

# Domande

1. [Situation calculus](#)
2. [Tesi di Brooks](#)
3. [Logiche temporali](#)
4. [Architettura METATEM](#)
5. [Logica modale e corrispondenza, assiomi](#)
6. [Model Checking](#)
7. [Automi di Buchi](#)
8. [Teoria dei giochi, torneo di Axelrod](#)
9. [problema dell'onniscienza logica nella logica epistemica](#)
10. [cos'è agent oriented programming](#)
11. [equilibrio nash](#)
12. [dilemma del prigioniero, strategia dominante](#)
13. [linguaggi bdi dei 4 tipi: coen levesque e gli altri due](#)

## DOMANDE BALDONI

1. Tipi di commitment (blind, single, open)
2. Speech Act Theory: Locuzione, Illocuzione, Perlocuzione
3. FIPA ACL -> Feasibility precondition, Rational effects, Conformità alla semantica
4. ContractNet Protocol, stati mentali e ruoli
5. Domande generiche su JADE
6. che cos'è un piano rilevante
7. in Jade lo scheduling dei behaviour è preemptive o non preemptive. Qual'è la differenza tra preemptive e non preemptive
8. in JADE in fase di scheduling quando e come viene eseguita la action.
9. dire quali sono le funzioni di selezione della semantica operazionale di AgentSpeak L e descrivere di cosa si occupano
10. Problema della trasduzione e della rappresentazione negli agenti simbolici
  1. differenza fra [AgentSpeak L](#) e la sua implementazione jason
  2. Speech Act Theory: Locuzione, Illocuzione, Perlocuzione
  3. Eventi interni vs esterni di jason
  4. Azioni interne jason
  5. Achievement goal vs test goal. il secondo non si puo usare nel contesto di un piano
  6. 3 tipi di commitment
  7. asta a busta chiusa
  8. control loop, dinamicità del mondo
  9. contract net protocol

# Lezione 1-2

Introduzione ad agenti e sistemi multiagente.

Perché gli agenti intelligenti?

5 principali trend che hanno caratterizzato questa storia:

- **ubiquità:**

Dispositivi economici e dotati di un'ottima capacità di calcolo → Concetto di computazione in luoghi che prima non erano attesi

- **Interconnessione**

I sistemi di elaborazione non esistono più da soli. Tramite il network possiamo interagire con agenti intelligenti -> Costruire paradigmi che trattano l'informatica come un processo di interazione

- **Intelligenza**

I compiti che deleghiamo sono sempre più complessi

- **Delega**

Vorremmo delegare dei compiti senza un nostro intervento. Spesso ci fidiamo più del giudizio della macchina che della persona (guida autonoma più affidabile del pilota)

- **Human orientation**

Spostare la programmazione verso l'uomo e non verso la macchina a livello di comprensione. usare metafore più vicine a come noi vediamo il mondo -> Astrazione, l'informatica è astrazione

Delega + intelligenza = sistemi informatici che agiscono in modo indipendente ed efficiente  
Agire in modo che rappresentino i nostri interessi -> diminuire la distanza da quello che è il sistema e come lo risolvo

Interconnessione e distribuzione = sistemi che cooperano/competono

Tutti questi problemi hanno portato ai **sistemi multi agenti**

**Agente** è un sistema di computazione, un sistema informatico, capace di agire in modo indipendente per conto di un utente o proprietario (capendo cosa deve essere fatto per soddisfare gli obiettivi di progettazione, piuttosto che essere costantemente informato)

**Sistema Multi Agente** consiste in una serie di agenti, che interagiscono l'uno con l'altro. Nel caso più generale, gli agenti agiscono per conto di utenti con differenti obiettivi e motivazioni. Per interagire con successo, richiedono la capacità di cooperare, coordinarsi e negoziare tra loro, in modo analogo a quanto fanno le persone.

**Microvisione**

Un Sistema multi agente risponde a tutte le domande di cooperazione coordinamento e cooperazione tra agenti self-interested.

### Macro visione

Gli agenti formano delle società, come possiamo creare agenti capaci di agire in modo indipendente, ed interagire con altri agenti per portare a termine i propri obiettivi. E' progettare società oltre che sistemi ad agenti.

I Mas sono un argomento multidisciplinare (come ogni argomento di AI).

Questo può essere un punto di forza ma anche una debolezza (molti punti di vista diversi che devono essere conciliati)

Non sono solo una branca di AI?

- spesso in realtà per costruire sistemi utili ci bastano cose semplici
- inoltre l'AI ignora gli aspetti sociali dell'agire, l'attività sociale è intelligenza

Siamo anche però interessati ad agenti computazionali (a differenza della teoria dei giochi)

### Cosa si intende per agenti Intelligenti

La principale caratteristica degli agenti è la loro **autonomia** (agire indipendentemente e controllo sul proprio **stato interno**)

Un agente è un sistema di computazione capace di agire **autonomamente** in un qualche **ambiente** con il fine di raggiungere gli **obiettivi** per cui è progettato.



Nel paradigma ad agenti, **agenti (processo) e ambiente (dati) sono sullo stesso livello**, devo dare importanza ad entrambi per ottenere lo stato desiderato

----- Parentesi -----

### Test di Turing

Capire se dall'altra parte del muro c'è una persona, è questa l'intelligenza? basta produrre gli output attesi per capire se vi è comprensione?

La stanza cinese è un esperimento che fa capire che anche se risponde non capisce, stessa cosa Ok Google

- AI forte: possibile riprodurre l'intelligenza umana? Compresa la consapevolezza di sé, l'essere senziente, . . .

- AI debole: esistono modi automatici per risolvere problemi che ad un essere umano richiedono intelligenza? Task-oriented, studio del pensiero e del comportamento umano

Gli informatici fanno AI debole, cerchiamo metodi automatici per rispondere

Cosa può fare automaticamente una macchina? Una macchina non è altro che un calcolatore di funzione  $f: N \rightarrow N$  (tra naturali, neanche numeri negativi) e lo calcolano in un ciclo di macchina. E anche la funzione è un numero naturale, le funzioni tra  $N$  ad  $N$  sono in ordine di grandezza dei reali, noi però rappresentandole solo con un numero naturale restringiamo di un infinito il numero di funzioni utilizzabili

AI non espande le funzioni possibili ma ci semplifica la costruzione di questa funzione

---

### Agente Intelligente

E' un sistema di computazione capace di esibire azioni **autonome** in modo **flessibile** in un **ambiente**.

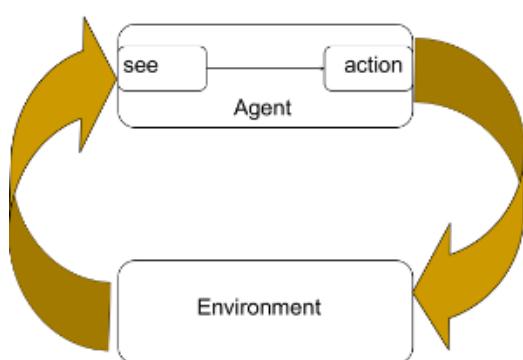
Per flessibile intendiamo:

### Reattivo

se l'ambiente è statico/fisso non ci preoccupiamo di questo (es il compilatore) nel mondo reale però è dinamico e le informazioni sono incomplete -> possibilità che le esecuzioni falliscono.

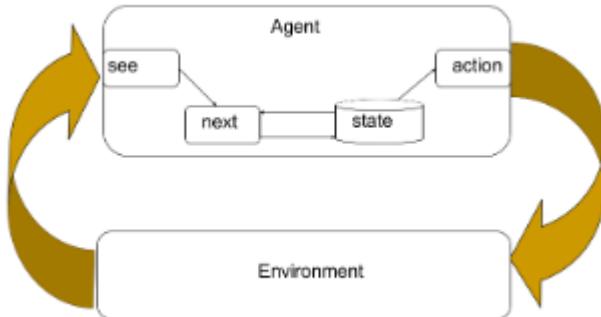
Definizione:

è un sistema che mantiene una costante interazione con l'ambiente e risponde ai cambiamenti che occorrono su di esso (in tempo perché la risposta sia utile)



### Proattivo

In generale desideriamo delegare dei compiti, comportamenti **guidati dagli obiettivi**  
Mantengono uno **stato**, info su come l'ambiente evolve e le proprie azioni influenzano l'ambiente, fa previsione per scegliere l'azione da effettuare



E' difficile trovare un trade-off tra Reattività e proattività

## Sociale

Il mondo è un ambiente multi-agente, non possiamo raggiungere i nostri obiettivi senza l'aiuto degli altri

Definizione:

Per abilità sociale (social ability) di un agente intelligente si intende l'abilità di interagire con altri agenti (anche umani) attraverso un qualche tipo di linguaggio di comunicazione (agent-communication language,ACL) e cooperare con essi.

Ma, stiamo parlando di agente o di un programma?

Parliamo di agente e non di un semplice programma quando:

Un agente autonomo è un sistema situato in un ambiente che può percepire ed agire su di esso nel tempo con il fine di perseguire una propria agenda e così facendo influire nelle successive percezioni

## Architetture per Agenti

Utilizzo di ragionamento logico per decidere cosa fare (fino al 1985)

dal 1985-oggi -> Agenti reattivi

dal 1990-oggi -> Architetture ibride (es Jason)

## Ragionamento simbolico

Vedere agenti come casi particolari di sistemi basati sulla conoscenza

- Contiene una esplicita rappresentazione (modello simbolico) dell'ambiente
- Prende decisioni (ad esempio, quale azione eseguire) attraverso un ragionamento simbolico

Abbiamo il problema della **trasduzione** (traduzione dell'ambiente in una descrizione simbolica utilizzabile in tempo)

E il problema della **rappresentazione/ragionamento** (come rappresentare simbolicamente le informazioni e ragionarci in tempo)

Questi problemi sono però complessi, complessità degli algoritmi di manipolazione simbolica e sono altamente intrattabili

## Lezione 3-4

### Agenti razionali

Non aumenteremo la potenza di calcolo di una macchina ma faremo astrazione per dominare la complessità nella costruzione del nostra soluzione al problema. Alcuni concetti mentali associati all'uomo sono uno strumento di rappresentazione della computazione di un agente (si parla di agenti BDI credo)

Parliamo di agenti come **sistemi intenzionali** -> Gionni prende il suo ombrello perché crede che pioverà; ho studiato molto perché voglio un phd. Queste affermazioni fanno uso della psicologia popolare, spieghiamo un comportamento con una credenza/desiderio popolare.

Le attitudini di queste descrizioni che fanno uso della psicologia popolare sono chiamate **azioni intenzionali**.

**Sistema intenzionale**, il comportamento è previsto da credenze desideri e di acume razionale

Il problema è se è possibile attribuire queste credenze popolari a sistemi informatici, quand'è che è naturale attribuire ad un sistema informatico credenze intenzioni desideri? Con sistemi complessi una spiegazione meccanicistica potrebbe non essere praticabile (a differenza di un interruttore) in questi casi una visione del sistema informatico in uno intenzionale può facilitare il processo di spiegazione (spieghiamo il nesso tra uno stato e un'attività che avviene successivamente)

più i sistemi di calcolo diventano complessi, più hanno bisogno di astrazioni e metafore potenti per spiegare il loro funzionamento – le spiegazioni di basso livello diventano impraticabili. Le **posizioni intenzionali sono una tale astrazione**  
es nascondiamo la complessità della macchina in strati per spiegare il SO

Le nozioni intenzionali sono quindi astrazioni, che forniscono un modo semplice e familiare per descrivere, spiegare e prevedere il comportamento di sistemi complessi

I sistemi intenzionali sono un sistema **potente di astrazione** (non di calcolo)

Idea, programmare sistemi informatici con **posizioni intenzionali**, più facili da comprendere

**La Posizione intenzionale** è uno strumento di astrazione: un modo conveniente di parlare di sistemi complessi, che ci permette di prevedere e spiegare il loro comportamento senza dover capire come funziona effettivamente

Il **practical reasoning** si valutano le varie opzioni per decidere cosa compiere, è il ragionamento sulle azioni e il processo per capire cosa fare:

- deliberazione quali stati di cose vogliamo raggiungere -> output: **intenzioni**
- pianificazione come vogliamo raggiungere questo stato di cose, questo processo è chiamato **means-ends reasoning**, il fine è raggiungere un **piano** che l'agente tenterà di eseguire

definizione di **intenzioni**:

Ci riferiamo allo stato di cose che un agente ha scelto e al quale si impegna come sue intenzioni

Il ruolo delle intenzioni è favorire la **proattività**, inoltre le intenzioni giocano un ruolo molto più forte nell'influenzare l'azione rispetto ad altri atteggiamenti proattivi come il desiderio

Le intenzioni hanno una proprietà di **persistenza** e **vincolano il futuro practical reasoning** inoltre:

- pongono problemi agli agenti, che devono determinare modi per realizzarli.
- forniscono un "filtro" per l'adozione di altre intenzioni, che non devono entrare in conflitto.
- Gli agenti monitorano il successo delle loro intenzioni e sono inclini a riprovare se i loro tentativi falliscono.
- Gli agenti credono che le loro intenzioni siano possibili.
- Gli agenti non credono che non porteranno avanti le loro intenzioni.
- In determinate circostanze, gli agenti credono che realizzeranno le loro intenzioni.
- Gli agenti non devono necessariamente avere intenzione su tutti gli effetti collaterali delle loro intenzioni (se ho intenzione di andare dal dentista non ho intenzione di avere dolore)

## Come realizzare un Agente che sfrutta il practical reasoning

Agent control loop

**Agent Control Loop Version 1:**

```
while true do
    observe the world;
    update internal world model;
    deliberate about what
        intention to achieve next;
    use means-ends reasoning to
        get a plan for the intention;
    execute the plan
```

Considerazioni: la **deliberazione** e il **means-ends reasoning** NON sono processi istantanei (hanno un costo in termini di tempo)

Inoltre un'azione ottimale era così quando inizio a deliberare ma non ho certezza che lo sia quando ho finito.. corro il rischio che l'intenzione ottimale non lo è più senza contare che devo ancora pensare a come farlo (mean-ends reasoning)

Funziona se il **mondo è garantito statico** oppure il tempo è minimo

Formalizziamo meglio il ciclo introducendo il concetto di **beliefs**

### Agent Control Loop Version 2:

```
/* initial beliefs */  $B := B_0;$ 
```

```
while true do
```

```
    get next percept  $\rho$ ;
```

```
     $B := brf(B, \rho);$ 
```

```
     $I := deliberate(B);$ 
```

```
     $\pi := plan(B, I);$ 
```

```
    execute( $\pi$ )
```

Estendiamo meglio il concetto di deliberazione esprimendo i **desire** e le **intention**

### Agent Control Loop Version 3:

```
 $B := B_0;$ 
```

```
 $I := I_0;$ 
```

```
while true do
```

```
    get next percept  $\rho$ ;
```

```
     $B := brf(B, \rho);$ 
```

```
     $D := options(B, I);$ 
```

```
     $I := filter(B, D, I);$ 
```

```
     $\pi := plan(B, I);$ 
```

```
    execute( $\pi$ )
```

(filtriamo i desire fattibili e li definiamo intention)

Introduciamo i **commitment strategies**

- **Blind commitment** (fanatico).

Un agente blind committed continuerà a mantenere un'intenzione fino a quando non crederà che l'intenzione sia stata effettivamente raggiunta. Il Blind commitment è talvolta indicato anche come fanatical commitment

- **Open-minded commitment.**

Un agente open-minded committed manterrà un'intenzione fintanto che la ritiene, la crede, possibile

- **Single-minded commitment** (risoluto).

Un agente single-minded committed continuerà a mantenere un'intenzione finché non ritiene che l'intenzione sia stata raggiunta, o che non sia più possibile raggiungerla

La differenza tra le ultime due sta proprio nella credenza nella secondo mentre l'effettiva irraggiungibilità della terza

Nel nostro ciclo le intenzioni verranno riconsiderate solo al ciclo dopo, quindi dopo che il piano è stato completato (o completa il piano o le intenzioni non verranno riconsiderate)

Anche il piano non viene considerato nei suoi effetti durante l'esecuzione.

