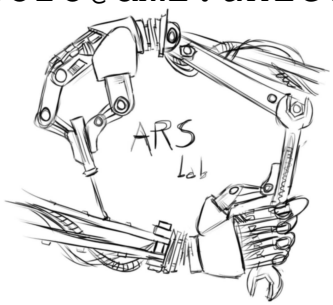
Intelligenza di un robot autonomo
Comportamento, ragionamento, pianificazione

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



Programmazione Sistemi Robotici

Comportamento, ragionamento, pianificazione

- I modelli/metodi/algoritmi studiati finora ci permettono di:
  - Comandare il moto degli attuatori sulla base di un certo modello cinematico
  - Tenere presente i vincoli dell'ambiente e creare i percorsi opportuni
- Tuttavia i vari movimenti devono essere **coordinati** tra loro allo scopo di eseguire compiti più complessi
- L'insieme dei movimenti coordinati può poi far parte di una **strategia** che consente al robot di portare a termine un obiettivo ben preciso
- Tuttavia, dato un certo **obiettivo**, è possibile esistano **differenti strategie**: occorre dunque **pianificare** la strategia più adatta in quel momento specifico

Il ciclo dell' "alto livello"

1. Dato un certo **obiettivo**
2. **Percepisco** l'ambiente per determinarne lo **stato** e verificare che l'obiettivo non sia stato effettivamente raggiunto
3. Determino la **strategia** (insieme di azioni) che potrebbe portare al raggiungimento dell'obiettivo
4. **Eseguo** le relative azioni
5. Ritorno allo step 1

Percezione e Stato dell'Ambiente

Percezione e Stato

- L'ambiente è un sistema dinamico a tutti gli effetti
- Tuttavia, durante la programmazione del comportamento di un robot, non siamo interessati agli aspetti prettamente "controllistici" (funzione di trasferimento, modello nello spazio di stato, etc.) ...
- ... ma sicuramente ad aspetti del tipo "com'è fatto l'ambiente" o "cosa accade in questo preciso momento"
- **Modellare l'ambiente** diventa pertanto uno degli aspetti fondamentale della programmazione del comportamento di un robot
- La modellazione dell'ambiente è inoltre strettamente legata alla tipologia di **sensori** impiegati per lo scopo

Ambiente e Percezione

- Un ambiente fisico è pervaso da:
  - **oggetti inanimati**, incapaci di azioni autome,
  - **oggetti animati** (umani, altri robot), dotati di autonomia
- Essi sono caratterizzati da opportune **proprietà** quali *forma*, *colore*, *posizione*, etc.
- Alcune proprietà sono **immutabili** (*forma*, *colore*), altre possono variare nel tempo (*posizione*)

- **Modellare l'ambiente** implica definire delle **entità informatiche** (classi, oggetti, variabili, etc.) in grado di rappresentare gli oggetti dell'ambiente e le relative proprietà
- Tali informazioni **devono essere percepibili** (determinabili) dai sensori scelti per il robot

Esempio: Il Mondo dei Blocchi

Il Mondo dei Blocchi

- Un ambiente è pervaso da solidi di forma, colore e dimensioni diverse
- Proprietà:
  - **Forma:** *prisma, sfera, cilindro*
  - **Colore:** *giallo, rosso, verde, nero, bianco*
  - **Dimensioni:** *piccolo, grande*
  - **Posizione:** (x, y, z)

Possibile rappresentazione in C/C++

```
typedef enum { PRISM, SPHERE, CYLINDER } t_shape;
typedef enum { YELLOW, RED, GREEN, BLACK, WHITE } t_color;
typedef enum { SMALL, LARGE } t_size;
typedef struct {
  t_shape shape;
  t_color color;
  t_size size;
  float x, y, z;
} t_block;

t_block my_blocks[....];
```

Esempio: Il Mondo dei Blocchi

Il Mondo dei Blocchi

- Se supponiamo che i blocchi possono trovarsi uno sopra l'altro è necessario modellare anche questa caratteristica:
  - attraverso una **funzione booleana** che si basa sulle coordinate dei blocchi, ...

