutto può essere formalizzato introducendo la nozione di **strategie miste**.

- se un giocatore ha k scelte possibili $s_1, s_2, s_3, \dots, s_k$, una **strategia mista** su queste scelte ha la seguente forma
 - esegui s_1 con probabilità p_1 ;
 - esegui s_2 con probabilità p_2 ;
 - esegui s_3 con probabilità p_3 ;
 - ...
 - esegui s_k con probabilità p_k ;

con $p_1 + p_2 + p_3 + \dots + p_k$

Nash ha dimostrato il seguente risultato, per cui a Nash è stato conferito il premio Nobel nel 1994.

- **Ogni gioco in cui ogni giocatore ha un insieme finito di strategie possibili ha un equilibrio con strategie miste**

Nel nostro esempio, la soluzione è quella intuita ossia eseguire destra e sinistra con uguale probabilità di 0.5

Problema e conclusioni

Il teorema di Nash dice che, per ogni gioco, esiste un equilibrio con strategie miste, ma come si fa a trovarlo e con quale complessità? Esistono solo alcuni risultati per particolari classi di giochi. Il piccolo problema sottolineato da Wooldridge è come trovare queste strategie miste. Alcune persone hanno trovato algoritmi per trovare la soluzione con l'uso di strategie miste, ma per una particolare classe di giochi.

La teoria dei giochi non è solo un qualcosa cosa per fare giochi. In realtà, viene utilizzata, o si tene, per risolvere *problem complessi*. Un esempio sono i problemi di tipo economico, sociale.

- un problema molto citato è di come due nazioni possono trattare un "gioco" che comporta il fatto di eliminare le armi nucleari possedute;

Se qualcuno cerca di risolvere questo tipo di gioco con i soli strumenti a nostra disposizione non va molto lontano. Wooldrige nel suo libro introduce un po di questi problemi. Quello che uno vorrebbe è trovare un modo per specificare come si fa a **cooperare** (refuse) piuttosto che **disertare** (testify). Problema fondamentale delle interazioni multi-agenti.

Il dilemma del prigioniero. Piccole considerazioni

È un problema fondamentale per l'interazione tra agenti. IN tutti i casi vorremmo che vincesse la **cooperazione** ma in molti casi questo non accade, tanto che la soluzione nel dilemma del prigioniero è (testify, testify).

- da quello analizzato, risulta che la cooperazione che uno vorrebbe non si presenta molto bene in una società di egoisti;

Wooldrige cita diversi esempi.

- la riduzione delle armi nucleari ne è un esempio. Ogni paese potrebbe cooperare, quindi eliminare le armi, oppure optare per disertare e questo porta a conservare le armi.
- se un paese *i* elimina le sue armi, c'è il rischio che l'altro paese *j* si tenga le sue lasciano *i* in una cattiva situazione;
- viceversa, se *i* conserva le sue armi, il risultato sarà che il paese *j* o restituisce le armi oppure conservano entrambi le armi

L'ombra del futuro

Un qualcosa che è stato analizzato è la seguente:

- non è detto che un gioco si svolta una volta sola;
- se uno sa che incontrerà ancora l'avversario avrà un incentivo in qualche modo per cancellare qualcosa oppure applicare alla fine la cooperazione, ovvero uno ripete sempre lo stesso gioco fino a quando non si arriva alla fine
- supponiamo di avere un gioco molto lungo, quindi ripetuto *n* volte, con la certezza che la *n+1* non accada;
- Al giro *n-1*, tu hai un incentivo per disertare, guadagnando un extra bit di payoff...Ma questo rende il giro *n2* ultimo giro reale, e quindi tu hai un incentivo per disertare ancora. Questo è il problema di **backwards induction**.
- Giocare il dilemma del prigioniero con un numero di giri fisso, finito, pre-determinato, conosciuto da tutti, rende disertare la miglior strategia.