Quindi si impegna sia sul fine che sui mezzi in maniera eccessiva

Soluzione: **ri-pianificare se qualcosa va storto**

### Agent Control Loop Version 4:

```
B := B0;
I := I0;
while true do
    get next percept  $\rho$ ;
    B := brf(B,  $\rho$ );
    D := options(B, I);
    I := filter(B, D, I);
     $\pi$  := plan(B, I);
    while not empty( $\pi$ ) do
         $\alpha$  := hd( $\pi$ );
        execute( $\alpha$ );
         $\pi$  := tail( $\pi$ );
        get next percept  $\rho$ ;
        B := brf(B,  $\rho$ );
        if not sound( $\pi$ , I, B) then
             $\pi$  := plan(B, I)
```

Abbiamo però ancora un over commitment rispetto alle intenzioni, dopo aver eseguito un azione sono le stesse di prima? quello che è cambiato nel mondo potrebbe farmi capire che non va più bene quella intenzione?

es scopro che chiedendo il prezzo è troppo caro, o trovo un altro modo di ottenerlo o abbandonare l'idea di avere quel prodotto

Idea: fermarsi per determinare se le intenzioni hanno avuto successo o se sono diventate impossibili da soddisfare (single-mind commitment)

## Agent Control Loop Version 5:

```
B := B0;
I := I0;
while true do
    get next percept  $\rho$ ;
    B := brf(B,  $\rho$ );
    D := options(B, I);
    I := filter(B, D, I);
     $\pi$  := plan(B, I);
    while not empty( $\pi$ ) or succeeded(I, B) or
        impossible(I, B) do
         $\alpha$  := hd( $\pi$ );
        execute( $\alpha$ );
         $\pi$  := tail( $\pi$ );
        get next percept  $\rho$ ;
        B := brf(B,  $\rho$ );
        if not sound( $\pi$ , I, B) then
             $\pi$  := plan(B, I)
```

Mi rendo conto che le intenzioni sono impossibili da raggiungere così, però potrei eventualmente rivederle, fare qualcosa di diverso e le intenzioni così vengono riviste solo quando ha:

- completato il piano
- ritiene di aver raggiunto le attuali intenzioni
- ritiene che non siano più possibili

Possiamo però pensare di rivedere le intenzioni dopo aver fatto la percezione:

## Agent Control Loop Version 6:

```
B := B0;
I := I0;
while true do
    get next percept  $\rho$ ;
    B := brf(B,  $\rho$ );
    D := options(B, I);
    I := filter(B, D, I);
     $\pi$  := plan(B, I);
    while not empty( $\pi$ ) or succeeded(I, B) or
        impossible(I, B) do
         $\alpha$  := hd( $\pi$ );
        execute( $\alpha$ );
         $\pi$  := tail( $\pi$ );
        get next percept  $\rho$ ;
        B := brf(B,  $\rho$ );
        D := options(B, I);
        I := filter(B, D, I);
        if not sound( $\pi$ , I, B) then
             $\pi$  := plan(B, I)
```

Rivedo le intenzioni dopo aver eseguito un'azione del piano oltre che a verificare la correttezza del piano sulle nuove beliefs e intentions

**Ma la riconsiderazione delle intenzioni costa!**

Soluzione: in base alle nuove beliefs e alle vecchie intenzioni mi chiedo se valga la pena riconsiderare le mie intenzioni

### Agent Control Loop Version 7:

```
B := B0;
I := I0;
while true do
    get next percept ρ;
    B := brf(B, ρ);
    D := options(B, I);
    I := filter(B, D, I);
    π := plan(B, I);
    while not empty(π) or succeeded(I, B) or
        impossible(I, B) do
        α := hd(π);
        execute(α);
        π := tail(π);
        get next percept ρ;
        B := brf(B, ρ);
        if reconsider(I, B) then
            D := options(B, I);
            I := filter(B, D, I)
        if not sound(π, I, B) then
            π := plan(B, I)
```

Questo **reconsider** è una funzione di meta-livello

Il reconsider deve essere flessibile da valutare e quindi il costo di questa fz è minore di deliberare

Scegliere se essere un agente **cauto** oppure un agente **audace** (quindi riconsiderare o meno le intentions) dipende dal tasso di cambiamento del mondo, ovvero **il dinamismo del mondo**.

# Lezione 5

## Logica per gli agenti

La logica classica non è particolarmente adeguata per modellare gli agenti o per ragionarci su agenti -> Soluzioni per adattare agli agenti la logica classica.

La logica fornisce strumenti formali per **rappresentare e ragionare** sulla conoscenza

E' ovvio quindi pensare che un agente possa usare una rappresentazione basata sulla logica per ragionare e formalizzare le conoscenze che ha sul mondo esterno, si può quindi pensare che sia uno strumento da utilizzare.

Ma la cosa più interessante è che può essere utilizzata per **ragionare sul comportamento di un agente**, e verificare che si comporti come desideriamo noi (in base alle specifiche fornite)

Partiamo dalla **logica per gli agenti** diversa da quella classica(proposizionale o FOL), la **logica modale**

Perché la logica classica non va bene per ragionare sugli agenti?

Gli agenti sono intenzionali basati su stati mentali, supponiamo che un agente voglia formulare con la logica: *John crede che Superman voli*

Io formuliamo così: *Bel(John, vola(Superman))* dove *Bel* è un predicato

C'è un problema che riguarda la **sintassi** della logica classica

Predicato(term, ... , term)

vola(Superman) però non è un termine della logica classica ma una formula..

Quindi quello che abbiamo scritto prima non è una formula della logica classica!

Ma anche **semanticamente**

Gli operatori intenzionali come *Bel* costituiscono un contesto chiuso (sono **referentially opaque**) non è detto che quello che è noto nei beliefs di un certo agente sia conosciuto generalmente! (es se so che Superman = Clark posso fare sostituzioni)

Utilizziamo quindi la logica **modale** che fornisce **operatori modali** e che può avere formule come argomenti (a differenza della logica classica)

## Logica modale

La semantica della logica modale è basata su **mondi possibili** (dove ogni mondo è una situazione possibile per l'agente)

---

----- Parte introduttiva della logica classica **proposizionale** -----

E' la cosa più semplice.

La **sintassi** (come sono fatte le formule)

Abbiamo delle **proposizioni atomiche** appartenenti ad un insieme P e dei **connettivi logici**

- p
- $\varphi \vee \psi$
- $\neg \varphi$

dove  $p \in P$  e  $\varphi, \psi$  sono formule

(tutti gli altri connettivi sono derivabili da and e not)

La **semantica** si basa sulla nozione di *modello*, verità delle formule rispetto ad ogni modello

Un modello corrisponde ad assegnare un valore di verità (true false) ad ogni simbolo proposizionale

Se  $|P| = n$  ci sono  $2^n$  modelli (quindi è decidibile)

Di solito un modello è un insieme M sottoinsieme di P, quello che c'è in M è vero, quello che non c'è è supposto essere falso

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  $\alpha$  **segue logicamente** da KB:

-  $KB \models \alpha$

se e solo se, in ogni modello in cui KB è vera, anche  $\alpha$  lo è.

Vale il **teorema di deduzione**: date due formule  $\alpha$  e  $\beta$ ,  $\alpha \models \beta$  se e solo se la formula  $(\alpha \Rightarrow \beta)$  è valida.

Ricordarsi anche il **modus ponens**

$$\begin{array}{c} \alpha \Rightarrow \beta \text{ and } \alpha \\ \hline \beta \end{array}$$

---

# Lezione 6

Riprendiamo dall'uso della **logica modale** per modellare agenti.

Ora vedremo:

- Sintassi e semantica della logica modale
- Le proprietà della logica modale
- Come può essere realizzato un sistema basato sulla logica modale

## Logica Modale

Studiata per modellare una verità necessaria o contingente.

Un partito ha la maggioranza, è una verità adesso ma magari domani no -> verità contingente

La radice di 2 è .. è una verità necessaria invece

Vediamo come può essere definita una **logica proposizionale modale**

Sintassi della formula:

- p
- $\varphi \vee \psi$
- $\neg \varphi$
- $\Box \varphi$
- $\Diamond \varphi$

dove p appartiene a P

$\Box$  e  $\Diamond$  sono operatori modali, come argomento hanno una formula della logica

I due operatori modali catturano due diverse modalità di verità: possibilità e necessità

- $\Box$ piove **necessariamente** piove
- $\Diamond$ piove **è possibile** che piova

Questi due operatori sono uno il **duale dell'altro**

$$\Box \varphi \equiv \neg \Diamond \neg \varphi$$

(per dire che è necessario qualcosa non è possibile che sia falso quel qualcosa)

e vale anche il viceversa

$$\Diamond \varphi \equiv \neg \Box \neg \varphi$$

## Semantica

E' basata su un insieme di mondi possibili, ovvero un **modello**, in un modello ci sono tanti mondi possibili, a differenza della logica classica.

Ad ogni mondo è associato un insieme di proposizioni

Esistono delle relazioni di accessibilità tra mondi, sono relazioni binarie che collegano due mondi possibili

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

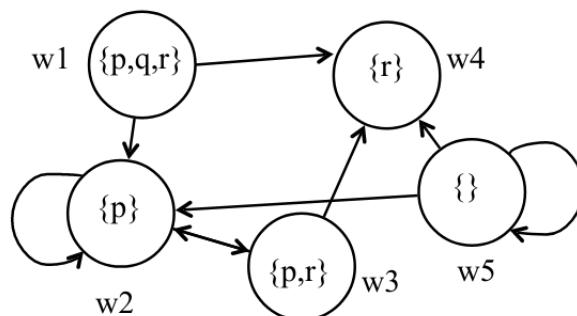
- $W$  è un insieme di mondi,
- $R \subseteq W \times W$  è una relazione di accessibilità fra mondi, e
- $L: W \rightarrow 2^P$  dà l'insieme di proposizioni vere in ogni mondo  $\in W$ .

Esempio in cui i nodi contengono insieme di proposizioni

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

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

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



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

L'insieme  $L$  mi dice cos'è vero in quel mondo,  $\langle w_1, \{p, q, r\} \rangle$  vuol dire che in  $w_1$  è vero  $p, q, r$

La formula  $\Box\varphi$  è vera in un mondo  $w \in W$  se  $\varphi$  è vera in tutti i mondi accessibili da  $w$ .

La formula  $\Diamond\varphi$  è vera in un mondo  $w \in W$  se  $\varphi$  è vera in qualche mondo accessibile da  $w$ .

La **soddisfabilità** di una formula  $\varphi$  è definita rispetto a un modello  $M$  e ad un mondo  $w$  di questo modello con la notazione  $M \models_w \varphi$ .

- $M \models_w p$  iff  $p \in L(w)$
- $M \models_w \varphi \vee \psi$  iff  $M \models_w \varphi$  or  $M \models_w \psi$
- $M \models_w \neg\varphi$  iff  $M \not\models_w \varphi$
- $M \models_w \Box\varphi$  iff  $(\forall w': R(w, w') \Rightarrow M \models_{w'} \varphi)$
- $M \models_w \Diamond\varphi$  iff  $(\exists w': R(w, w') \Rightarrow M \models_{w'} \varphi)$

(le prime 3 sono come la logica classica, le ultime due sono la riscrittura formale di quello che abbiamo scritto prima)

Nessuno mi vieta inoltre che  $w$  e  $w'$  siano diversi

Una formula  $\varphi$  è **valida** se è **vera** in tutte le coppie modello/mondo.

La nozione di validità è analoga a quella di tautologia nella logica proposizionale classica.

I modelli della logica modale sono spesso chiamati **Kripke structures**.

La coppia  $\langle W, R \rangle$  è chiamata **Kripke frame**.

## Proprietà

- **Axioma K**

$$\Box(\varphi \Rightarrow \psi) \Rightarrow (\Box\varphi \Rightarrow \Box\psi)$$

per fare inferenza invece:

- **Necessitation**

da  $\varphi$  inferire  $\Box\varphi$

(se  $\varphi$  è valida allora  $\Box\varphi$  è valida)

- **distributività di  $\Box$**

$$\Box(\varphi \wedge \psi) \equiv (\Box\varphi \wedge \Box\psi) \text{ mentre non distribuisce su}$$

$$\Box(\varphi \vee \psi) \not\equiv (\Box\varphi \vee \Box\psi)$$

Come utilizziamo questi concetti per gli agenti?

$\Box\varphi$  possiamo modellarlo come l'agente crede che  $\varphi$  sia vera (belief)

Generalmente al  $\Box$  diamo un nome mentre a  $\Diamond$  no, tanto sappiamo essere duali

$$\Diamond\varphi \equiv \neg\Box\neg\varphi$$

Noi adatteremo questi operatori modali a seconda della logica

## Logica epistemica (conoscenza e credenze)

**K**<sub>a</sub>φ l'agente *a* sa che φ è vero

**B**<sub>a</sub>φ l'agente *a* crede che φ sia vero

## Logiche Belief-Desire-Intention

**B**<sub>a</sub>φ l'agente *a* crede che φ sia vero

**D**<sub>a</sub>φ l'agente *a* desidera φ

**I**<sub>a</sub>φ l'agente *a* ha l'intenzione φ

## Logiche deontiche (obblighi e permessi)

**O**φ è obbligatorio che φ

**P**φ è permesso che φ

Le lettere maiuscole sono operatori modali, dobbiamo vedere come modellare le proprietà degli agenti tramite la logica modale

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  $\Box\varphi \Rightarrow \varphi$ .

Oppure, potremmo esprimere che *le conoscenze dell'agente non devono essere contraddittorie* aggiungendo l'assioma  $\Box\varphi \Rightarrow \neg\Box\neg\varphi$

E' stato dimostrato che un'altra possibilità di modellare una certa proprietà di un agente oltre ad aggiungere assiomi è possibile restringere la classe dei modelli restringendo i frame.

Ad esempio se vale la riflessività, ovvero ogni mondo accede a se stesso:  $\forall w \in W. (w,w) \in R$

Alcune proprietà principali:

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

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

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

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

Ricordiamo che un **frame** è un normale grafo costituito da  $\langle W, R \rangle$ .

Esiste a volte una corrispondenza tra stringente dei frame e assiomi, per esempio si può dimostrare che l'assioma  $\Box\varphi \Rightarrow \varphi$  corrisponde alla riflessività dei frame.

Corrispondenza:

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

(ci si fida del fatto che siano vere)

I sistemi modali più usati sono i seguenti, e si ottengono combinando le 4 regole sopra descritte (T, D, 4, 5) insieme all'[assioma K](#) che è sempre valido nella logica modale

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

In totale le combinazioni sarebbero 16 ma alcune combinazioni sono equivalenti, quindi si ottengono 11 sistemi modali

A differenza della logica classica in cui sono stati definiti dei metodi come resolution, qui non esistono metodi di prova per fare ragionamenti con la logica modale

## Lezione 7

Come la logica modale viene utilizzata per descrivere proprietà degli agenti  
Inizieremo con la **logica epistemica** per rappresentare le credenze dell'agente

### Logica epistemica

E' la logica della **conoscenza e delle credenze**

Vedremo come questa logica può essere vista come una logica modale

Innanzitutto introduciamo due operatori modali  $K_a$  e  $B_a$  che rappresentano la conoscenza (knowledge) o le credenze (belief) di un agente  $a$ .

- $K_a\varphi$  significa "l'agente  $a$  sa che  $\varphi$  è vero"
- $B_a\varphi$  significa "l'agente  $a$  crede che  $\varphi$  sia vero"

Questi due operatori hanno la stessa semantica del  $\square$ , nel caso ci serva il rombo usiamo il duale.

## Problema della onniscienza logica

Sappiamo che per tutte le logiche modali valgono le seguenti proprietà:

- $K\alpha(\varphi \Rightarrow \psi) \Rightarrow (K\alpha\varphi \Rightarrow K\alpha\psi)$  **assioma K**
- se  $\varphi$  è valida, allora  $K\alpha\varphi$  è valida **necessitation**

Queste due proprietà portano al problema della onniscienza logica, l'agente dovrebbe sapere tutto..

Assumiamo che  $\psi$  sia una conseguenza logica di  $\varphi$ . Allora  $\varphi \Rightarrow \psi$  deve essere valida. Secondo la necessitation, questa formula deve essere conosciuta dall'agente:  $K\alpha(\varphi \Rightarrow \psi)$ .

**Questo significa che l'agente deve conoscere tutte le tautologie, che sono infinite.**

Inoltre se l'agente conosce  $\varphi$ , ossia  $K\alpha\varphi$ , allora, per l'assioma K, deve conoscere  $\psi$ . **Questo significa che la conoscenza dell'agente è chiusa rispetto alla conseguenza logica.**

Noi comunque rimaniamo con la logica modale e vediamo altri aspetti importanti.

## Proprietà Knowledge e Beliefs per la logica modale

Vediamo come gli assiomi D, T, 4 and 5 possono descrivere le proprietà di knowledge e belief.