Possibile rappresentazione in C/C++

```
typedef enum { PRISM, SPHERE, CYLINDER } t_shape;
typedef enum { YELLOW, RED, GREEN, BLACK, WHITE } t_color;
typedef enum { SMALL, LARGE } t_size;
typedef struct {
  t_shape shape;
  t_color color;
  t_size size;
  float x, y, z;
} t_block;

t_block my_blocks[....];

bool upon(t_block * block1, t_block * block2)
{
   //....
}
```

Esempio: Il Mondo dei Blocchi

Il Mondo dei Blocchi

- Se supponiamo che i blocchi possono trovarsi uno sopra l'altro è necessario modellare anche questa caratteristica:
  - ... oppure aggiungendo **un'ulteriore proprietà** che rappresenta il link tra due blocchi

Possibile rappresentazione in C/C++

```
typedef enum { PRISM, SPHERE, CYLINDER } t_shape;
typedef enum { YELLOW, RED, GREEN, BLACK, WHITE } t_color;
typedef enum { SMALL, LARGE } t_size;
typedef struct t_block {
  t_shape shape;
  t_color color;
  t_size size;
  float x, y, z;
  struct t_block * upon_block;
} t_block;

t_block my_blocks[....];

bool upon(t_block * block1, t_block * block2)
{
  return block1->upon_block == block2;
}
```

Esempio: Il Mondo dei Blocchi

Il Mondo dei Blocchi

- Se i blocchi vanno **catturati** è necessario modellare anche questo tipo di operazione:
  - utilizzando **due array**, uno per i blocchi non catturati ed un per quelli catturati ....

Possibile rappresentazione in C/C++

```
typedef enum { PRISM, SPHERE, CYLINDER } t_shape;
typedef enum { YELLOW, RED, GREEN, BLACK, WHITE } t_color;
typedef enum { SMALL, LARGE } t_size;
typedef struct t_block {
  t_shape shape;
  t_color color;
  t_size size;
  float x, y, z;
  struct t_block * upon_block;
} t_block;

t_block free_blocks[....], captured_blocks[....];
```

Esempio: Il Mondo dei Blocchi

Il Mondo dei Blocchi

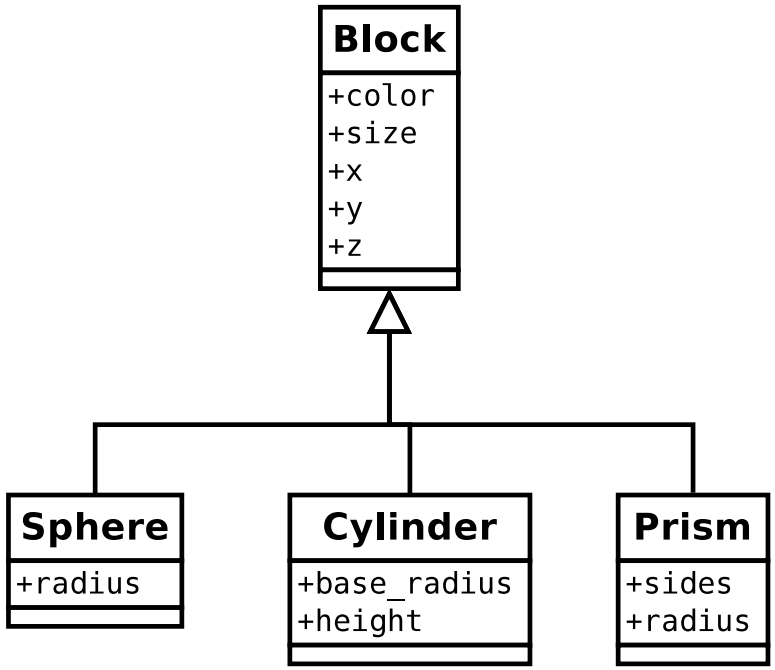
- Se i blocchi vanno **catturati** è necessario modellare anche questo tipo di operazione:
  - ... oppure aggiungendo un **booleano** che indica se il blocco è stato catturato

Possibile rappresentazione in C/C++

```
typedef enum { PRISM, SPHERE, CYLINDER } t_shape;
typedef enum { YELLOW, RED, GREEN, BLACK, WHITE } t_color;
typedef enum { SMALL, LARGE } t_size;
typedef struct t_block {
  t_shape shape;
  t_color color;
  t_size size;
  float x, y, z;
  struct t_block * upon_block;
  bool captured;
} t_block;

t_block my_blocks[....];
```