Il lavoro di Axelrod

Axelrod ha preso un insieme di giochi ed ha analizzato le strategie che si utilizzavano. Quello che ha fatto è prendere un certo numero di computer, ciascuno programmato per risolvere il dilemma del prigioniero, ma con algoritmi diversi l'uno dall'altro. Poi, ha preso tutti questi computer (dove ognuno giocava al meccanismo del prigioniero in modo diverso) e sono stati eseguiti circa 200 round. La strategia usate erano di questo tipo:

-

- **ALLD**:
– “Always defect” — the **hawk** strategy;
- **TIT-FOR-TAT**:
o On round $u = 0$, cooperate
o On round $u > 0$, do what your opponent did on round $u - 1$
- **TESTER**:
– On 1st round, defect. If the opponent retaliated, then play TIT-FOR-TAT. Otherwise intersperse cooperation and defection.
- **JOSS**:
– As TIT-FOR-TAT, except periodically defect

Figure 11.6: Strategie usate da Axelrod

Il vincitore del torneo, cioè il programma che ha fatto meglio complessivamente giocando contro tutti i programmi partecipanti, è risultato TIT-FOR-TAT, la strategia più semplice. Questo risultato sembra dimostrare che l'analisi iterata del dilemma del prigioniero è sbagliata: la cooperazione è la cosa giusta da fare. In realtà, TIT-FOR-TAT ha vinto perché il punteggio complessivo è stato calcolato considerando tutte le strategie contro cui ha giocato. Per esempio, ha perso contro ALL-D, come previsto. Alla fine del torneo Axelrod ha dato un suo elenco di regole/suggerimento.

- **Non essere invidioso**: Non giocare come se fossi a somma zero!
- **Sii simpatico**: Inizia cooperando e poi ricambia la cooperazione
- **Vendicati appropriatamente**: Punisci sempre il disertore, ma usa forza "misurata" non esagerare
- **Non tenere rancore**: Ricambia sempre immediatamente la cooperazione

Chapter 12

Progettazione di meccanismi

Cambiamo argomento, ma cerchiamo di rimanere comunque all'interno del contesto dell'*interazione tra agenti*. All'inizio del capitolo della teoria dei giochi, abbiamo accenato che la teoria in qualche modo è legata al concetto di **meccanismi**.

Con *mechanism design* intendiamo una sequenza di azioni che gli agenti eseguono per arrivare ad una certa conclusione. Nella teoria dei giochi abbiamo dei meccanismi che permettono di soddisfare l'obiettivo di un certo numero di agenti (nel nostro caso sono due e sono chiusi rispetto agli interessi degli altri, quindi sono **egoisti**) I protocolli specificano cosa gli agenti (quelli che interagiscono) possono fare e cosa non possono fare e quindi arrivare a trovare un risultato migliore. I protocolli in questione hanno certe proprietà desiderabili, come ad esempio:

- **successo garantito**, il protocollo garantisce che prima o poi verrà raggiunto un accordo;
- **massimizzare il welfare sociale**: un accordo massimizza la somma delle utilità dei partecipanti
- **pareto efficiency**: il risultato della negoziazione deve essere Pareto optimal;
- **individuale rationality**: i partecipanti hanno interesse nel seguire le regole del protocollo;
- **stability**, il protocollo dà un incentivo a comportarsi in un certo modo (es. equilibrio di Nash)
- **simplicity**, la strategia del protocollo è "ovvia".

Sono proprietà che in qualche modo sono legate alla teoria dei giochi, si tratta sempre di interagire in un insieme di agenti. Esempio di progetti di meccanismo è il punto a.

- a. Esempi di progetti di meccanismi includono la messa all'asta di biglietti aerei economici, l'instradamento di pacchetti TCP tra computer, la suddivisione di un gruppo di medici in più ospedali e la cooperazione all'interno di una squadra di calciatori robot.
- Formalmente un meccanismo consiste in un linguaggio per la descrizione delle possibili strategie adottabili dagli agenti e una regola per i risultati che determina le vincite degli agenti dato un profilo di strategie adottabili.
- A prima vista, il problema di progettare un meccanismo potrebbe sembrare banale. Se ogni agente massimizza la propria utilità, questo porterà automaticamente alla massimizzazione dell'utilità globale. Sfortunatamente questa soluzione non funziona, perché le azioni di ogni agente possono influenzare il benessere degli altri in modo tale da ridurre l'utilità globale.