- L'assioma D (serialità) era il seguente:

$$\square\varphi \Rightarrow \diamond\varphi$$

Noi lo riscriviamo come (essendo  $K = \square$  e  $\diamond = \neg\Box\neg$ ):

$$K\varphi \Rightarrow \neg K\neg\varphi$$

se l'agente conosce  $\varphi$  allora non conosce  $\neg\varphi$  (non può conoscere qualcosa che sia vero e conoscere la stessa cosa in maniera falsa -> consistenza)

La stessa cosa vale per le Belief usando B al posto di K

- L'assioma T (riflessività):

$$K\varphi \Rightarrow \varphi$$

ciò che l'agente conosce è vero

In questo caso non possiamo dire che con le belief valga questa proprietà, lui può credere ad una cosa che in realtà è falsa

- Assioma 4  
 $K\varphi \Rightarrow KK\varphi$

introspezione positiva, se un agente sa qualcosa sa di saperlo

- Assioma 5  
 $\neg K\varphi \Rightarrow K(\neg K\varphi)$

introspezione negativa, se l'agente non sa qualcosa allora sa di non saperlo

Per i Beliefs valgono tutti gli assiomi tranne l'assioma T, usiamo quindi il sistema modale KD45 is known as weak-s5

Vedi esempio slide 7/19 per fare ragionamento con questa logica

*Per fare le dimostrazioni usiamo anche proprietà fondamentali della logica classica come transitività e contrapposizione.*

## Lezione 8

Ragioneremo sul tempo e sulle azioni

### Logiche temporali

Un agente per poter svolgere le sua attività deve essere in grado di **ragionare su tempo e su azioni**.

La logica temporale può essere vista come una logica modale, esistono tante logiche temporali, noi analizzeremo il caso in cui il tempo:

- è discreto
- ha un istante iniziale
- è infinito nel futuro

Inoltre esistono solo **operatori relativi al futuro**

Consideriamo due logiche significative:

## Linear-time logic (LTL)

la struttura del tempo è lineare (da un istante di tempo si passa ad un altro e poi ad un altro ancora e così via.. )

In maniera più formale:

Un modello **M** è una struttura lineare  $\langle S, x, L \rangle$  dove:

- $S$  è un insieme di stati (mondi)
- $x: N \rightarrow S$  è una sequenza infinita di stati ( $N$ : numeri naturali)
- $L: S \rightarrow 2^P$  dà l'insieme delle proposizioni vere in ogni stato

Questo per quanto riguarda la semantica, per la sintassi invece:

- $p$
- $\alpha \vee \beta$
- $\neg \alpha$
- $X\alpha$
- $\alpha U \beta$

dove  $p \in P$  e  $\alpha, \beta \in PLTL$  (propositional linear time logic)

I due operatori modali **X** e **U** hanno la seguente semantica:

- $X\alpha$  ("nexttime  $\alpha$ ") significa che  $\alpha$  è vero nel prossimo istante di tempo
- $\alpha U \beta$  (" $\alpha$  until  $\beta$ ") è vero al tempo  $t$  se e solo se  $\beta$  è vero in un futuro istante  $t'$  e  $\alpha$  è vero in tutti gli istanti fra  $t$  e  $t'$ .

E' possibile derivare altri operatori modali

- $F\alpha \equiv \text{true} U \alpha$  ("prima o poi  $\alpha$ )
- $G\alpha \equiv \neg F \neg \alpha$  ("sempre  $\alpha$ )  $G$  viene definito come duale di  $F$

Alcuni esempi di formule:

La formula  $G(p \Rightarrow Xq)$  è soddisfatta da tutte le sequenze in cui ogni stato in cui  $p$  è vero è seguito immediatamente da uno stato in cui  $q$  è vero.

La formula  $GFp$  è soddisfatta da tutte le sequenze in cui  $p$  è vero infinite volte.

La formula  $Gp \wedge F\neg p$  non è soddisfatta da nessuna sequenza

## Computation Tree Logic (CTL\*)

La struttura del tempo non è lineare ma è un albero (branching-time), quindi da ogni istante si può avere più istanti successori.

Anche in questo caso va sempre verso il futuro..

Si possono usare **path formulas** quindi formule che si riferiscono ad un cammino specifico nell'albero e sono come quelle della LTL descritte prima, ma ci sono anche le **state formulas** che riguardano tutti i cammini infiniti uscenti da uno stato..

## State formulas

- $p$
- $\alpha \vee \beta$
- $\neg \alpha$
- $\mathbf{A}\pi$
- $\mathbf{E}\pi$

dove  $p \in P$ ,  $\alpha, \beta$  sono state formulas, e  $\pi$  è una path formula

- $\mathbf{A}$  significa "per tutti i cammini uscenti da uno stato vale  $\pi$ "
- $\mathbf{E}$  significa "esiste un cammino uscente da uno stato per cui vale  $\pi$ "

A e E sono duali.

## Path formulas

- $\alpha$
- $\pi \vee \rho$
- $\neg \pi$
- $X\pi$
- $\pi U \rho$

dove  $\alpha$  è una state formula, e  $\pi, \rho$  sono path formulas

Sono come le formule della logica LTL

La logica LTL è un sottocaso della CTL\*, essendo però CTL\* troppo complicata da utilizzare si prende un sotto caso di CTL\* e chiamarla CTL

Questa logica CTL non permette le combinazione e l'annidamento degli operatori linear-time, quindi le ultime due regole della sintassi delle Path formulas diventano

- $X\alpha$
- $\alpha U \beta$

dove  $\alpha, \beta$  sono state formula

LTL e CTL sono diverse quindi uno sceglie quale usare in base ai propri obiettivi, *vedi slide 12 per un esempio di cose non esprimibili in una logica e nell'altra si*

Si può vedere che:

- non c'è nessuna formula CTL che sia equivalente alla formula LTL  $FGp$  (c'è uno stato lungo un cammino dopo il quale  $p$  sarà per sempre vero)
- Non potresti avere in CTL  $AGFp$  ma in CTL\* si
- non c'è nessuna formula LTL che sia equivalente alla formula CTL  $AG(EFp)$ .

NOTA. La formula  $FGp$  qui sopra non può essere espressa in CTL perché questo non accetta operatori modali annidati.

## Ragionare su Azioni - Situation Calculus

I termini principali del calcolo delle situazioni sono:

- **Situazioni:** stato del mondo a qualche istante del tempo.  $S_0$  situazione iniziale,  $do(a,s)$  denota una situazione risultante dall'esecuzione dell'azione  $a$  nello stato  $s$

- **Fluenti**: proposizioni il cui valore varia da una situazione ad un'altra. Descrivono lo stato del mondo.  
Esempio: `sta_portando(robot,oro,S0)`. Il robot sta portando oro nella situazione iniziale.
- **Azioni**: azioni che possono essere eseguite nel mondo, causano un cambiamento nello stato del mondo

Chi ha implementato il calcolo delle situazione per realizzare linguaggi di programmazione si è basato sulla logica classica

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

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

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

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

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

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

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

Se uno esegue un'azione questa in qualche modo modifica lo stato e passa ad uno stato diverso, come esprimo in modo parsimonioso che quando cambia qualcosa il resto non cambia? (**Frame problem**)

Reiter ha proposto una soluzione del frame problem introducendo i **Successor State Axioms**, che fanno riferimento ad un fluente per volta. Ad esempio la formula fra parentesi quadre qui sotto mostra come una azione può cambiare il valore del fluente broken. Ipotizziamo che a sia un'azione generica e che sia possibile in uno stato s ( $\text{Poss}(a, s)$ )

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

# Lezione 9-10

## Sistemi Multi-agente

E' impossibile costruire sistemi complessi del tutto centralizzati.. E' bene quindi realizzare sistemi distribuiti tramite:

- modularità
- distribuzione
- astrazione
- intelligenza

I sistemi multi agente appartengono alla categoria DAI (intelligenza artificiale distribuita)

Gli agenti necessitano di essere eseguiti in maniera autonoma e sviluppati indipendentemente

Gli agenti si coordinano e cooperano per la soluzione di problemi, condividendo capacità e lavorando in parallelo, affrontando possibili errori attraverso la ridondanza e rappresentando molteplici punti di vista ed esperienze

Sistemi autonomi di agenti rappresentano soluzioni modulari e riutilizzabili

Quindi:

I sistemi multiagente sono il modo migliore per caratterizzare o progettare sistemi distribuiti di computazione

Gli agenti operano in un **ambiente** con altri agenti e interagiscono tra loro tramite **comunicazione**.

Quindi questi ambienti devono fornire un'infrastruttura di comunicazione e protocolli, essere aperti e distribuiti e gli agenti ospitati sono autonomi, self-interested o cooperativi.

Le **infrastrutture** includono:

- **protocolli di comunicazione**: per permettere agli agenti di comprendere i messaggi scambiati
- **protocolli di interazione**: per permettere agli agenti di svolgere conversazioni, ossia scambi strutturati di messaggi

Un protocollo di comunicazione permette quindi agli agenti di **coordinare** (attività in ambiente condiviso) le proprie azioni e i loro comportamenti (al fine di raggiungere i propri obiettivi)

Possono anche **cooperare** (agenti non antagonisti) o **negoziare** (fra agenti antagonisti o comunque self-interested)

## Linguaggi di comunicazione fra agenti (ACL)

- Fornisce agli agenti i mezzi per scambiarsi informazioni e conoscenza
- Si colloca ad un livello abbastanza alto di astrazione
- Tratta con proposizioni, regole, azioni anziché semplici oggetti
- Descrive uno stato desiderato attraverso un linguaggio dichiarativo

Nella **speech act theory** si trattano la comunicazione come azioni quindi gli atti comunicativi sono spiegati in termini di intenzioni degli agenti, con riferimento a *beliefs, desires, intentions* e altre modalità

Un atto comunicativo ha tre aspetti, si consideri l'esempio "Shut the door!"

- **Locuzione**(locution), l'atto fisico compiuto da chi parla, la produzione di enunciati grammaticali
- **Ilocuzione**(illocution), il significato inteso dell'enunciato da parte del parlante, quello che il parlante desidera esprimere (es. il parlante desidera che l'ascoltatore chiuda la porta)
- **Perlocuzione**(perlocution), l'azione che risulta dalla locuzione (es. l'ascoltatore chiude la porta (forse))

La speech act theory usa il termine **performativa**(performative) per identificare la forza illocutoria dell'enunciato, es.promise,tell,request,convince sono performatives

2 esempi di standard ALC **KQML** (jason) e **FIPA** (jade)

### Knowledge Sharing Effort

**Obiettivo:** sviluppare tecniche, metodologie e strumenti software per **condivisione** della conoscenza e il **riutilizzo** della conoscenza

Nell'ambito della **Knowledge Sharing Effort** si sviluppano le seguenti componenti:

- **KQML**: un linguaggio di interazione di alto livello
- **KIF(Knowledge Interchange Format)**: un linguaggio logico, basato sulla logica del prim'ordine, per esprimere proprietà di oggetti in una base di conoscenza
- **Ontolingua**: un linguaggio per definire ontologie condivise, permettendo di dare significati uniformi su applicazioni diverse agli stessi concetti.Un'ontologia è una cettualizzazione di un insieme di oggetti, concetti e altre entità su cui si esprime la conoscenza, e dei rapporti tra esse

KIF permette di dichiarare proprietà e relazioni di qualcosa nel dominio e proprietà generali di un dominio

esempi: (il linguaggio è Lisp-like)

- "The temperature of m1 is 83 Celsius"  
(= (temperature m1) (scalar 83 Celsius))
- "An object is a bachelor if the object is a man and is not married"  
defrelation bachelor (?x) := (and (man ?x) (not (married ?x)))
- "Any individual with the property of being a person also has the property of being a mammal"  
(defrelation person (?x) :=> (mammal ?x))

KQML è un linguaggio di comunicazione di alto livello e indipendente da

- il meccanismo di trasporto (tcp/ip, RMI, . . . )
- il linguaggio in cui è espresso il contenuto (KIF, Prolog, . . . )

- l'ontologia utilizzata

Consiste in una performativa seguita da un numero di coppie “parola chiave / valore” (la sintassi non è rilevante)

### Esempio di una query da Joe a proposito del prezzo di una quota di IBM

```
(ask-one           ← performativa
  :sender joe
  :receiver stock-server
  :reply-with ibm-stock
  :language LPROLOG   ← il linguaggio di rappresentazione del contenuto
  :content (PRICE IBM ?price)
  :ontology NYSE-TICKS
)
```

Ovviamente esistono varie performatives oltre ad ask-one ed altre parole chiavi (:something)

La sequenza di attivi comunicativi in KQML costituisce un **dialogo** fra i 2 interlocutori, inoltre è possibile inserire in :content un altro messaggio KQML

Sono stati introdotti dei **facilitatori** introducendo una classe speciale di agenti chiamati **“communication facilitators”** che dispongono di un insieme di performatives per inoltrare messaggi, trovare servizi, ecc.

Inizialmente **non disponeva di semantica** (dava solo un contenitore in cui mettere il messaggio non dando significato né al contenuto né alle performatives e questo ovviamente non va bene) Labrou e Finin introducono una semantica per KQML in termini di **precondizioni** (preconditions), **postcondizioni** (postconditions) e **condizioni di completamento** (completions conditions)

Semantica di KQML

Precondizioni, postcondizioni e condizioni di completamento descrivono lo stato degli agenti in un linguaggio basato su stati mentali espressi mediante attitudini (beliefs,knowledge,desire intention) e descrizioni di azioni (per inviare e elaborare un messaggio).

Non è fornita un modello semantico per gli stati mentali espressi mediante attitudini

**Semantica di tell( $A, B, X$ )** <https://arxiv.org/pdf/cs/9809034.pdf>

**Pre(A)**  $BEL(A, X) \wedge KNOW(A, WANT(B, KNOW(B, S)))$

**Pre(B)**  $INT(B, KNOW(B, S))$

dove  $S$  può essere  $BEL(B, X)$  oppure  $\neg BEL(B, X)$

**Post(A)**  $KNOW(A, KNOW(B, BEL(A, X)))$

**Post(B)**  $KNOW(B, BEL(A, X))$

**Completion**  $KNOW(B, BEL(A, X))$

Come precondizione in questo caso abbiamo che B dice qualcosa se è interrogato su qualcosa Una Proactive tell potrebbe avere come pre condizioni

**Pre(A)**  $BEL(A, X)$

**Pre(B)** vuoto

FIPA

Iniziativa a stampo Europeo al fine di produrre degli standard per il software per agenti interagenti ed eterogenei e sistemi basati su agenti

Conta 22 atti comunicativi, come ACL sono molto simili, la differenza principale sta nella semantica (che in KQML era arrivata un po' tardi) inoltre non introduce i facilitatori

Le performative inoltre vengono categorizzate in modo da sapere quando usarle

- **passing info**
- **request info**
- **negotiation**
- **performing actions**
- **error handling**

La principale differenza comunque sta nella semantica utilizzata, in FIPA si utilizza infatti il **Semantic Language(SL)**

SL è una logica multimodale quantificata con operatori modali:

$B_i\varphi$   $i$  crede  $\varphi$

$U_i\varphi$   $i$  è incerto su  $\varphi$  ma pensa che  $\varphi$  è più probabile di  $\neg\varphi$

$C_i\varphi$   $i$  desidera (choice, goal) che  $\varphi$  valga correntemente

Per permettere il ragionamento su azioni, l'universo del discorso coinvolge sequenze di eventi (actions) e vengono introdotti i seguenti operatori a tal proposito

- **Feasible(a, $\varphi$ )** a può occorrere e se occorre,  $\varphi$  sarà vera dopo tale occorrenza
- **Done(a, $\varphi$ )** a è appenaoccorsa e  $\varphi$  è vera dopo tale occorrenza
- **Agent(i,a)** i è il solo agente che esegue l'azione a

Sulla base di belief, choice ed eventi viene definito il concetto di goal persistente (*persistent goal*)  $PG; \varphi$ . L'intenzione  $I; \varphi$  è definita come un goal persistente che impone all'agente di agire

La semantica degli atti comunicativi è specificata come un insieme di formule SL che descrivono le precondizioni di fattibilità (**feasibility preconditions FP**) e gli effetti razionali (**rational effects RE**).

Per valutare la conformità di un atto comunicativo basta valutare le FP per il mittente

## Protocolli di Interazione

Definiscono come costruire una conversazione

Nelle conversazioni umane si possono notare dei pattern.