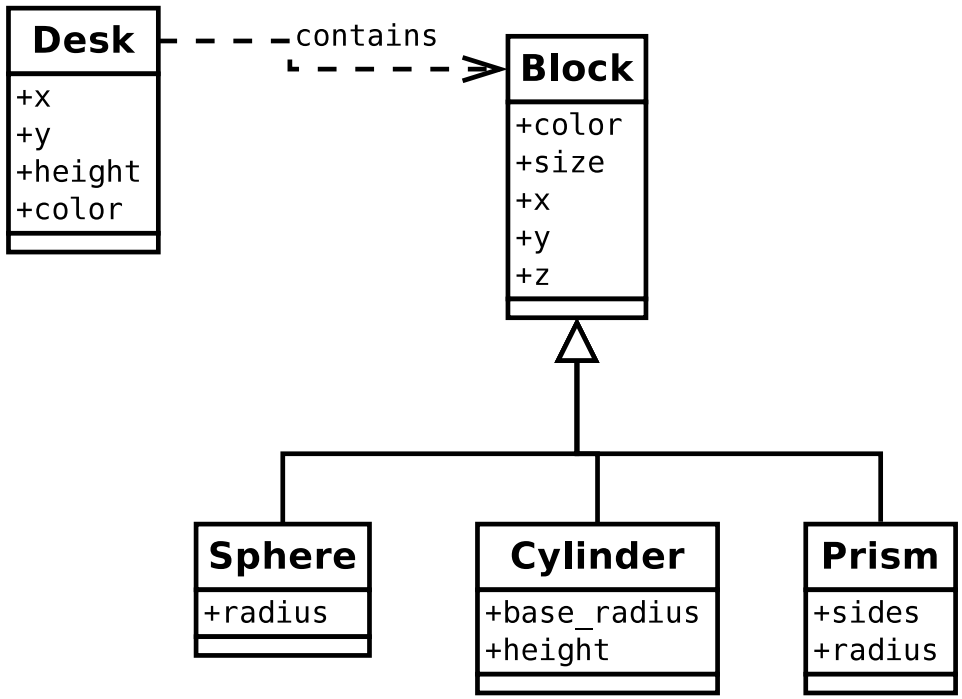
Rappresentazione “Object-Oriented”

- Poichè l’ambiente fisico è pervaso da **oggetti**, una rappresentazione spesso usata è basata sul modello **object-oriented**
- In realtà l’*object-orientation* è nata nel 1965, all’interno dell’AI, proprio per poter rappresentare i “mondi” dei sistemi artificiali



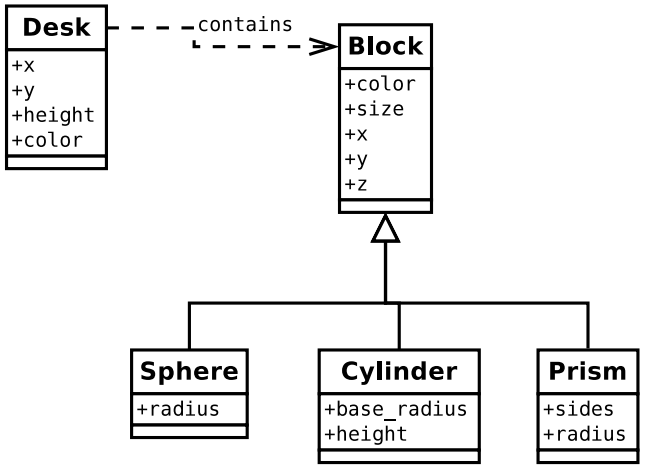
Le Ontologie

- Il modello **object-oriented** permette non solo di rappresentare ciò di cui è composto l'ambiente ma anche le **relazioni** tra le entità
- Il risultato è una **mappa concettuale**, denominata **ontologia**, che rappresenta (nel modo più fedele possibile) l'ambiente di riferimento/il contesto in cui si opera



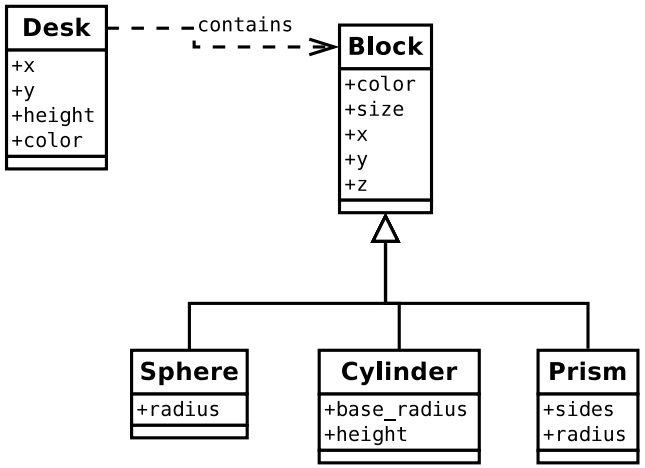
Le “Basi di Conoscenza” (Knowledge Base)