- Un esempio è la **tragedy of commons**, una situazioni in cui tutti i contadini portano le loro bestie a brucare gratis sui prati comunali, con il risultato di distruggerli e giungere così a un'utilita negativa per tutti. Ogni contadino ha agito singolarmente in modo razionale, ragionando che l'uso del prato comune era libero
- Argomentazioni simili si applicano all'uso dell'atmosfera e degli oceani come discariche gratuite di sostanze inquinanti.
- Un approccio standard per gestire simili problemi è "far pagare" a ogni agente l'uso delle risorse collettive.

12.1 Le aste

Fra tutti questi meccanismi possiamo elencare le aste. Le citiamo senza entrare troppo nei dettagli. Ce nè sono di tanti tipi. Normalmente vengono elencate quelle che sono certe proprietà che riguarda appunto le aste

- c'è un bene messo in vendita, che però può avere un valore **pubblico** (valore che vale per tutti i partecipanti all'asta) o **privato**(dove ogni partecipante ha la sua idea di quanto valga l'oggetto)

Esistono altri tipi di asta. La più nota è quella *inglese*.

- acquirenti interessati partecipavano all'asta finchè non si arriva a chi offre di più;
- c'è un banditore che incrementa via via il prezzo dell'articolo, controllando che ci siano ancora acquirenti.
- parlando di meccanismi viene fuori anche un discorso di strategia, in particolare i partecipanti hanno una *strategia dominante*, continuare a puntare finchè il costo corrente è inferiore al valore personale.

Un tipo di asta meno nota è quella **olandese**, in cui il banditore parte offrendo il bene a un prezzo molto alto e continua abbassando il prezzo finchè un qualche partecipante accetta l'offerta del banditore.

Nella teoria delle aste rientrano anche le aste a **busta chiusa**. Una variante di questo meccanismo è l'asta di **Vickrey**, detta anche **asta in busta chiusa al secondo prezzo**, in cui come nel caso precedente vince chi fa l'offerta più alta, ma paga un prezzo corrispondente alla seconda offerta più alta e non alla sua propria. In questo caso si può verificare che esiste una *strategia dominante* molto semplice: offrire il proprio *valore privato*, non serve offrire di meno o di più di quello che si ritiene essere il valore dell'oggetto in vendita. L'agente offre quello che secondo lui è il valore dell'oggetto. Supponiamo che l'offerta più alta sia stata fatta dall'agente i , che ha offerto 9Euro, mentre l'agente j risulta secondo con un offerta pari a 8Euro, allora l'agente i vince l'asta pagando il bene solo 8Euro.

In generale, se il tuo valore privato è v_i e hai offerto b_i , è semplice verificare che non si ottiene un risultato migliore se b_i è diverso da v_i :

- $b_i > v_i$ puoi risultare primo, ma rischi di pagare una cifra maggiore di v_i ;
- $b_i < v_i$ hai meno probabilità di vincere rispetto a offrire v_i . Ma, anche se vinci, la cifra da pagare sarà sempre quella che pagheresti offrendo v_i , ossia quella proposta dal secondo vincitore.

12.2 L'instradamento dei pacchetti

Consideriamo adesso il problema dell'instradamento di pacchetti su Internet, che è stato molto studiato in ambito informatico. I giocatori corrispondono agli archi nel grafo delle connessioni di rete. Ogni giocatore fornisce il costo di inviare un messaggio sul proprio arco. L'obiettivo è trovare il cammino più economico per trasmettere un messaggio dalla sua origine alla destinazione. La soluzione è facile da trovare mediante un normale algoritmo per cammini minimi. Purtroppo però c'è il rischio che un giocatore fornisca un costo maggiore di quello reale. Per evitare questo è stato proposto il seguente meccanismo: pagare ad ogni partecipante i un payoff pari alla lunghezza del cammino più breve che non contiene l' i -esimo arco meno la lunghezza del cammino più breve in cui il costo dell' i -esimo arco è uguale a 0. È possibile dimostrare che, con questo meccanismo, la strategia dominante di ogni giocatore diventa quella di riportare il valore corretto del costo del proprio arco.