FIPA standardizza un gran numero di protocolli di interazione

I protocolli sono solitamente modellati come macchine a stati finiti

Le specifiche di FIPA definiscono i protocolli attraverso AUML, una estensione di UML per gli agenti

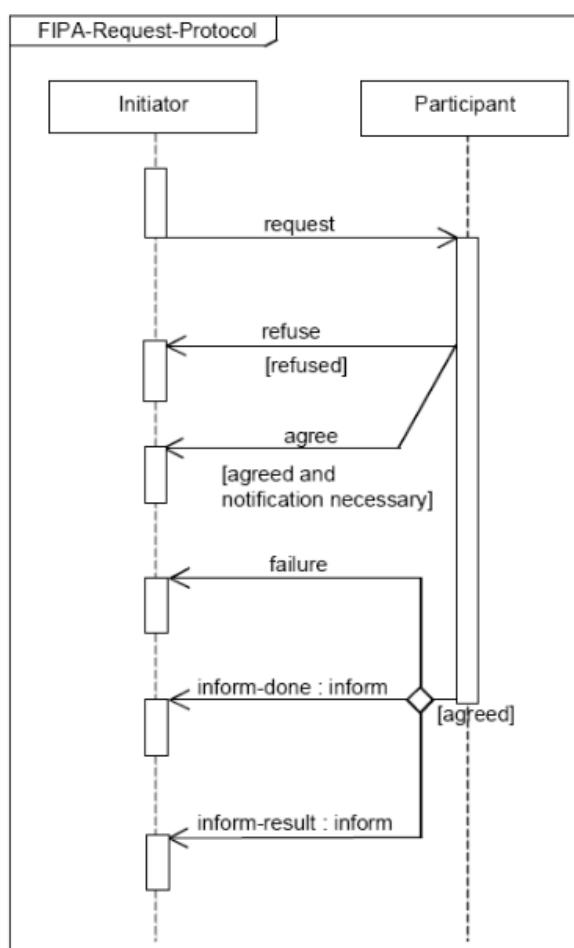


Figure 1: FIPA Request Interaction Protocol

Un sistema alternativo di comunicazione è definibile disponendo un ambiente programmabile in cui gli agenti ricevono informazioni e lasciano informazione (Cartago)

## Semantica sociale per la comunicazione

E' possibile notare che l'eterogeneità e la competizione di agenti creano problemi nei sistemi con semantica basata a stati mentali

Vediamo quindi l'approccio sociale per la comunicazione

L'approccio Sociale considera le **conseguenze sociali** di eseguire un atto comunicativo  
la comunicazione non è verificabile negli stati mentali ma negli effetti pubblici (gli elementi di verifica coinvolgono fatti oggettivi)

*es. comunico la data dell'esame, io ci posso credere o meno ma se non mi viene detta o se non c'era l'esame alla data stabilita mi incazzo*

Buone caratteristiche in un ACL

- **Formal:** chiarezza della specifica per guidare al meglio colui che realizza un sistema
- **Declarative:** descrivere cosa(what)piuttosto che come(how)
- **Verifiable:** determinare se un agente sta agendo in accordo con una data semantica
- **Meaningful:** trattare la comunicazione secondo il suo significato e non come arbitrari token che devono essere ordinati in una qualche maniera

L'approccio mentalistico NON è verificabile (un agente non può determinare se un altro agente sta agendo secondo la semantica dei messaggi, FIPA infatti non considera il rational effect)

### Verificare un protocollo:

- necessita di una semantica basata su fatti "osservabili"
- Una semantica che sia indipendente dagli stati mentali dei partecipanti
- Una semantica che sia verificabile sia da un partecipante sia da una terza parte che deve monitorare l'interazione
- Una semantica che sia adatta per open MAS
- Una semantica che renda osservabile la violazione di una specifica da parte di un partecipante all'interazione, senza che si debba prendere in esame la natura dell'agente o eseguire introspezione dello stato mentale dell'agente

L'approccio è basato sui **practical commitments** (impegni) tra agenti: un agente (il debtor) si impegna verso un altro agente (il creditor) a rendere vero un qualche fatto o ad eseguire una qualche azione

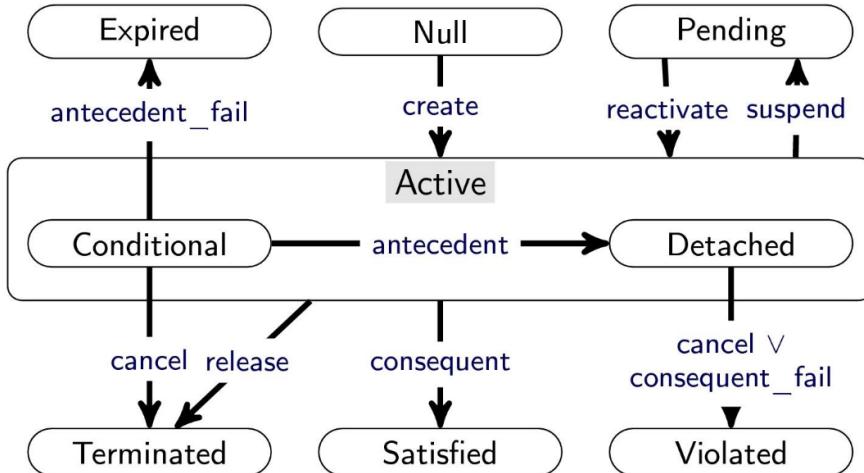
Nella definizione dei commitments viene inclusa una notazione di **contesto sociale** ed è possibile specificare dei **metacommitments** per catturare un'ampia varietà di relazioni legali e sociali (come gli impegni, le richieste, i diritti, i privilegi, i poteri)

I commitments sono manipolabili degli agenti e influenzano il loro comportamento, vengono definiti come **una quadrupla**

**C(debtor, creditor, antecedent, consequence)**

il debtor è socialmente legato al creditor a rendere vera la condizione conseguente se la condizione antecedente è vera

Ciclo di vita di un commitment:



Un agente può:

- creare un commitment, ed è creato necessariamente dal debtor a differenza degli obblighi
- cancellare
- rilasciare (è eseguita dal creditor)
- delegate( $x, y, z, r, u$ ) è eseguita da  $x$  e causa  $C(z, y, r, u)$
- assign (analogo alla delegate ma il trasferimento è il trasferimento del creditor ad un'altra parte) quindi assign( $x, y, z, r, u$ ) è eseguita da  $y$  e causa  $C(x, z, r, u)$

Vengono anche stabiliti dei postulati logici che devono essere rispettati dai commitment

Anche i protocolli vengono definiti sulla base dei commitment che deve associare un significato alle azioni in termini di manipolazione di commitment che portano con su impegni su azioni/fatti verificabili dalle due controparti.

Si comporta molto bene quando i protocolli sono molto flessibili, e quindi descriverli tramite un automa a stati finiti risulterebbe oneroso

**Il solo vincolo che un protocollo a commitment deve soddisfare perché è una interazione sia di successo è che tutti commitment siano discharged.**

Un **run** di successo del protocollo è una sequenza (finita) di azioni che portano in uno stato in cui tutti i commitment sono **fulfilled**, ossia non ci siano commitment pendenti (commitments  $C(x, y, T, p)$  dove  $p$  non è stata resa vera)

Contract Net Protocol

cfp means  $\text{create}(C(i, p, \text{propose}, \text{accept} \vee \text{reject}))$

propose means  $\text{create}(C(p, i, \text{accept}, \text{done} \vee \text{failure}))$

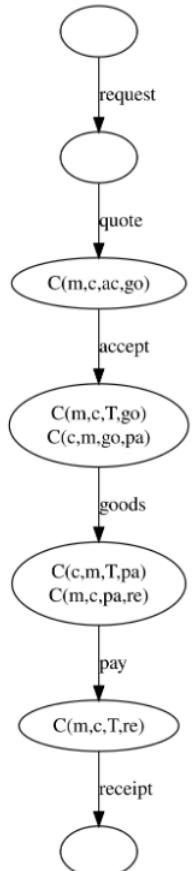
refuse means  $\text{release}(C(i, p, \text{propose}, \text{accept} \vee \text{reject}))$

accept means none  
 reject means release( $C(p, i, \text{accept}, \text{done} \vee \text{failure})$ )  
 done means none  
 failure means none

## NetBill Protocol

- **request** means none
- **quote** means  $\text{create}(C(m, c, \text{accept}, \text{goods}))$
- **accept** means  $\text{create}(C(c, m, \text{goods}, \text{pay}))$
- **reject** means  $\text{release}(C(m, c, \text{accept}, \text{goods}))$
- **goods** means  $\text{create}(C(m, c, \text{pay}, \text{receipt}))$
- **pay** means none
- **receipt** means none

**Run:** request; quote; accept; goods; pay; receipt



## Lezione 11

### ...FIPA

Completiamo il discorso su FIPA

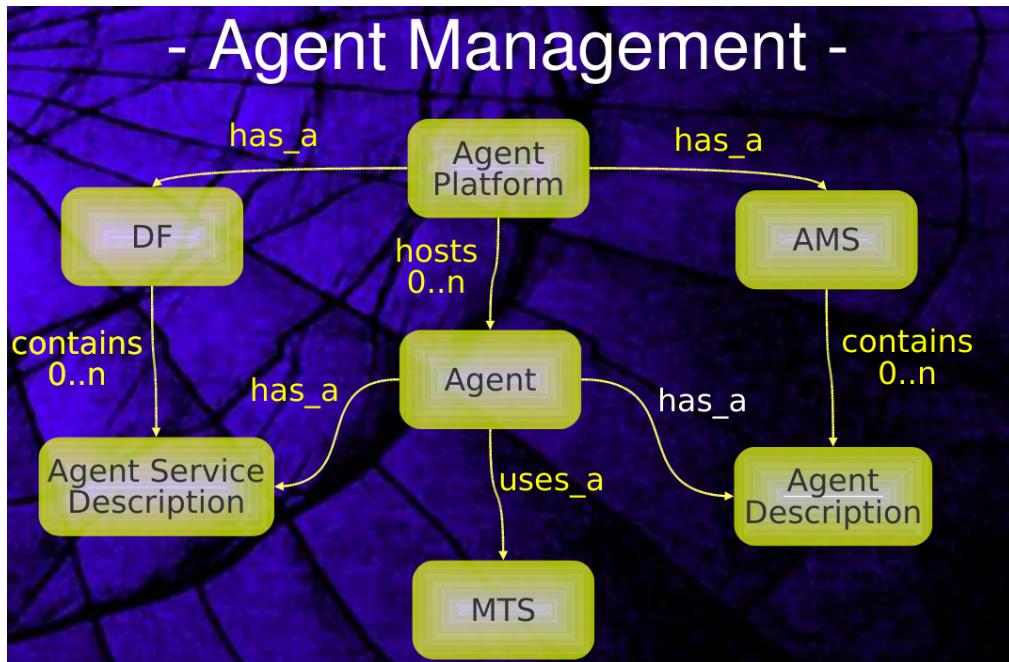
Missione di FIPA: The promotion of technologies and interoperability specifications that facilitate the end to end interworking of intelligent agent systems in modern commercial and industrial settings.

Diventa uno standard nel 2002

- Obiettivi di FIPA:

- Agent Lifecycle Management
- Message Transport
- Message Structure
- Interagent Interaction Protocols
- Ontologies
- Security

Gli agenti sono fuori dallo scope di FIPA!!



Dove con

- DF si intende Directory Facilitator
- AMS Agent Management System
- MTS Message Transport Service

DF e AMS contengono varie informazioni sugli agenti,

- DF: Name Location Services Protocols Ontologies Lease-time Scope
- AMS: Name Owner State

## JADE

E' diventato praticamente uno standard oltre ad essere compliant con gli standard FIPA e si occupa di tutto ciò che riguarda un sistema multi-agente oltre a fornire un debugger

**Nasconde ai programmatori tutti gli aspetti di programmazione FIPA!**

Ci basta usare le primitive di JADE per costruire messaggi ACL e anche i protocolli di interazioni vengono già forniti

Rende anche trasparente gli aspetti di località in rete degli agenti creando una piattaforma per gestire i container

Come lanciare la piattaforma:

```
java jade.Boot -gui -nomtp -port 1200 W1:x.y.W(20) W2:x.y.W(10)
```

inserendo eventualmente come classpath il bin di jade

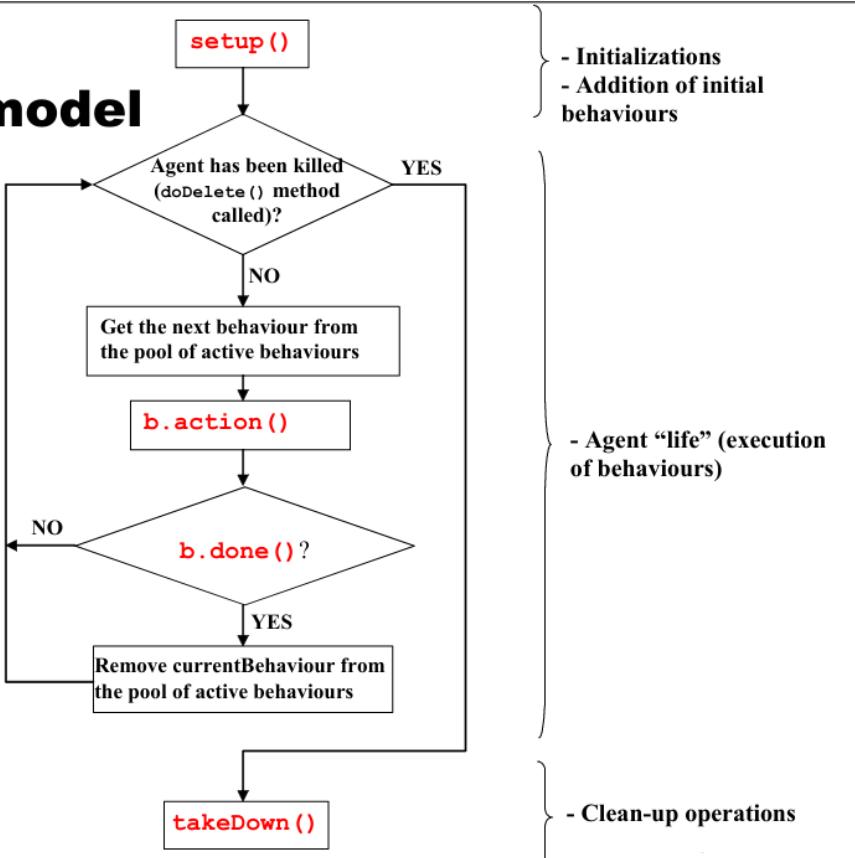
Essendo open source è stato stuprato un po' da tutti, pure da protègè..

## Lezione 12-13

(ascolta a x2 queste due lezioni perchè sono solo esercizi)

# The agent execution model

*Highlighted in red  
the methods that  
programmers have  
to/can implement*



Un Behaviour è una semplice classe con metodo `action()` 'cosa fare' e `done()` 'è stato fatto?'  
 Ci sono un po' di behaviour standard che dicono come e quando le action sono da eseguire  
 (one shot, cyclic, semplici (specifici il done) Waker (dopo tot tempo) e Ticker (ogni tot  
 tempo))

# Lezione 14

Martelli

## Modellazione di Agenti BDI

2 Approcci:

- Cohen and Levesque's intention logic
- Rao and Georgeff's BDI logics

Struttura di agenti che permette di combinare diversi aspetti, gli agenti fanno parte della AI ma spesso l'AI si limitava a considerare le proprietà degli agenti uno per volta (es la *pianificazione*)

La logica ci serve per rappresentare aspetti dinamici tra gli agenti. For example, it will need to show how an agent's information and intentions are related; how an agent's cognitive state changes over time; how the environment affects an agent's cognitive state; and how an agent's information and intentions lead it to perform actions

I metodi formali giocano 3 ruoli:

- **specificare** sistemi
- **programmare direttamente** sistemi, esempio è il prolog che utilizza la logica classica per programmare un sistema
- **verificare** sistemi, lo logica può servire per verificare che le proprietà di un agente siano quello che ci si aspetta che abbia

Oggi parliamo dell'aspetto della **specificità di sistemi**

### Cohen and Levesque's intention logic

Cohen and Levesque ritengono che il comportamento razionale dovrebbe essere analizzato in termini di **beliefs, desires and intentions (BDI)**.

