Cognitive Robotics

Maurice Pagnucco
School of Computer Sc. & Eng.
University of New South Wales
NSW 2052, AUSTRALIA

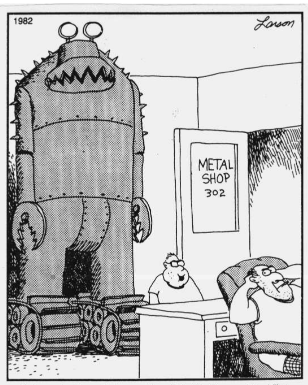
morri@cse.unsw.edu.au
http://www.cse.unsw.edu.au/~morri/

Cognitive Robotics

- Study of the knowledge representation and reasoning problems faced by an autonomous agent in a dynamic and incompletely known world
- If possible, do not want to develop robot controllers that only work in a restricted class of application domains
- Problems that need to be addressed:

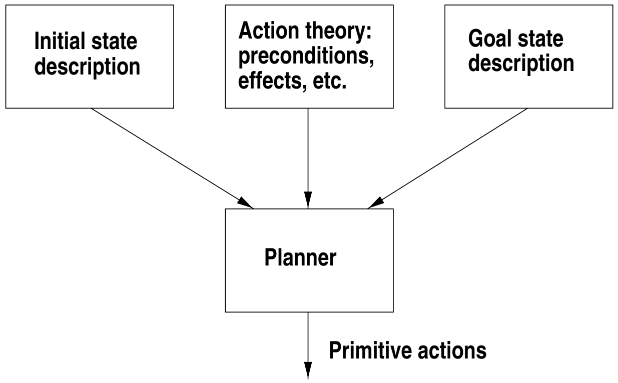
  Specification of user actions (their preconditions and effects), exogenous actions, sensing, incomplete/partial information, incorrect/noisy information, knowledge/belief of agents and introspection, execution monitoring, discrete/continuous actions, probabilistic occurrences of actions and effects, complex actions, concurrency, continuous actions, hypothetical/counterfactual reasoning, agent epistemic attitudes, reactive behaviour, real-time behaviour, revising beliefs, planning . . .

Cognitive Robotics



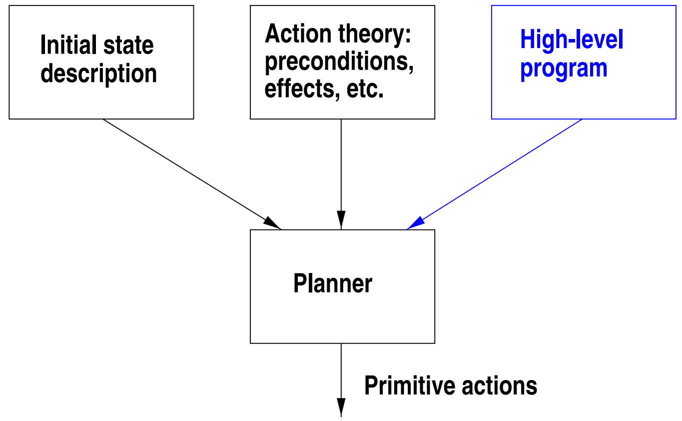
“My project’s ready for grading, Mr. Big Nose. . . . Hey! I’m talkin’ to *you*, squidbrain!”

Classical Planning



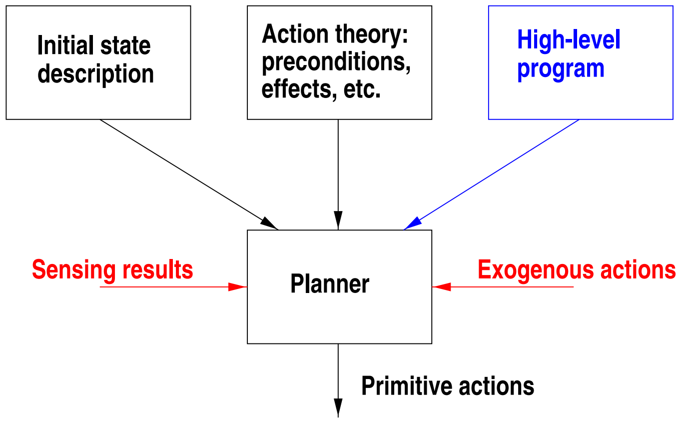
Idea: determine sequence of (primitive) actions that can be legally performed starting at the initial state and ending at the goal state