- Qualunque tipo di rappresentazione deve essere “interrogabile” al fine di consentire una forma più o meno complessa di ragionamento
- Ad esempio, se il robot volesse **catturare l’oggetto verde che si trova sul tavolo bianco**, esso dovrà essere in grado di risalire all’oggetto (istanza di `Block`) relativo per poterlo poi raggiungere
- Data un’ontologia (o una rappresentazione analoga), le informazioni devono essere organizzate in un’apposita **base di conoscenza** (**knowledge base**) che possa essere interrogabile facilmente



Le “Basi di Conoscenza” (Knowledge Base)

- Una **Knowledge Base** è pertanto una struttura dati che permette di memorizzare la **conoscenza che il robot possiede sull’ambiente** di riferimento
- E’ costituita da **concetti** che il robot **ritiene veri** (credenze, beliefs)
- Deve consentire la **interrogazione**, **navigazione** e l’**inferenza** di nuova conoscenza



Rappresentazione tramite logica del primo ordine

- Il modello **logico** costituisce un'alternativa per la rappresentazione della conoscenza
- E' basato sulla definizione di **fatti** (credenze, beliefs) rappresentati da **predicati in logica del primo ordine**
- L'inferenza/interrogazione in questi sistemi si basa su **formule logiche**
- Esistono linguaggi di programmazione, come il **Prolog** supportano direttamente questo tipo di modello

Rappresentazione tramite logica del primo ordine

Esempio

- Cilindro, prisma e sfera sono dei blocchi:
  `block(cylinder), block(prism), block(sphere)`
- Il cilindro è rosso, il prisma è bianco:
  `color(cylinder, red), color(prism, white)`
- Il cilindro è sul tavolo nero, il prisma è sul tavolo verde:
  `upon(cylinder, black_desk), upon(prism, green_desk)`
- Qual è l'oggetto sul tavolo verde e qual è il suo colore?:
  `upon(Obj, green_desk) and color(Obj, Col)`
- Quali sono gli oggetti bianchi sul tavolo verde?:
  `upon(Obj, black_desk) and color(Obj, white)`

Percezione, Rappresentazione, Obiettivi, Pianificazione

Percezione, Rappresentazione e Obiettivi

- Un **obiettivo** rappresenta uno **stato dell'ambiente** che si desidera raggiungere
- Un obiettivo può pertanto essere espresso tramite un predicato (o funzione booleana) sulle variabili che rappresentano l'ambiente

Il Mondo dei Blocchi

- **Obiettivo: catturare tutti i blocchi**
- L'obiettivo è raggiunto quando i campi **`captured`** di tutti gli elementi dell'array **`my_blocks`** sono `true`

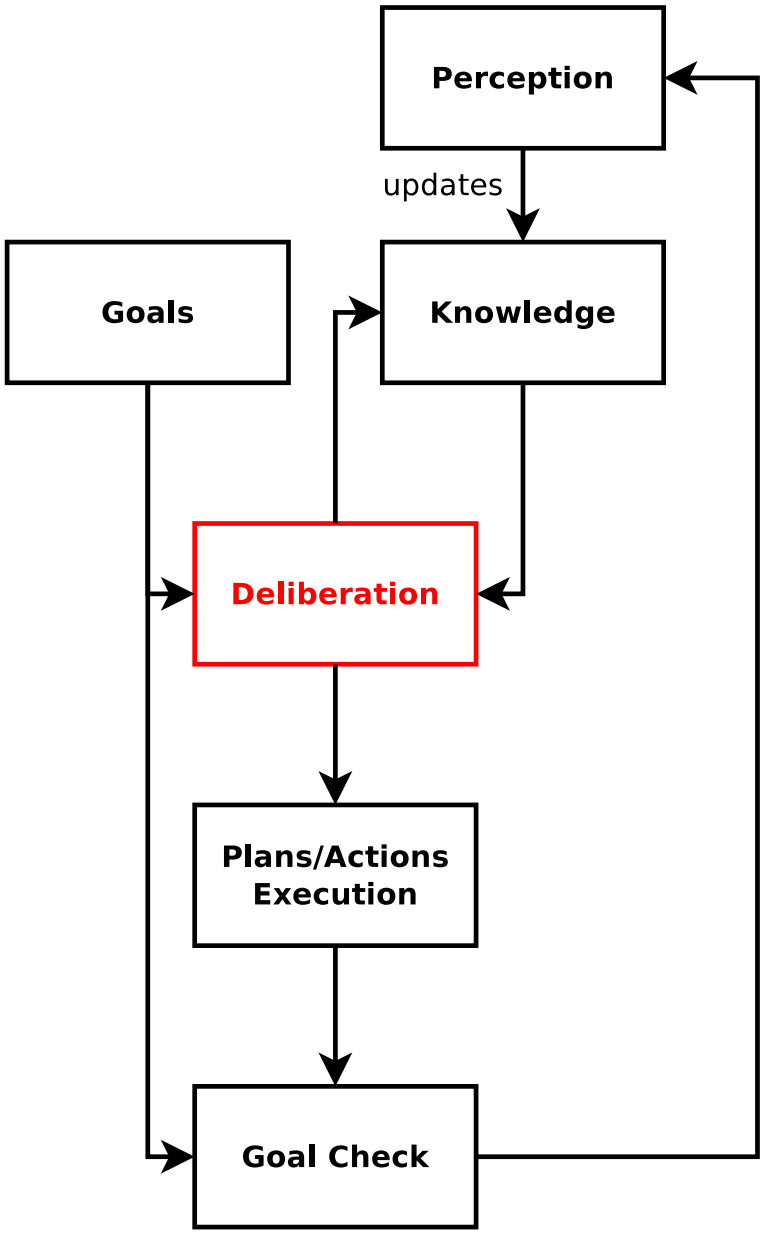
Possibile rappresentazione in C/C++

```
typedef struct t_block {
  ....
  bool captured;
} t_block;

t_block my_blocks[....];

bool goal_done()
{
   for (i = 0; i < NUM_OF_BLOCKS; i++) if (!my_blocks[i].captured) return false;
   return true;
}
```