Proprietà delle intenzioni:

- 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.
- 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.
- In certe circostanze, gli agenti credono che riusciranno a soddisfare le loro intenzioni.
- Gli agenti non si aspettano tutti i side-effects delle loro intenzioni

### I side-effects non sono intenzioni

Es l'intenzione di un agente è di curarsi il dente, il sentire il dolore non fa parte dell'intenzione dell'agente, è un side effect dell'intenzione di curarsi il dente

### La Logica

#### Questa logica ha 4 operatori modali primari

- $(BEL i \varphi)$  agent  $i$  believes  $\varphi$
- $(GOAL i \varphi)$  agent  $i$  has goal (desire) of  $\varphi$
- $(HAPPENS \alpha)$  action  $\alpha$  will happen next
- $(DONE \alpha)$  action  $\alpha$  has just happened

Le **intention** come vediamo non è un operatore primario

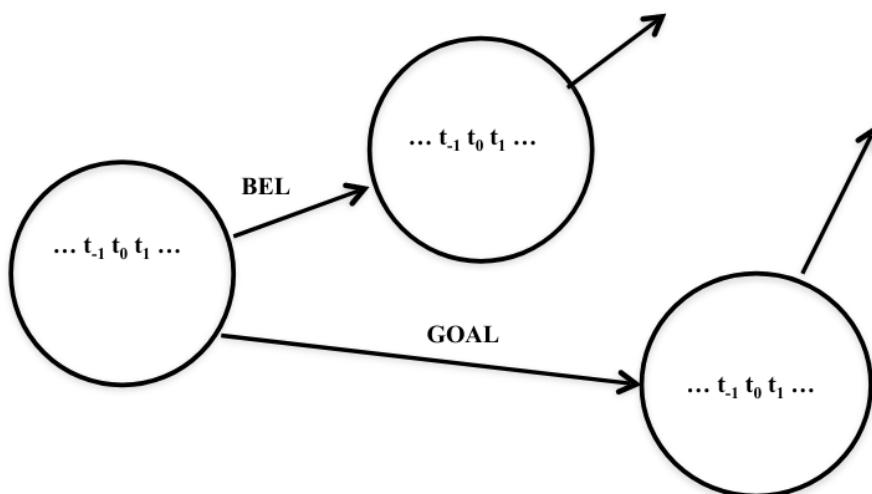
### La semantica

E' quella dei mondi possibili

Le relazioni di accessibilità tra i mondi hanno etichette (BEL, GOAL, ..)

Questa semantica si basa su una logica temporale, quindi ogni mondo è basata sulla logica temporale LTL, è quindi costituito da una sequenza di istanti di tempo lineare

Dentro ogni mondo sappiamo che ci sono le verità di alcune asserzioni



La semantica di BEL e GOAL è data nel solito modo, assegnando ad ogni agente una belief accessibility relation, e una goal accessibility relation.

La belief accessibility relation è KD45 (weak S5).

La goal accessibility relation è KD.

Si assume che la goal relation di ogni agente sia un sottoinsieme della sua belief relation.

Axiom D: non contraddizione (serialità)  $BEL \varphi \equiv \neg BEL \neg \varphi$

## Modalità Derivate

Gli operatori standard di LTL nel tempo futuro G (always) and F (sometime), possono essere definiti come abbreviazioni, con l'aggiunta di un altro operatore: LATER

$F\varphi \equiv \exists \alpha. (\text{HAPPENS } \alpha; \varphi?)$  esiste una sequenza di azioni  $\alpha$  tale che, dopo la sua esecuzione

vale  $\varphi$ . ? è

l'operatore di test.

$G \varphi \equiv \neg F \neg \varphi$

$(\text{LATER} \varphi) \equiv \neg \varphi \wedge F \varphi$

E' possibile derivare anche un operatore di precedenza temporale (BEFORE $\varphi\psi$ ), ossia  $\varphi$  vale prima di  $\psi$ :

$(\text{BEFORE } \varphi \psi) \equiv \forall \alpha. (\text{HAPPENS } \alpha; \psi?) \Rightarrow \exists \beta. (\beta \leq \alpha) \wedge (\text{HAPPENS } \beta; \varphi?)$

## Achievement goals

Un Goal in cui si vuole arrivare in una situazione in cui qualcosa è vero (il planning è questo)

$(A\text{-GOAL } i \varphi) \equiv (\text{GOAL } i (\text{LATER } \varphi)) \wedge (\text{BEL } i \neg \varphi)$

A-GOAL sta per Achievement goal

Vorremmo rendere l'Achievement Goal un **Goal Persistente**

$(P\text{-GOAL } i \varphi) \equiv (\text{GOAL } i (\text{LATER } \varphi)) \wedge (\text{BEL } i \neg \varphi) \wedge$   
[BEFORE ((BEL  $i \varphi$ )  $\vee$  (BEL  $i \mathbf{G} \neg \varphi$ ))  
 $\neg(\text{GOAL } i (\text{LATER } \varphi))]$

Un agente ha il **persistent goal**  $\varphi$  se:

- Ha un goal che  $\varphi$  prima o poi diventerà vero, e crede che  $\varphi$  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

## Intenzioni rispetto alle azioni

$(\text{INTEND } i \alpha) \equiv (P\text{-GOAL } i [\text{DONE } i (\text{BEL } i (\text{HAPPENS } \alpha)? ; \alpha)], \text{ con } \alpha \text{ azione}$

L'agente  $i$  intende eseguire l'azione  $\alpha$  se **ha un goal persistente di raggiungere uno stato in cui lui aveva appena creduto di stare per eseguire  $\alpha$ , e subito dopo  $\alpha$  viene eseguita**. Il criterio che l'agente sia committed a credere di stare per eseguire l'azione  $\alpha$ , evita che l'agente sia tenuto ad eseguire l'azione accidentalmente o senza pensarci

## Rao and Georgeff's BDI logic

A differenza di quella precedente anche le Intend sono trattate come modalità di base, non è quindi costruita come prima

Inoltre la struttura temporale dei mondi cambia, le strutture sono temporali branching time (ad albero)

Con questa logica basata su [CTL\\*](#) possiamo descrivere dei **commitment** in questa maniera:

- Un agente **blindly committed (fanatical)** mantiene le sue intenzioni finché non arriva a credere di averle soddisfatte.  
 $\text{INTEND(AF}\varphi\text{)} \Rightarrow A(\text{INTEND(AF}\varphi\text{)} \cup \text{BEL}(\varphi))$
- Con **single-minded commitment**, un agente mantiene le sue intenzioni finché crede che queste possano essere realizzabili.  
 $\text{INTEND(AF}\varphi\text{)} \Rightarrow A(\text{INTEND(AF}\varphi\text{)} \cup (\text{BEL}(\varphi) \vee \neg\text{BEL(EF}\varphi\text{)}))$
- Questa formulazione può essere modificata definendo un agente **open-minded** come un agente che mantiene le sue intenzioni finché queste intenzioni sono ancora i suoi goal.  
 $\text{INTEND(AF}\varphi\text{)} \Rightarrow A(\text{INTEND(AF}\varphi\text{)} \cup (\text{BEL}(\varphi) \vee \neg\text{GOAL(EF}\varphi\text{)}))$

## Lezione 15

### Model Checking

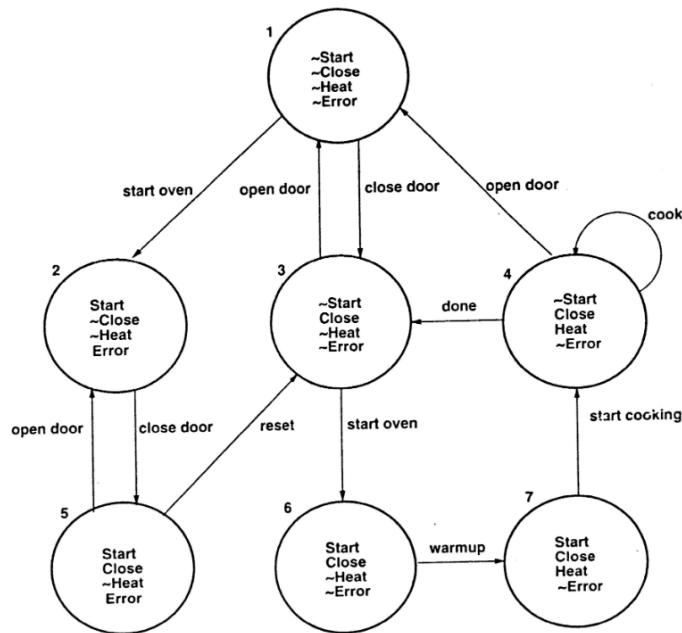
Fin'ora abbiamo visto metodi formali per definire i comportamenti degli agenti, ora li usiamo per il model checking per verificare delle proprietà di un sistema

E' un metodo utilizzato per **verificare le proprietà** di sistemi, molto spesso per sistemi concorrenti

E' un approccio basato sui modelli, e quindi sull'aspetto semantico. l'idea è che se abbiamo un modello M in una logica L e una formula  $\varphi$  di L, vogliamo determinare se  $\varphi$  è valida in M  
Normalmente L è una logica temporale (LTL o CLT)

Il comportamento di un sistema (concorrente)  $\pi$  può essere formulato come un transition system consistente di un insieme di stati, un insieme di transizioni fra gli stati e una funzione che associa ad ogni stato un insieme di proposizioni vere in quello stato.

Es di un Transition system di un microonde. A differenza di un modello della logica modale, che è costituito da un insieme di mondi che contengono ciò che è vero e ciò che è falso ma non dando nomi agli archi



## Model checking per LTL

Supponiamo di avere un **sistema  $\pi$**  formulato come un **transition system**, e una formula **LTL  $\varphi$**  che descrive una proprietà che vogliamo verificare. Per mostrare che  $\varphi$  vale per  $\pi$ , dobbiamo dimostrare che  $\varphi$  è vera per ogni esecuzione di  $\pi$ . Siccome sappiamo che un modello in LTL è una sequenza infinita di stati, dimostrare che  $\varphi$  è vera per ogni esecuzione di  $\pi$ , significa che ogni cammino infinito di  $\pi$  è un modello di  $\varphi$ . Quindi per ogni cammino  $\varphi$  deve essere vera.

Di solito, la dimostrazione che una formula  $\varphi$  è valida, ossia è vera per tutti i cammini infiniti del modello, viene fatta per refutazione: **dimostrare che  $\neg\varphi$  è insoddisfacibile nel modello**, ossia non esiste nessuna computazione che soddisfi  $\neg\varphi$ .

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

Esempi di formule  $\varphi$  da verificare:

- $G(Start \Rightarrow F Heat)$ , per ogni cammino, da ogni stato in cui vale Start si raggiunge uno stato in cui vale Heat.
- $(\neg Heat) U Close$ , è impossibile che il forno diventi caldo finché la porta è aperta

## Automi di Büchi

Sono automi che accettano stringhe infinite, per il resto è un automa a stati finiti classico

L' automa  $\langle \Sigma, S, \delta, S_0, F \rangle$  consiste di un alfabeto  $\Sigma$ , un insieme di stati  $S$ , una funzione di transizione  $\delta \subseteq S \times \Sigma \times S$ , un insieme di stati iniziali  $S_0$  e un **insieme di stati di accettazione  $F$** . La differenza è in  $F$  perché un automa a stati finiti si ferma quando arriva ad uno stato finale, qui invece accetta stringhe infinite. Una stringa infinita viene

accettata dall'automa se appare infinite volte in  $w$  almeno uno stato di  $F$ , quindi se questa stringa contiene infinite volta uno stato di  $F$ .

Una stringa infinita è una stringa finita che ad un certo punto ricomincia e va in loop

Il sistema modellato e le proprietà da verificare sono rappresentati allo stesso modo, ossia con automi di Büchi.

- Infatti il transition system di un sistema  $\pi$ , ossia il modello, può essere visto come un automa di Büchi  $\mathcal{B}(\pi)$  i cui stati sono tutti di accettazione, quindi qualunque run che finisce in un loop è visto come una terminazione.
- Inoltre, esiste un algoritmo\* per tradurre una formula LTL  $\varphi$  in un automa di Büchi  $\mathcal{B}(\varphi)$  che accetta esattamente le sequenze infinite che sono modelli di  $\varphi$ .

Operativamente:

siano dati il transition system  $\pi$  e la formula LTL  $\varphi$  da verificare

- costruire i due automi di Büchi  $\mathcal{B}(\pi)$  e  $\mathcal{B}(\neg\varphi)$
- costruire l'automa di Büchi che accetta l'**intersezione** dei linguaggi accettati da  $\mathcal{B}(\pi)$  e  $\mathcal{B}(\neg\varphi)$  (il prodotto dei due automi di Büchi)
- ogni run dell'intersezione è sia un run infinito di  $\pi$  sia un modello di  $\neg\varphi$
- se l'intersezione è vuota, allora  $\varphi$  vale per  $\pi$ , altrimenti un run dell'intersezione fornisce un controsenso

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

Questo algoritmo può essere usato ovviamente sia per trovare dei controsensi per verificare che una proprietà sia valida o meno oppure (dando  $\varphi$  invece che  $\neg\varphi$ ) per pianificare

Oltre al model checking basato su LTL, sono stati proposti e sviluppati numerosi model checker basati su approcci diversi. In particolare, possiamo citare l'approccio per verificare formule CTL, basandosi direttamente sulla decomposizione della struttura di Kripke (e quindi direttamente sul transition system) in componenti strettamente connesse.

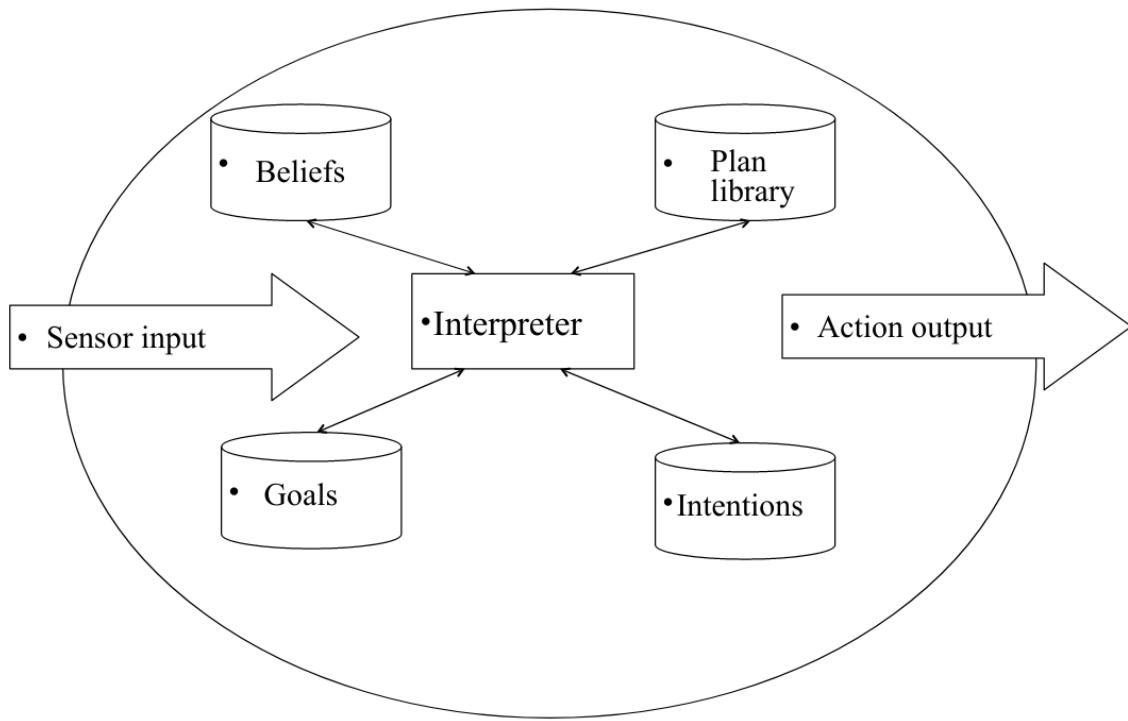
# Lezione 16

Linguaggi e architetture per agenti BDI

## Procedural Reasoning System (PRS)

Una delle **prime architetture** per agenti che abbia fatto uso del paradigma BDI.  
Questa architettura può essere poi adattata a diversi tipi di linguaggi

### L'architettura del PRS



L'**interprete** riceve input dall'esterno e può eseguire azioni nell'ambiente esterno  
Essendo un architettura BDI sono presenti dei “db” contenenti Beliefs, Desire (Goals)  
Intentions, vedremo dopo cosa sono i Plan

L'interprete lavora sulla **coda di eventi**, gli eventi possono essere sia esterni che interni  
(come aggiunta o cancellazione di belief o goal)

I **belief** sono rappresentati come fatti del Prolog: in pratica come **atomi della logica del prim'ordine**.

I Piani contenuti nella Plan library sono built-in quando si comincia ad eseguire il programma, l'agente non pianifica, ci sono già e vengono forniti dal programmatore -> **non esiste un meccanismo per pianificare**

I piani hanno:

- **nome**
- **condizione di invocazione**: a triggering event (goal)
- **precondizioni**: le condizioni che devono valere prima di avviare l'esecuzione del piano
- **body**: è una sequenza di simple plan expressions, ossia:
  - una azione atomica o
  - un sotto-goal

Un piano può essere visto come una procedura (un metodo) che ha però un trigger event che lo fa scattare

Esempio

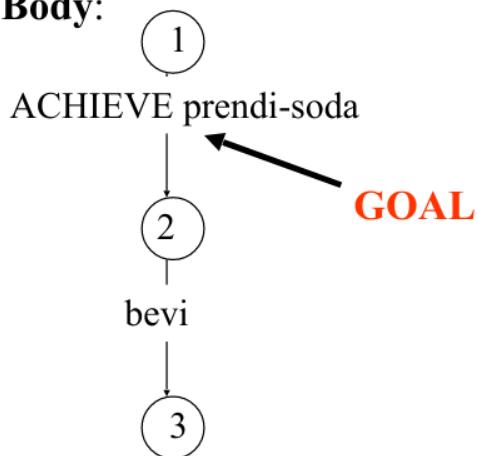
### Due piani per spegnere la sete.