High-level Program Approach



Idea: Find legal execution of program starting in the initial state

High-level Program Approach — Mark II



Situation Calculus

- The situation calculus is a way of describing change in first-order predicate calculus
- Situation/State — a snapshot of the world at a particular point in time
- Aspects we need to consider:
  - The state of the world
  - Actions that change state of the world and when they may occur/what changes they effect
  - Constraints on legal scenarios
- Reasoning problems that arise:
  - Frame, ramification and qualification problems

Do formula

Want to be able to use complex actions composed of primitive actions and "inherit" solution to the frame problem

For complex action A can define a situation calculus formula $Do(A, s, s')$ — action A when started in situation s may legally terminate in situation s'

- Primitive actions
  $Do(A, s, s') = Poss(A, s) \wedge s' = do(A, s)$
- Sequence
  $Do([A;B], s, s') = \exists s''.Do(A, s, s'') \wedge Do(B, s'', s')$
- Conditionals $Do([\textbf{if } \phi \textbf{ then } A \textbf{ else } B], s, s') =$
  $\phi(s) \wedge Do(A, s, s') \vee \neg\phi(s) \wedge Do(B, s, s')$
- Nondeterministic branch $Do([A|B], s, s') = Do(A, s, s') \vee Do(B, s, s')$
- Nondeterministic choice $Do([\pi x.A], s, s') = \exists x.Do(A, s, s')$

GOLOG

- GOLOG (Al**go**l in **log**ic) is a programming language that generalises conventional imperative programming languages
  - usual imperative constructs + concurrency nondeterminism
  - other features bottom out not on operations on internal states (assignment statements, pointer updates) but on primitive actions in the world (e.g. pickup a block)
  - what the primitive actions do is user-specified by precondition and successor state axioms
  - conditionals (tests) — fluents affected by action occurrence
  - programs may contain non-deterministic constructs — reasoned choice
  - legal execution of program requires reasoning about action

GOLOG

- What does it mean to “execute” a GOLOG program?
  - find a sequence of primitive actions such that performing them starting in some initial situation s would lead to a situation s' where the formula $Do(A,s,s')$ holds
  - give the sequence of actions to a robot or simulator for actual execution in the world
  - **Note:** to find such a sequence, it will be necessary to reason about the primitive actions
  - A; **if** $Holding(x)$ **then** B **else** C to decide between B and C we need to determine if the fluent Holding would be true after doing A

GOLOG Example

- Primitive actions: $pickup(x)$, $putonfloor(x)$, $putontable(x)$
- Fluents: $Holding(x,s)$, $OnTable(x,s)$, $OnFloor(x,s)$
- Action preconditions:
  $Poss(pickup(x),s) \equiv \forall z.\neg Holding(z,s)$
  $Poss(putonfloor(x),s) \equiv Holding(x,s)$
  $Poss(putontable(x),s) \equiv Holding(x,s)$
- Successor state axioms:
  $Holding(x,do(a,s)) \equiv a = pickup(x) \vee$
  $\quad Holding(x,s) \wedge a \neq putontable(x) \wedge a \neq putonfloor(x).$
  $OnTable(x,do(a,s)) \equiv a = putontable(x) \vee$
  $\quad OnTable(x,s) \wedge a \neq pickup(x).$
  $OnFloor(x,do(a,s)) \equiv a = putonfloor(x)$
  $\quad OnFloor(x,s) \wedge a \neq pickup(x).$

GOLOG Example

- Initial situation:
  $\forall x.\neg Holding(x,S_0)$
  $OnTable(x,S_0) \equiv x = A \vee x = B$
- Complex actions:
  proc ClearTable: **while** $\exists b.OnTable(b)$ **do**
  $\quad \pi b[OnTable(b)?; RemoveBlock(b)]$

  proc $RemoveBlock(x) : pickup(x); putonfloor(x)$

GOLOG

- To find a sequence of actions constituting a legal execution of a GOLOG program, we can use Resolution with answer extraction.
- For the above example, we have

$$KB \models \exists s. Do(ClearTable, S_0, s)$$

- The result of this evaluation yields

  $s = do(putonfloor(B), do(pickup(B),$
  $\quad do(putonfloor(A), do(pickup(A), S_0))))$

  and so a correct sequence is

  $pickup(A), putonfloor(A), pickup(B), putonfloor(B)$

GOLOG

- When what is known about the actions and initial state can be expressed as Horn clauses, this evaluation can be done directly in Prolog
- The GOLOG interpreter in Prolog has clauses like

```
do(A,S1,do(A,S1)) :-
  prim_action(A), poss(A,S1).

do(seq(A,B),S1,S2) :-
  do(A,S1,S3), do(B,S3,S2).
```

  This provides a convenient way of controlling a robot at a high level.