Obiettivi e sotto-obiettivi

- Raggiungere un obiettivo implica eseguire un certo insieme di azioni
- E tuttavia a volte un obiettivo cela, al suo interno, dei sotto-obiettivi
- Esempio: **catturare tutti i blocchi implica catturarli uno alla volta**
- Obiettivo: **catturare tutti i blocchi**
- Sotto-obiettivi: *catturare la sfera*, *catturare il cilindro*, *catturare il prisma*, etc.
- I sotto-obiettivi non devono necessariamente essere eseguiti secondo una sequenza prefissata
- Tuttavia alcuni sotto-obiettivi potrebbero non essere fattibili (esempio: il cilindro non può essere catturato perchè c'è il prisma sopra)
- Occorre pertanto un processo di pianificazione/scelta dei sotto-obiettivi

Modello del processo di ragionamento e deliberazione



Ragionamento e Deliberazione

- Il processo di deliberazione può essere più o meno complesso a seconda della complessità dell'obiettivo da raggiungere
- Dal punto di vista computazionale può considerarsi un insieme di `if` su vari elementi della knowledge base
- Il risultato dei confronti determina poi il "pezzo di codice" che implementa le azioni da eseguire
- Il modello sottostante è sempre comunque assimilabile a un **automa a stati finiti** (finite-state machine, FSM)
- Spesso è proprio il modello FSM ad essere impiegato in questi casi