**Name:** bevi-soda

**Invoc:** ACHIEVE spegni-sete

**Precond:** avere-bicchiere

**Body:**

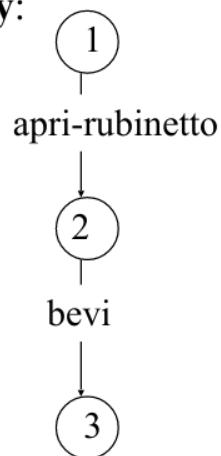


**Name:** bevi-acqua

**Invoc:** ACHIEVE spegni-sete

**Precond:** avere-bicchiere

**Body:**



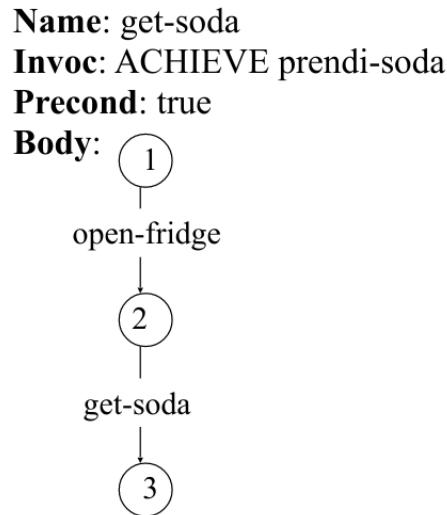
*bevi* e *apri-rubinetto* sono azioni.

Entrambi questi piani hanno come trigger ACHIEVE spegni-sete

Il piano di bevi-soda però prevede un goal (bisogna prima procurarsi la soda)

Bisogna quindi predefinire un piano per soddisfare questo goal

## A plan to get soda



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 una intenzione, ed è aggiunto alla intention structure.

A ogni passo del loop principale, l'interprete sceglie un piano (parzialmente eseguito) nella intention structure, e ne esegue un passo

I passi principali del control loop del PRS sono i seguenti:

- aggiorna belief e goal secondo gli eventi nella event queue
- i cambiamenti dei goal e belief “trigger” diversi piani
- uno o più dei piani applicabili sono scelti e messi nella intention structure
- scegliere una intenzione (task) dalla intention structure
- eseguire un passo di quel task. Questo può risultare in:
  - esecuzione di una azione primitiva, o
  - scelta di un nuovo subgoal, che è “posted” nella event queue.

Quindi l'esecuzione dei task può essere “interleaved”, come in un sistema multithreaded

## AgentSpeak(L)

Linguaggio di programmazione per agenti BDI, basato su clausole di Horn come prolog.  
Da questo linguaggio è nato Jason  
Tanto lo vedremo poi..

## Agent-oriented programming (AOP)

Un nuovo paradigma di programmazione, viene presentato:

- un linguaggio formale per descrivere stati mentali;
- un linguaggio di programmazione AGENT-0 per definire agenti.

Possiamo considerare AOP come basato su un paradigma BDI

Ci sono **due categories mentali** principali: **belief** and **obligation** (o commitment).

Decision (choice) è trattata come una obligation a se stesso.

Un'ulteriore categoria, che non è un costrutto mentale, è capability (cosa è capace di fare quell'agente)

**Le formule fanno esplicitamente riferimento al tempo**

$\text{holding}(\text{robot}, \text{cup})_t$

### Belief

$B_a^t \varphi$  : agent a believes  $\varphi$  at time t.

### Obligation

$OBL_{a,b}^t \varphi$

means that at time t agent a is obligated, or committed, to agent b about  $\varphi$ .  $\varphi$  can be a fact representing an action.

### Decision (choice)

Decision is defined as commitment to oneself:

$DEC_a^t \varphi =_{\text{def}} OBL_{a,a}^t \varphi$

### Capability

The fact that at time t agent a is capable of  $\varphi$  is represented by

$CAN_a^t \varphi$

Example:  $CAN_{\text{robot}}^t \text{open(door)}^8$

## AOP: AGENT-0

Un programma in AGENT-0 si riferisce ad un singolo agente, il cui nome è implicito in tutte le istruzioni.

I principali costrutti sono:

**Facts:** (t atom)      atom holds at time t

**Private actions:** (DO t p-action)

**Communicative actions:**

    (INFORM t a fact)      at time t inform agent a that fact

**Commitment**

AGENT0 è implementato come una estensione al LISP

Ogni agente in AGENT-0 ha 4 componenti:

- a set of **capabilities** (things the agent can do)
- a set of **initial beliefs**
- a set of **initial commitments** (things the agent will do)
- a set of **commitment rules** (piani)

The commitment rule set is the "program", that determines how the agent acts.

La forma di una commitment rule e':