GOLOG—Programming Constructs

α	primitive action
$\phi?$	condition (wait)
$(\delta_1;\delta_2)$	sequence
if ϕ **then** δ_1 **else** δ_2 **endIf**	conditional
while ϕ **do** δ **endWhile**	loop
proc $\beta(\bar{x})\delta$ **endProc**	procedure definition
$\beta(\bar{t})$	procedure call
$(\delta_1 \mid \delta_2)$	nondeterministic choice of actions
$(\pi\, \bar{x})[\delta]$	nondeterministic choice of arguments
δ^*	nondeterministic iteration
$(\delta_1 \| \delta_2)$	concurrent execution
$(\delta_1 \rangle\rangle \delta_2)$	prioritised concurrency
$\delta^{\|}$	concurrent iteration
$\langle x : \phi \rightarrow \delta \rangle$	interrupt
search(δ)	search

Simple GOLOG Program

[Levesque *et al.* 1997]

proc *Control*
[**while** ($\exists n$) on(n) **do** *Serve_a_floor* **endWhile**]; *Park* **endProc**

proc *Serve(n)*
Go_floor(n); *Turnoff(n)*; *open*; *close* **endProc**

proc *Go_floor(n)*
(current_floor = n)? | *up(n)* | *down(n)* **endProc**

proc *Serve_a_floor*
(π n) [Next_floor(n)?; *Serve(n)*] **endProc**

proc *Park*
if current_floor = 0 **then** *open* **else** *down(0)*; *open* **endIf endProc**

Delivery Task

- GOLOG program

  $\langle$motion = lost $\rightarrow$ *(recover); start_to_next_station*$\rangle$ $\rangle\rangle$
  $\langle$motion = moving $\rightarrow$ *wait*$\rangle$ $\rangle\rangle$
  $\langle$StopRequested(location) $\rightarrow$ *signal_arrival; wait*$\rangle$ $\rangle\rangle$
  $\langle$n: NextLocationToServe(n) $\rightarrow$
  if location $<$ n **then** *Head_to_next_station(1)*
  else *Head_to_next_station*$(-1)$$\rangle$ $\rangle\rangle$
  $\langle$location $> 1 \rightarrow$ *Head_to_next_station*$(-1)$$\rangle$ $\rangle\rangle$
  $\langle$**true** $\rightarrow$ *wait*$\rangle$

Summary

- Cognitive robotics takes largely theoretical developments in knowledge representation and reasoning about action and applies it to high-level (robot) control
- Complementary to work in traditional robotics
- Other artificial intelligence approaches: fluent calculus, Belief-Desire-Intention (BDI) models, event calculus, stable models ($\mathcal{A}$-languages), agent languages
- System for experimenting with LEGO® MINDSTORMS™ robots
  `http://www.cs.toronto.edu/~cogrobo/Legolog/`

Navigation Problem

Let's explore a formalisation of a simple delivery task. We consider a world consisting of way stations connected by pathways that can run north, east, south and west; at any location, there may not be pathways leading in all of the directions. You wish to navigate a robot around this world. The state of the robot is governed by two fluents: $Location(x, s)$—the robot is located at way station x in situation s; and, $Direction(x, s)$—the robot is facing direction x (North, East, South, West) in situation s. The robot is capable of performing the following actions: $forward$ which takes it to the next station in the direction it is facing; and , $turn\_clockwise$ changes its direction 90° by turning in a clockwise direction. You may also assume the following relation: $Connected(x, y, direction)$—the robot can go from location x to location y by moving forward if it is facing $direction$; and, $Clockwise(x, y)$—when facing direction x a clockwise turn by 90° will make it face direction y. There are also constant symbols for each of the way stations but for our purposes here it is sufficient to distinguish only two of them: $Home$ and $Depot$. In the initial situation the robot is located at $Home$ facing $North$. You are to consider navigating the robot so that it ends up being located at $Depot$.