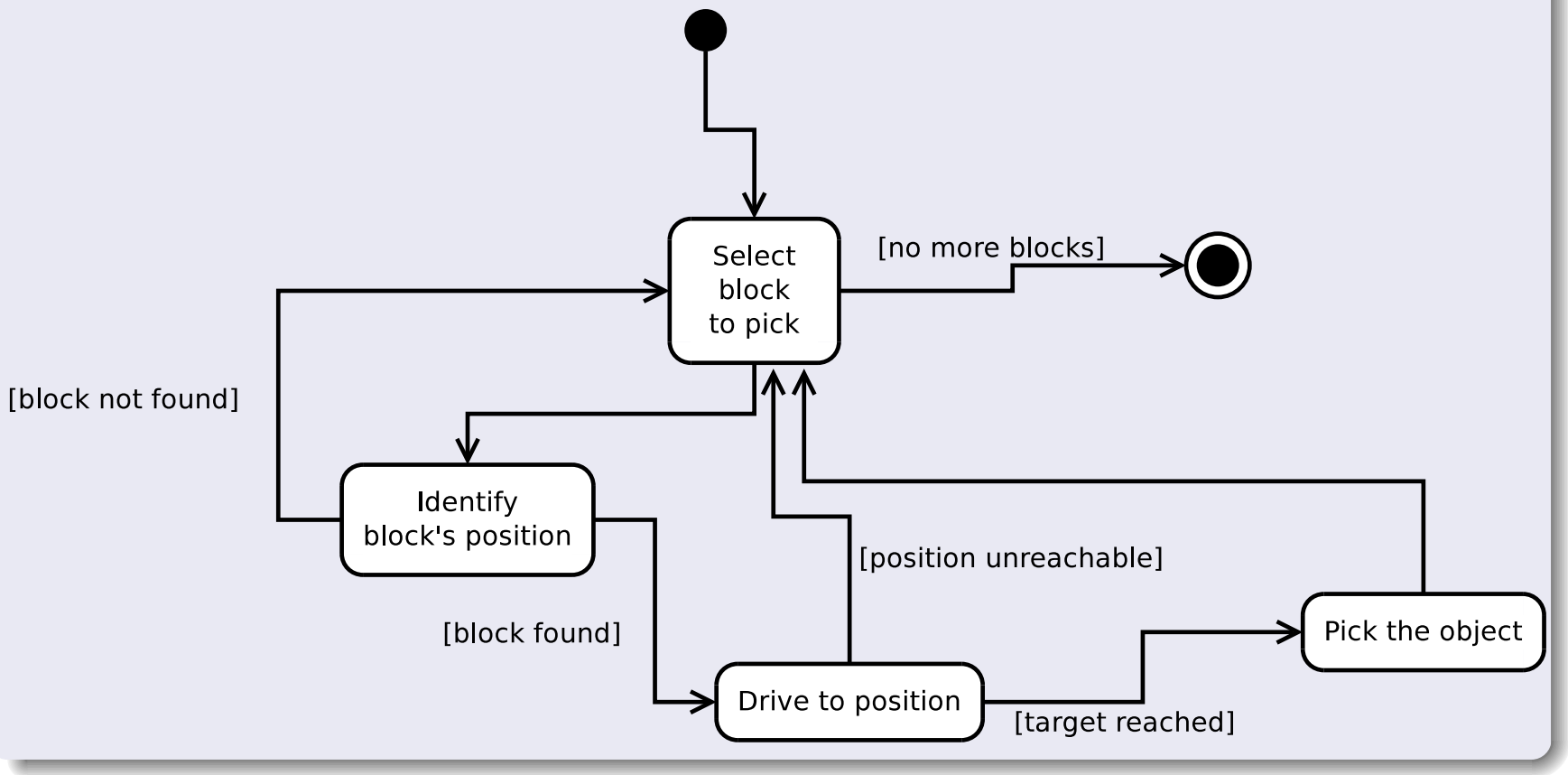
Esempio: il mondo dei blocchi

```
void gather_all_blocks()
{
   for (;;) {
     bool have_prism = kb.have_i_the_object("prism");
     bool have_cylinder = kb.have_i_the_object("cylinder");
     bool have_sphere = kb.have_i_the_object("sphere");
     if (have_prism && have_cylinder && have_sphere) {
         return; // goal accomplished
     }

     // here we have a FIXED selection of sub-goals

     if (!have_prism) gather_object("prism");
     else if (!have_cylinder) gather_object("cylinder");
     else if (!have_sphere) gather_object("sphere");
   }
}
```

Esempio: il mondo dei blocchi

```
void gather_object(const char *object_type)
{
   float x,y,z;
   if (!kb.get_object_position(object_type, &x, &y, &z) {
      // uh? I don't know object's position, let us try to find it
      if (!camera_sensor.detect(object_tpe, &x, &y, &z))
          return; // cannot detect object, the sub-goal fails
      kb.update_object_position(object_type, x, y, z);
   }

   robot.drive_to(x, y, z);
   while(true) {
      if (robot.position_reached(x,y,z)) break; // we got the position
      if (robot.motion_blocked()) return;       // goal failed
      if (timeout()) return;                    // goal failed
   }

   robot.pick_the_object();
   while(true) {
      if (robot.object_picked()) break; // we got the object
      if (timeout()) return;            // goal failed
   }

   kb.mark_object_picked(object_type);
}
```

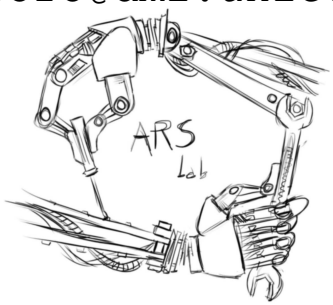
Il mondo dei blocchi: FSM equivalente



Intelligenza di un robot autonomo
Comportamento, ragionamento, pianificazione

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



Programmazione Sistemi Robotici