(COMMIT message-cond mental-cond (agent action)\*)

```
(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* al tempo *time*.

Se l'agente ha un commit aspetta che scatti il tempo *time* specificato prima di eseguirlo,  
quindi:

L'interprete esegue il seguente ciclo:

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

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

# Lezione 17

## Linguaggi logici per agenti

Richiamiamo quanto detto quando parlavamo di metodi formali per programmare agenti.

3 ruoli per che giocano i metodi formali:

- **specificare** sistemi
- **verificare** sistemi
- **programmare direttamente** sistemi, descrizione formale del sistema e del linguaggio di programmazione e usare direttamente la formulazione degli aspetti dell'agente e formalizzarli in una qualche logica -> linguaggio logico che possa essere eseguito (es prolog)

## GOLOG

E' un linguaggio di programmazione basato sul **calcolo delle situazioni** (come descritto nella Lezione 8)

E' utilizzato per programmare robot di alto livello e agenti software intelligenti

Esempio:

### Preconditions

$$\begin{aligned} Poss(pickup(x), s) \equiv \\ [(\forall z) \neg holding(z, s)] \wedge nexto(x, s) \wedge \neg heavy(x) \end{aligned}$$

### Successor state axioms (uno per ogni fluente)

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

La formula dice semplicemente che l'oggetto è rotto se è caduto ed era fragile oppure se era rotto e nessuno l'ha riparato.

Il successor state axioms e' una soluzione proposta al problema dei frame, considerando la maggior parte dei fluenti da uno stato a quello successivo rimangono invariati, utilizzando questa formulazione si associa una regola per ogni fluente riducendo la complessità

GOLOG permette di definire azioni complesse usando l'abbreviazione **Do( $\delta$ ,  $s$ ,  $s'$ )** dove  $\delta$  è una **espressione di azione complessa**

Eseguendo  $\delta$  si passa da  $s$  ad  $s'$

Do (*maiuscolo*) sono azioni complesse, do (*minuscolo*) azioni semplici

**Primitive actions:**

$$Do(a, s, s') = Poss(a, s) \wedge s' = do(a, s)$$

dove Poss(a,s) specifica se è possibile eseguire l'azione a in s

**Test actions:**

$$Do(\varphi?, s, s') = \varphi \wedge s = s'$$

per verificare se una formula e' vero o falsa, ovviamente questa condizione viene valutata in uno stato ( $s = s'$ )

**Sequence:**

$$Do([\delta_1; \delta_2], s, s') = \exists s''. Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$$

Sequenza di azioni

**Nondeterministic choice:**

$$Do((\delta_1 | \delta_2), s, s') = Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$$

devo scegliere tra  $\delta_1$  e  $\delta_2$

**Nondeterministic choice of action:**

$$Do((\pi x) \delta(x), s, s') = \exists x. Do(\delta(x), s, s')$$

( $\pi x$ ) permette di specificare un'azione x

## Eseguire un Programma in Golog

Ci sono degli assiomi utilizzati e il programma, dall'insieme degli assiomi deriva quello che si ottiene dall'esecuzione del programma

Eseguire un programma significa stabilire la seguente derivazione

$$\text{Axioms} \models \exists s. Do(\text{program}, S_0, s)$$

dove  $S_0$  è la situazione iniziale.

Il risultato alla fine è una sequenza di azioni e stati

Un programma eseguito con successo restituisce un legame per s:

$$s = do(a_1, \dots, do(a_n, do(a_{n-1}, \dots, do(a_1, S_0))))$$

Si ottiene quindi un piano che va dallo stato iniziale a quello finale tramite una sequenza di azioni, quindi l'interprete del GOLOG è un theorem prover general-purpose (in generale per la logica del second'ordine).

Come per il Prolog, i programmi GOLOG sono eseguiti per i loro side effects, ossia per ottenere legami per le variabili quantificate esistenzialmente.

Un implementazione del Golog prototipale e' stata fatta in Prolog (e quindi scompare la logica del second'ordine)

## Concurrent METATEM

E' un linguaggio basato sulla **esecuzione diretta di formule temporali**

E' un linguaggio concorrente, meccanismo di **comunicazione** tra agenti in **broadcast**

Ogni agente è programmato dando una specifica del suo comportamento mediante logica temporale, in questo modo e' possibile eseguire direttamente la specifica

Quindi i **programmi in METATEM sono costituiti da insieme di formule della logica temporale LTL**(equivalente del prolog che invece di usare la logica classica usa quella temporale LTL)

Noi avevamo discusso la LTL come andare solo avanti, qui invece si può analizzare anche il passato

Sintassi:

### **futuro:**

$\bigcirc\varphi$        $\varphi$  must be satisfied in the next state

$\varphi \bigcup \psi$      $\varphi$  will be true until  $\psi$

$\Diamond\varphi$        $\varphi$  must be satisfied at *some* state in the future

$\Box\varphi$        $\varphi$  must be satisfied at *all* states in the future

### **passato:**

$\varphi \mathcal{S} \psi$      $\varphi$  has been true since  $\psi$

$\bullet\varphi$        $\varphi$  was satisfied in the previous state

qualche esempio carino

$\Box\text{important(agents)}$

“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”

$(\neg\text{friends(us)) U}\text{apologize(you))}$

“non siamo amici finché non chiedi scusa”

$\bigcirc\text{apologize(you)}$

“domani (nel prossimo stato), chiederai scusa”

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

$\square(\text{past and present formula} \Rightarrow \text{present or future formula})$

La parte a dx è obbligata ad essere o una **disgiunzione** o una formula **sometime**

- indeterminatezza statica, con l'operatore classico  $\vee$
- indeterminatezza temporale, con l'operatore sometime  $\diamond$  (però, dato  $\diamond\phi$ , l'esecuzione cerca di soddisfare  $\phi$  appena possibile).

L'interprete verifica le regole che hanno gli antecedente soddisfatti, prende i conseguenti e li mette tutti assieme in forma disgiuntiva e ne sceglie uno da eseguire

Se si trova una contraddizione, può essere possibile fare backtrack a una scelta precedente

Le comunicazioni tra agenti sono gestite tramite un **broadcast message-passing**

I messaggi sono predicati di una qualche regola ricevuti dall'esterno o inviati tramite broadcast

Tre categorie di predicati:

- *Environment predicates* ricevuti dall'esterno senza sapere da chi
- *Component predicates* che corrisponde ai messaggi che escono dall'agente
- *Internal predicates*

**rp** (ask1, ask2) [give1, give2]

**O**ask1  $\Rightarrow \diamond\text{give1}$  commits to eventually give to any agent that asks

**O**ask2  $\Rightarrow \diamond\text{give2}$

**start**  $\Rightarrow \square\neg(\text{give1} \wedge \text{give2})$  can give to only one agent at a time

Two consumers:

**rc1** (give1) [ask1]

**start**  $\Rightarrow$  ask1

**O**ask1  $\Rightarrow$  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.

**rp** (ask1, ask2) [give1, give2] vuol dire che l'agente rp può ricevere ask1 e ask2 e fornire give1 e give2

# Lezione 18

## Agenti reattivi (e ibridi)

### Architetture reattive

Un agente ha degli obiettivi, in alcuni casi andrebbe bene seguire un approccio simbolico dell'AI in altre è meglio applicare un paradigma reattivo

#### L'approccio di Brooks

Brooks ha avanzato tre tesi:

- Un comportamento intelligente può essere generato **senza rappresentazione esplicita** del tipo di ciò che è proposto dall'IA simbolica.
- Un comportamento intelligente può essere generato **senza ragionamento esplicito** del tipo di ciò che è proposto dall'IA simbolica
- L'intelligenza è una proprietà emergente di certi sistemi complessi

La logica viene utilizzata come meccanismo per rappresentare la conoscenza, data una logica e un insieme di regole della logica e dei meccanismi di ragionamento, si può quindi utilizzarla per rappresentare conoscenza e fare inferenza. Brooks crede che si possa fare a meno questo.

Per illustrare le sue idee, Brooks fa uso della **subsumption architecture**, che va bene per i robot

Una subsumption architecture è una gerarchia di task che eseguono behaviours

Ogni behaviour è una struttura a regole molto semplici

Ogni behaviour 'competes' con altri per esercitare il controllo sugli agenti

Gli strati più bassi rappresentano tipi di behaviour più primitivi (come evitare ostacoli), e hanno la precedenza su strati più in alto nella gerarchia

I sistemi risultanti sono estremamente semplici in termini della quantità di computazione che fanno

Alcuni robot eseguono task che sarebbero sconvolti se eseguiti da sistemi simbolici di AI

Un **behaviour** è una coppia **(c,a)**, dove c è un insieme di percezioni dette condizioni, e a è una azione. Questo behavior scatta quando c è una percezione

Leggiamo b1 < b2 come "b1 **inibisce** b2", cioè, b1 è più basso nella gerarchia di b2, e quindi ha la priorità su b2.

Guarda l'esempio Steels' Mars Explorer da [slide 9/14](#)

Vantaggi degli agenti reattivi

- Semplicità
- Economia
- Trattabilità computazionale
- Robustezza verso i fallimenti

- Eleganza

### Limiti agenti reattivi

- Agenti senza modello dell'ambiente devono avere sufficienti informazioni disponibili localmente
- Se le decisioni sono basate sull'ambiente locale, come può 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 più ambiente, è 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)

## Architetture ibride

Un 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

Generalmente quello reattivo ha precedenza

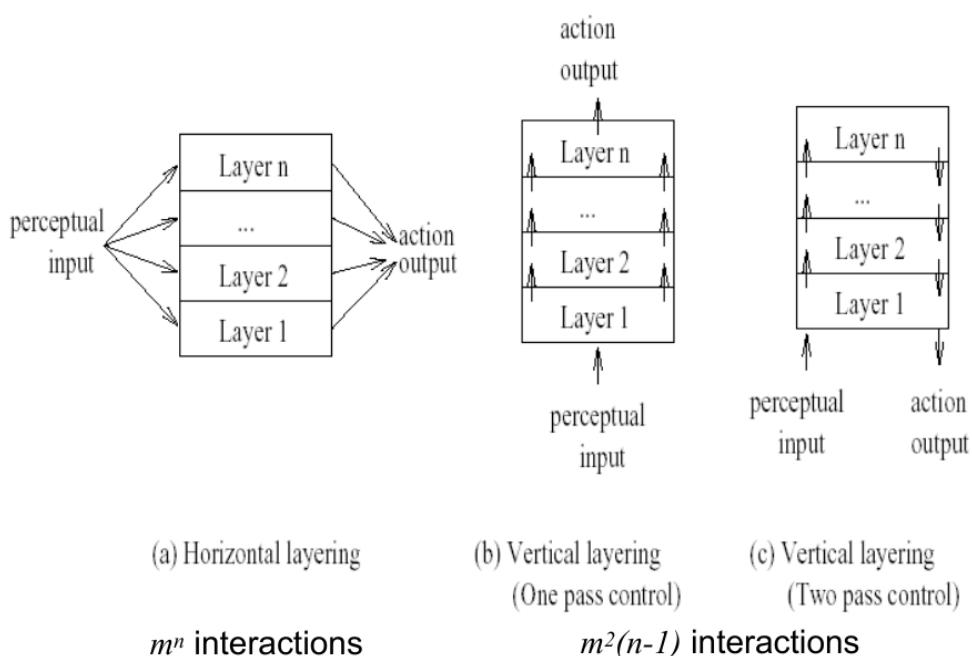
Architetture ibride non sono molto diffuse e utilizzate

TOURINGMACHINES e INTERRAP sono esempi di architetture ibride.

Come mettere insieme questi due sottoinsiemi?

### Horizontal Vs Vertical Layering

$m$  possible actions suggested by each layer,  $n$  layers



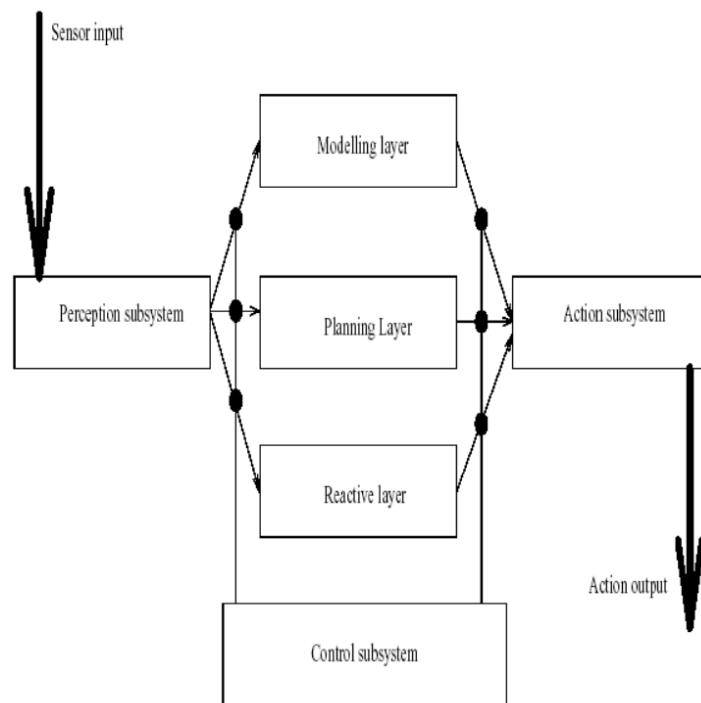
Horizontal layering Tutti gli strati sono direttamente connessi all'input e all'output: ogni strato opera come un agente, producendo suggerimenti su quale azione eseguire.

Vertical layering Input dei sensori e output delle azioni sono gestiti al massimo da uno strato ciascuno

## TOURINGMACHINES

L'architettura di TOURINGMACHINES è orizzontale

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



Il Control system serve per gestire cosa entra e cosa esce da tutti i livelli, visto che nel modello orizzontale tutti i livelli ricevono l'input e tutti producono un output

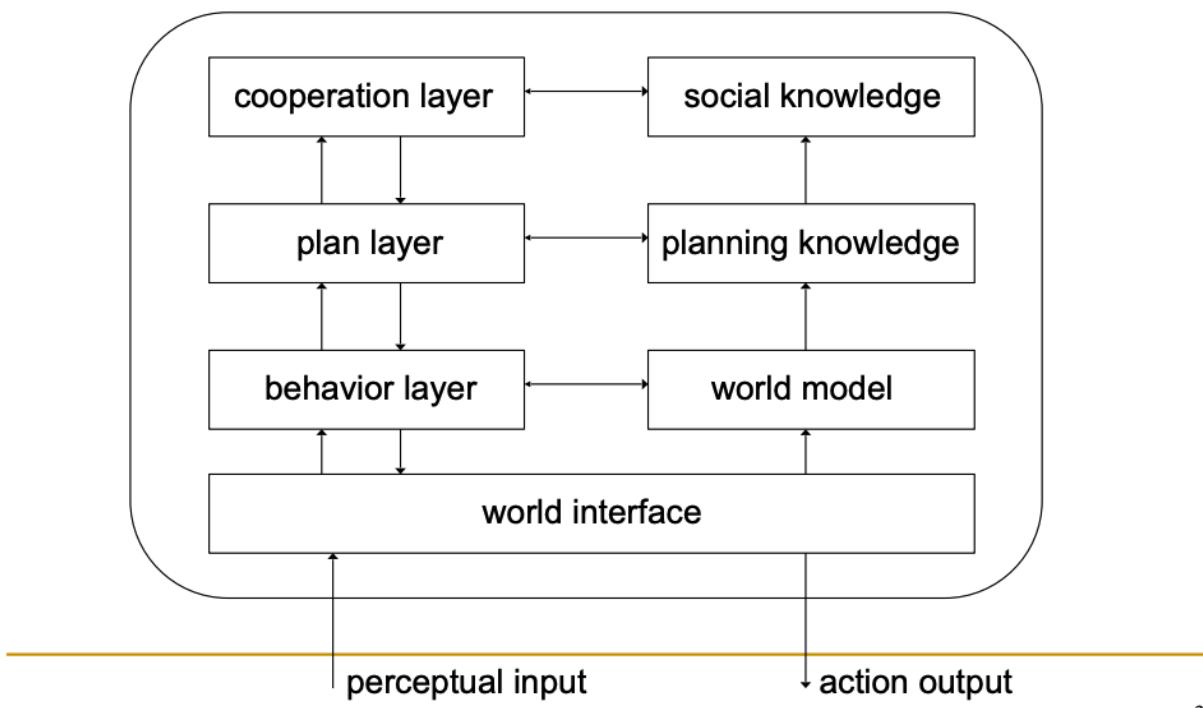
- Lo strato **reactive** è implementato come un insieme di regole situation-action, a la subsumption architecture
- The **planning** layer costruisce piani e sceglie azioni da eseguire, in modo da raggiungere i goal dell'agente
- Il **modelling** layer contiene la rappresentazione simbolica degli ‘stati cognitivi’ di altre entità nell’ambiente dell’agente

I tre layers comunicano tra di loro e sono inseriti in un “control framework”, che usa “control rules”

Il control subsystem sceglie che output scegliere tra i 3 layer

Muller

## ▪ Vertically layered, two-pass architecture



# Lezione 19

## Teoria dei giochi (pt.1)

Siamo in un ambiente **multi agente** e la teoria dei giochi ha come obiettivo trovare un accordo tra gli agenti che partecipano al gioco che sono **agenti indipendente (egoisti)** e non interessati agli obiettivi degli altri.

A differenza dei giochi classici della AI qui le **mosse sono eseguite simultaneamente** e non può consultarsi con altri agenti per trovare una strategia condivisibile  
Per semplicità i giochi consistono in **una sola mossa**.

La teoria dei giochi può essere usata in almeno due modi:

- **Progettazione di agenti**, cosa può fare l'agente e quali sono le regole per partecipare al gioco
- **Progettazione di meccanismi**, trovare dei protocolli per descrivere come l'agente può agire, quindi quali sono le regole usabili dall'agente nello svolgimento del gioco

**Proprietà** principali della teoria dei giochi:

- **Giocatori** che dovranno prendere delle decisioni (noi solo due giocatori)
- **Azioni (strategie)** che i giocatori possono scegliere, nel nostro caso non sono altro che le mosse del gioco. L'insieme di azioni può variare nel tempo. Noi vendendo giochi ad una sola mossa, lo scegliere una **strategia pura** equivale ad eseguire una singola azione. Esistono anche **strategie miste**. La strategia pura è deterministica, per ogni situazione c'è un'azione migliore da compiere, mentre la strategia mista è stocastica e ogni azione possibile ha una probabilità di successo.
- il **risultato** del gioco è lo **strategy profile**, che specifica quale strategia è adottata da ogni giocatore

*Esempio gioco della morra a due-dita*

Due giocatori: P e D, che mostrano simultaneamente 1 o 2 dita.

Sia f la somma delle dita, se f è dispari D vince f dollari, mentre se f è pari P vince f dollari

Ci sono **4 strategy profiles**:

- $s1=(P:\text{uno}, D:\text{uno})$
- $s2 = (P:\text{uno}, D:\text{due})$
- $s3=(P:\text{due}, D:\text{uno})$
- $s4=(P:\text{due}, D:\text{due})$

Possiamo descrivere le **preferenze** per ogni agente con le **utility functions**  
esempio in s1:

$$\begin{aligned} u_P(s1) &= 2 \\ u_D(s1) &= -2 \end{aligned}$$

perché il valore di f è 2, e quindi P guadagna 2 dollari e D li perde

Normalmente si descrivono le preferenze mediante una **matrice di payoff** che dà l'utilità a ogni giocatore per ogni combinazione di azioni di tutti i giocatori.

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

Una **soluzione** a un gioco è uno strategy profile in cui ogni giocatore adotta una **strategia razionale**

(L'argomento più importante in teoria dei giochi è di definire cosa significa razionale quando ogni agente sceglie solo una parte dello strategy profile che determina il risultato)

Consideriamo il **dilemma del prigioniero** ([slide 8](#))

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

Strategia dominante

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

Dato un giocatore A e due strategie sA1 e sA2 di questo giocatore, diciamo:

sA1 **domina strettamente** sA2 se, qualunque cosa faccia l'altro giocatore, sA1 dà al giocatore A un payoff maggiore di quello che sA2 può dare  
 sA1 **domina debolmente** sA2 se, qualunque cosa faccia l'altro giocatore, sA1 dà al giocatore A un payoff maggiore o uguale a quello che sA2 può dare e, in almeno un caso sA1 dà un payoff maggiore di quello che sA2 dà.

sA1 è una **strategia fortemente dominante** se sA1 domina fortemente ogni altra strategia del giocatore A.

Analogamente per debolmente dominante.

Entrambi testimoniando si beccano 5 anni di carcere.. sarebbe stato meglio però se entrambi rifiutassero..

Diciamo che lo strategy profile (testify, testify) non è **Pareto optimal**, ma il profilo (refuse, refuse) lo è.

Pareti optimal vuol dire che abbiamo uno strategy profile che è meglio delle altre

In generale, un risultato è Pareto optimal se non c'è un altro risultato preferibile da tutti i giocatori.

La ragione per cui la soluzione Pareto optimal è improbabile è che **non è un punto di equilibrio**.

Di solito si parla di **equilibrio di Nash** in onore del matematico John Nash

Equilibrio di Nash:

Uno strategy profile  $S=(P1:s1,P2:s2)$  è un equilibrio di Nash se

- assumendo che il primo agente giochi  $s_1$ , il secondo agente non può fare niente di meglio che giocare  $s_2$
- assumendo che il secondo agente giochi  $s_2$ , il primo agente non può fare niente di meglio che giocare  $s_1$

E quindi nessuno dei due giocatori ha intenzione di modificare le proprie azioni.

**Si può dimostrare che un equilibrio con strategie dominanti è un equilibrio di Nash, ma non è detto il viceversa.**

Infatti lo strategy profile (testify, testify) è un equilibrio di Nash!

## Lezione 20

### Teoria dei giochi (pt.2)

Non tutti i giochi hanno strategie dominanti.

Esempio Acme (produttore di HW) e Best (produttore di SW) ([slide 2](#))

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

**Non c'è una strategia dominante.** A non è in grado di scegliere cosa fare indipendentemente dalla scelta di B, e viceversa.

Però ci sono due equilibri di Nash (DVD,DVD) e (CD,CD)

A noi interessa però trovare una strategy profile, se ci sono diverse soluzioni (equilibri di nash) è difficile trovare quale sarà quella giusta..

Una possibilità è che entrambi scelgano la soluzione Pareto-optimal (DVD, DVD)

Sfortunatamente, è facile trovare giochi dove non ci sia questa soluzione - per esempio, potremmo dare ai payoff per (DVD, DVD) l'utilità 5 invece di 9

Per poter trattare un esempio di questo genere gli agenti dovrebbero essere in grado di comunicare e negoziare per raggiungere durante il gioco un accordo che soddisfi entrambi.

Questo era un esempio in cui sono presenti più di un equilibrio di Nash, ci sono esempi però in cui **non è presente nessun equilibrio**, esempio il gioco della morra a due dita. In questo caso si parla di **giochi a somma zero** in quanto la somma dei payoff in ogni cella della matrice è zero.

In questo caso non si possono applicare strategie pure ma bisogna considerare **strategie miste.**

(Lo accenniamo solo)

facciamo riferimento all'esempio dei rigori in cui A tira e B para o a sx o dx.

In pratica ci si aspetta che A tiri un po' a destra e un po' a sinistra con uguale probabilità, e analogamente per B.

Un giocatore ha più strategie da scegliere, se un giocatore ha k scelte possibili s1, s2, ..., sk, **una strategia mista** su queste scelte ha la forma:

- esegui s1 con probabilità p1
- esegui s2 con probabilità p2
- ...
- esegui sk con probabilità pk

con  $p_1 + p_2 + \dots + p_k = 1$

Nash disse prendendosi un Nobel:

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

Il problema è che però **non ha dimostrato come** trovare un equilibrio con strategie miste..

## Problemi importanti

Come risolvere problemi importanti cooperando piuttosto che disertando

Il dilemma di **disertare** (testify) oppure **cooperare** (refuse) è il problema fondamentale delle interazioni multi-agenti.

Sembra implicare che la cooperazione non si presenti in società di agenti self-interested (egoisti)..

*Esempi*

### Riduzione delle armi nucleari

Supponiamo che due paesi i e j abbiano firmato un trattato per eliminare le loro armi nucleari. Ogni paese può o cooperare (eliminare le armi) o disertare (conservare le proprie armi). Ma, se i elimina le sue armi, corre il rischio che j conservi le sue, facendo soffrire ad i il cosiddetto **“sucker’s payoff”**. Viceversa, se i conserva le sue armi, il risultato sarà che i manterrà il proprio arsenale nucleare mentre j non lo farà o, al peggio, ambedue conserveranno le proprie armi.

e altri.. da [slide 9](#)

Una risposta: **ripeti il gioco più di una volta**.

Se sai che incontrerai ancora il tuo avversario, l'incentivo per disertare sembra evaporare.

- La cooperazione è la scelta razionale nel dilemma del prigioniero ripetuto infinite volte. (Hurrah!)

MA ..supponiamo che i giocatori sappiano che il gioco sarà ripetuto n volte

Al giro n - 1, tu hai un incentivo per disertare, guadagnando un extra bit di payoff..

Ma questo rende il giro n – 2 l'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

### Torneo di Axelrod

Supponiamo che tu giochi il dilemma iterato del prigioniero contro un insieme di avversari...

Quale strategia dovresti scegliere, in modo da massimizzare il tuo payoff complessivo?

Axelrod (1984) ha investigato questo problema con un torneo di computer, ognuno programmato per giocare il dilemma iterato del prigioniero con una propria strategia. Ogni gioco consisteva di duecento rounds.

Le strategie usate in questo torneo sono:

- ALLD: – “Always defect” — the hawk strategy;
- TIT-FOR-TAT: On round  $u = 0$ , **cooperate**, 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

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.

Axelrod ha suggerito le seguenti regole per avere successo nel suo torneo:

- 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

## Lezione 21

### Progettazione di meccanismi

Meccanismi (protocolli) che gli agenti possono eseguire per arrivare ad una certa conclusione.

Abbiamo un insieme di agenti (come abbiamo visto nella teoria dei giochi) egoisti, quindi per soddisfare gli obiettivi di questi agenti servono dei meccanismi.

Proprietà desiderate nei protocolli:

- **Successo di 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.
- **Individual 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”.

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.

Fare per se non funziona.. Un esempio è la **tragedy of commons**, una situazioni in cui tutti i contadini portano le loro bestie a brucare gratis sui prati comunitari, con il risultato di distruggerli e giungere così a un'utilità negativa per tutti. Ogni contadino ha agito singolarmente in modo razionale, ragionando che l'uso del prato comune era libero.

## Aste

Assumiamo che ci sia un singolo bene in vendita, che ha un *valore pubblico* per tutti i partecipanti e un *valore privato* che può essere diverso da un partecipante ad un altro.

L'asta classica è l'**asta inglese** il cui prezzo sale finchè il costo corrente è inferiore al valore personale (**strategia dominante**).

L'**asta olandese** invece funziona al contrario.

Un meccanismo alternativo è quello dell'asta in **busta chiusa**

Una variante di questo meccanismo è l'**asta 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*.

Infatti, 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

Vedi da [slide 8](#) in poi un po' di esercizi sull'individuazione di strategie dominanti, equilibri di Nash o soluzioni Pareto optimal

## Lezione 22 - Lab Baldoni

AgentSpeak(L) & Jason

AgentSpeak(L) linguaggio di programmazione di agenti BDI. Jason è la prima significativa implementazione di AgentSpeak(L), realizzata in Java.

Le implementazioni di agenti BDI si occupano di **rappresentare con strutture dati** beliefs, desires e intentions anzichè usare operatori modali della logica.

Il gap tra teoria e pratica rimane eccessivo per via della complessità dei theorem prover e del model checking delle logiche usate per formalizzare i BDI agents

[Rao, 1996] introduce una formalizzazione alternativa degli agenti BDI: **AgentSpeak(L)**

AgentSpeak(L) come estensione della programmazione logica per supportare l'architettura DI.

Agenti BDI:

- Sistemi collocati in un ambiente dinamico, che muta nel tempo
- Sistemi in grado di percepire informazioni provenienti dall'ambiente
- Sistemi in grado di compiere delle azioni (ed apportare modifiche all'ambiente) sulla base delle proprie attitudini mentali: beliefs, desires intentions le principali attitudini

AgentSpeak(L) ha l'obiettivo di preservare queste caratteristiche.

belief: cattura la **componente di informazione**

desire: cattura la **componente motivazionale**

intention: cattura la **componente decisionale**

Piani realizzati dagli utenti anziché da una pianificazione automatica permettono di migliorare l'efficienza.

C'è un gap tra teoria e pratica, AgentSpeak(L) si trova a metà, introduce definizioni per caratterizzare BDI ma offre anche una chiara semantica operazionale che permettono al programmatore di descrivere il programma da eseguire.

AgentSpeak(L) è basato su un linguaggio della logica del prim'ordine semplificato, con l'aggiunta di eventi e azioni.

Il comportamento di un agente (ossia, le sue interazioni con l'ambiente) è dettato dai programmi scritti in AgentSpeak(L).

beliefs, desires intentions non sono rappresentati esplicitamente con formule modali, bensì il progettista attribuisce tali nozioni all'agente usando il linguaggio AgentSpeak(L).

- **Current belief state dell'agente:** modello di sé stesso, dell'ambiente e degli altri agenti
- **Desires:** stati che l'agente vuole raggiungere (sulla base di stimoli interni o esterni); per la precisione, AgentSpeak(L) fa riferimento ai goals, che si possono intendere come desires adottati/scelti
- **Intentions:** adozione di programmi per soddisfare certi stimoli

## AgentSpeak(L)

Linguaggio testuale per scrivere programmi per agenti

Vi è la nozione di:

- base beliefs
- piani
- azioni
- intentions
- eventi
- goals

I piani codificano (in base a come lo descrive un programmatore) il modo in cui deve essere raggiunto un certo obiettivo con lo specifico modo di eseguire le azioni e altre esecuzione che l'agente può fare tramite una specifica coordinata modalità.

### Un piano è il modo in cui descrivere un programma

- Context sensitive, possono essere specificate delle condizioni di contesto che devono essere validate affinché il piano sia applicabile
- Invocati dall'utente specificando un goal che vogliamo essere raggiunto
- Consentono una decomposizione gerarchica dei goals, se un piano permette di raggiungere un certo goal il piano stesso potrebbe esprimere dei sotto goals necessari da raggiungere affinché il goal sia ottenuto
- Sintatticamente simili alle clausole della programmazione logica (anche se con un diverso comportamento)

### Alfabeto del linguaggio formale

- Variabili, caratterizzate dall'iniziale maiuscola
- Costanti
- Simboli funzionali
- Simboli predicativi
- Simboli di azione
- Connettivi
- Quantificatori
- Simboli di punteggiatura

### Connettivi della logica del primo ordine:

- &: congiunzione "and"
- not: negazione
- <-: implicazione

!: achievement goal

??: test goal

:: sequenza

#### Belief atom

Sia  $b$  un simbolo predicativo e siano  $t_1, \dots, t_n$  termini. Allora:

$$b(t_1, \dots, t_n)$$

è un **belief atom** e si scrive anche  $b(\mathbf{t})$ .

Un belief atom e la sua negazione sono detti **belief literal**

Un belief atom ground (cioè senza variabili con occorrenze libere) è detto **base belief**

Siano  $b(\mathbf{t})$  e  $c(\mathbf{s})$  belief atoms; allora:

- $b(\mathbf{t}) \& c(\mathbf{s})$
- **not**  $b(\mathbf{t})$

sono beliefs.

Istanze ground di beliefs si dicono **base beliefs**

Vedi esempio da [slide 18](#).

## Triggering event

Sono eventi percepiti e associati a delle modifiche che possono essere fatte nel nostro agente.

Possono essere di diverso tipo: **Aggiunta/Rimozione di belief/goal**

Questi eventi possono far scaturire l'esecuzione di un piano.

L'aggiunta di un belief/goal è rappresentata dall'operatore “+”

La rimozione di un belief/goal è rappresentata dall'operatore “-”

### Triggering event

Sia  $b(t)$  un belief atom e siano  $!g(t)$  e  $?g(t)$  goals. Allora:

- $+b(t)$  (*belief addition*)
- $-b(t)$  (*belief deletion*)
- $+!g(t)$  (*achievement-goal addition*)
- $-!g(t)$  (*achievement-goal deletion*)
- $+?g(t)$  (*test-goal addition*)
- $-?g(t)$  (*test-goal deletion*)

sono **triggering events**.

## Azioni

Per raggiungere i propri goal bisogna eseguire azioni che possono avere impatto sull'ambiente.

Se move è un simbolo di azione, move(X,Y) è un'azione e rappresenta lo spostamento del robot

Sia a un simbolo di azione e siano t1,...,tn termini, allora **a(t1,...,tn) è un'azione**

## Piano

Un piano specifica il modo in cui un agente potrebbe raggiungere un determinato obiettivo.

**< piano > := < head > <- < body >**  
**< head > := < triggering event > : < context >**

cioè

**piano ==**  
    **trigger event : context(opzionale) <- body**

- **context** specifica quali beliefs dovrebbero essere soddisfatti nel set delle credenze dell'agente quando il piano è attivato
- **body** è una sequenza di goals o azioni e specifica.

**true** rappresenta un componente vuoto (context oppure body)

## Lezione 23

### Semantica operazionale

Describe come una macchina esegue un programma AgentSpeak(L).

A run-time un agente può essere visto come costituito da:

- Un insieme di beliefs B
- Un insieme di piani P, quello che il programmatore ha associato ai possibili modi di agire ad un trigger event
- Un insieme di intentions I, quello che si è impegnato a svolgere
- Un insieme di eventi E, eventi interni o esterni
- Un insieme di azioni A, come può agire l'agente verso il mondo
- Un insieme di funzioni di selezione:  $S_e$ , SO, SI, per scegliere quando ci sono più scelte quale effettuare.

$$S_\epsilon, S_O, S_I$$

Il progettista specifica un agente scrivendo:

- Un insieme di base beliefs
- Un insieme di piani

(guarda da [slide 32](#))

La **Belief Revision Function** ha il compito di revisionare i beliefs interni dell'agente a fronte di un evento (es credere ad una beliefs può implicare rimuovere un'altra o eliminarne un'altra)

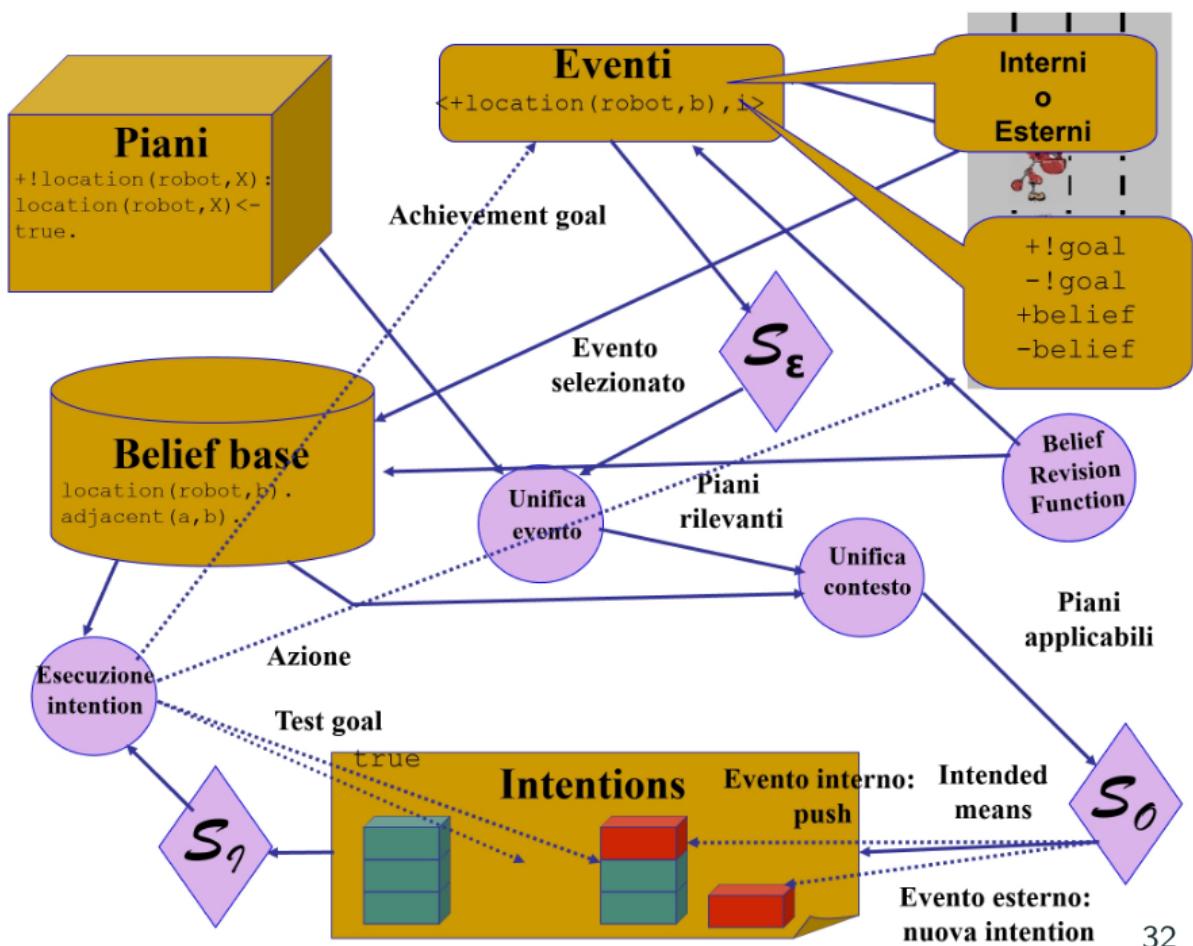
Quando un evento è presente nell'insieme degli eventi E ciclicamente l'interprete di agentSpeak(L) fa:

- $S_\epsilon$  seleziona in  $E$  un evento  $E_0$   
da processare
- $E_0$  viene rimosso da  $E$  e viene  
usato per unificare con i  
*triggering events* dei piani del set  
 $P$
- I piani i cui *triggering events*  
unificano con  $E_0$  sono detti **piani  
rilevanti**; l'unificatore è detto  
**unificatore rilevante**

**Unificare:** definire un istanziazione delle variabili libere di due termini affinché questi diventino equivalenti. (**Wildcard**)

- L'unificatore rilevante è applicato al contesto dei piani rilevanti
- Una **correct answer substitution** è ottenuta dal contesto, ad esempio abbiamo  
 $+t(a,X) : t'(c,d,X) \& t''(Y,e) \leftarrow \text{body}$   
e come evento  $+t(a,b)$   
quindi è un piano rilevante con  $\sigma = \{X/b\}$  (aka sostituzione)  
dobbiamo verificare che nella mia base belief sia vero  $t'(c,d,b) \& t''(Y,e)$   
  
se io ho una belief  $t'(c,d,b)$  e un'altra belief  $t''(g,e)$  e anche  $t''(h,e)$   
avrò come sostituzioni corrette  $\{Y/e\}$  e  $\{Y/h\}$  -> sostituzioni di risposta al contesto  
(aka correct answer substitution). Generalmente viene presa la prima
- Per alcuni piani le **condizioni** dei contesti risultano essere conseguenze logiche del set di base beliefs B: questi piani sono detti **piani applicabili mediante un unificatore applicabile**, risultato dalla composizione della correct answer substitution con l'unificatore rilevante
- Dato un evento  $E_0$ , diversi piani/opzioni risultano applicabili
- La funzione di selezione SO sceglie uno di questi piani/opzioni che chiamiamo Po
- Applicando l'unificatore applicabile a PO si ottiene l'**intended means** in risposta al *triggering event*  
(aka prendo il body e applico l'unificatore applicabile, e quindi sostituisco tutte le occorrenze di X e Y nell'esempio con b ed e)
- Ogni Intention è uno stack di piani parzialmente istanziati o *intention frames*
- **Evento esterno:** l'intendere means è usato per creare una nuova intention, che viene aggiunta al set I
- **Evento interno:** l'intendere means è inserito in cima all'intention esistente che ha "fatto scattare" (triggered) l'evento interno (per raggiungere un goal)
- La funzione di selezione SI sceglie una intention da eseguire

- Quando l'agente esegue una intention, esegue il primo goal o azione del body del top dell'intention:
  - Eseguire un **achievement goal** equivale a generare un evento interno per aggiungere il goal alla corrente intention
  - Eseguire un **test goal** equivale a trovare una sostituzione per il goal che lo renda una conseguenza logica di base beliefs; se viene trovata una sostituzione, il test goal è rimosso dal corpo del top dell'intention
  - Eseguire **un'azione** equivale ad aggiungerla al set di azioni A e rimuoverla dal corpo del top dell'intention



## Intentions

L'insieme delle intentions I è uno stack di piani parzialmente istanziati (potrebbe avere altre variabili oltre a X e Y dell'esempio non istanziate),

Un Intention è denotata con uno stack:

[p1|p2|...|pz]

Una particolare intention è la true intention: [+!true: true<-true]  
per comodità, la denotiamo con T

## Evento

L'insieme E è l'insieme degli eventi.

Ogni evento è una coppia:

$\langle e, i \rangle$

dove:

- e è un triggering event•
- i è una intention

Se i è la true intention T, l'evento è un evento esterno, altrimenti si dice evento interno, non ho riferimento ad una precedente intenzione che ha determinato questo evento

## Jason

Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net

Interpreta il linguaggio AgentSpeak(L) originale

Aggiunge alcune importanti caratteristiche:

- Gestione del fallimento dei piani
- Supporto per lo sviluppo di ambienti da programmare in Java
- Possibilità di eseguire un ambiente multi-agente utilizzando SACI
- Possibilità di personalizzare (in Java) funzioni di selezione, di belief-revision, di azione, di comunicazione fra agenti
- Libreria di "azioni interne" di base
- Possibilità di aggiungere "azioni interne" definite in Java dall'utente
- Aggiunta della negazione forte (invece di not a si scrivere ~a)

### Differenze con AgentSpeak(L):

- La Negazione debole è usata nel contesto dei piani come in AgentSpeak(L), introdotta dal not, es not location(car,Y)
- La Negazione forte è usata per negare un predicato/fatto, es.~location(waste,b)
- gestione fallimenti dei piani (vedi sotto)
- annotazioni (vedi sotto)

I Termini Possono essere:

- Atomi

- Strutture
- Variabili
- Liste (tipo Prolog)
- Numeri (interi o floating point)
- Stringhe (tra "")

Possibilità di **annotare i predicati**: ps(t1, ..., tn)[a1, ..., an]

Particolarmente comodo per annotare i predicati della beliefs base conservando l'origine dell'informazione. Infatti due annotazioni particolari:

- [source(percept)]: l'informazione è stata percepita dall'ambiente
- [source(self)]: l'informazione è stata aggiunta dall'agente stesso durante l'esecuzione di un piano

tutti i predicati nella belied base hanno una annotazione speciale che riporta la sorgente dell'informazione.

Queste annotazioni possono essere usate nel context per essere più fini nello scegliere il piano da eseguire (possono non credere ad un agente in quanto tale)

E possibile associare una **label a ciascun piano**

La label può essere un qualsiasi predicato, quindi anche un predicato con annotazione.

Utili per personalizzare le funzioni di selezione

Esempio:

```
@aPlan[
    chance_of_success(0.3),
    usual_payoff(0.9),
    any_other_property]
+!g(X): c(t)<- a(X)
```

## Gestione fallimento dei piani

- Sono previsti eventi per la gestione del fallimento dei piani
- Tale evento è generato se un'azione fallisce o non vi sono piani applicabili per un evento con aggiunta di un goal +!g
- In tali situazioni viene generato un evento interno per -!g associato alla stessa intention. Se il programmatore ha previsto un piano per -!g e questo risulta applicabile, verrà eseguito. Altrimenti, viene eliminata l'intera intention e segnalato un warning

```
1 +!g : g <- true .
2 +!g : ... <- ... ?g .
3 ...
4 -!g : true <- !g .
```

(esempio di agente blind committed)

## Azioni interne

Possono essere usate sia nel contesto che nel *body* di un piano e vengono introdotte da un punto (es .send) e distinte dalle azioni che l'agente compie sull'ambiente mediante gli attuatori

E' possibile realizzare un sistema multi agente in Jason, nel file di configurazione di un sistema multi-agente: file con estensione.mas2j

Nel file vengono indicati il nome attribuito alla società di agenti, gli agenti AgentSpeak(L) che ne fanno parte, l'ambiente in cui si collocano tali agenti (cioè, la classe Java, eventualmente ridefinita dall'utente, per programmare l'ambiente)ta