Navigation Problem

1. Write the action precondition axioms for the actions forward and turn_clockwise.

Navigation Problem

$$Poss(forward, s) \equiv Location(x, s) \wedge Direction(y, s) \supset \exists z.Connected(x, z, y)$$

Alternatively

$$Poss(forward, s) \equiv \exists x, y, z.Location(x, s) \wedge Direction(y, s) \wedge Connected(x, z, y)$$

$$Poss(turn\_clockwise, s) \equiv TRUE$$

Navigation Problem

2. Write the effect axioms for the actions forward and turn_clockwise.

Navigation Problem

$$Location(x, s) \wedge Direction(y, s) \wedge Connected(x, z, y) \supset Location(z, do(forward, s))$$

$$Location(x, s) \supset \neg Location(x, do(forward, s))$$

$$Direction(x, s) \supset \neg Direction(x, do(turn\_clockwise, s))$$

$$Direction(x, s) \wedge \neg Clockwise(y, x) \supset \neg Direction(y, do(turn\_clockwise, s))$$

Navigation Problem

3. Show how a successor state axiom for Location would be derived from these effect axioms. Is the successor state axiom logically implied by the effect axiom? Explain.

Navigation Problem

$$\begin{aligned}&Location(x,do(a,s)) \equiv\\ &\quad a = forward \wedge Location(z,s) \wedge Direction(y,s) \wedge Connected(z,x,y)\\ &\quad Location(x,s) \wedge a \neq forward\end{aligned}$$

- The successor state axiom implies the effect axioms

Navigation Problem

4. Show how a frame axiom is implied by this successor state axiom.

Navigation Problem

$$Location(x,s) \supset Location(x,do(turn\_clockwise,s))$$

Others are possible.

Navigation Problem

5. Write a sentence of the situation calculus whose only situation term is S_0, describing the initial situation. Write a sentence in the situation calculus of the form $\exists s.\alpha$ which asserts the existence of a goal situation.

Navigation Problem

$Location(x, S_0) \wedge Direction(y, S_0) \equiv x = Home \wedge y = North$

$\exists s. Location(Depot, s)$

Navigation Problem

6. Explain how you could use Resolution to automatically solve this delivery task for any initial situation: how would you generate the clauses and, assuming the process stops, how would you extract the necessary moves? (Do not attempt to write down a derivation!) Explain why you need to use the successor state axioms, and not just the effect axioms.

Navigation Problem

- Replace goal by
  $$\exists s.[Location(Depot, s) \wedge Legal(s) \wedge \neg A(s)]$$
  and use this as a query where A is an answer predicate. The knowledge base contains:
  - precondition axioms (*Poss*)
  - successor state axioms
  - axioms for $Legal(s)$: $Legal(S_0)$; $Legal(do(a, s)) \equiv Poss(a, s) \wedge Legal(s)$
  - initial state axioms
- We derive a clause containing just the answer predicate. If there is an answer it will be of the form
  $$do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, S_0))\ldots))$$
  To solve the navigation task (i.e., to get from the initial location to the goal location) execute the actions $a_1, \ldots, a_n$ in that sequence. We need to use the successor state axioms so that we can reason about the values of all fluents after any action is performed.

Navigation Problem

7. Suppose we are interested in solving the task in Prolog. Write but do not run the Horn clauses (using negation as failure) which define *Poss* and *Location*. Relate these clauses to the above axioms.

Navigation Problem

```
% From precondition axioms
poss(forward, S) :-
    location(X, S),
    direction(Y, S),
    connected(X, _, Y).

poss(turn_clockwise, _).

% From successor state axioms
location(X, do(forward, S)) :-
    location(Z, S),
    direction(Y, S),
    connected(Z, X, Y).

location(X, do(turn_clockwise, S)) :-
    location(X,S).
```


Navigation Problem

8. Explain the difficulty in using these clauses to solve the task in Prolog, given its depth-first search strategy. Would a breadth-first strategy do the job, assuming we were willing to wait for an answer?

Navigation Problem

We could continually apply actions creating longer and longer histories that get us nowhere with depth-first search. In particular, there are two common scenarios. In the above axiomatisation, the action *turn_clockwise* is always possible. If we were to always apply it first (say, by reordering the last two clauses above), we would end up on an infinite branch of the Prolog search tree. The other common possibility is where there are loops in the map and the Prolog code gets stuck going around a loop.

Breadth-first search would work (provided there are infinitely many actions or action instances to consider).

Navigation Problem

9. Sketch briefly how the clauses might be modified to work with the depth-first search of Prolog, again assuming we were willing to wait as long as necessary for an answer.

Navigation Problem

Many are possible. Some examples are modifying the ordering of the clauses, using an iterative deepening search strategy, modify the clauses to keep track of when we've been to a location/direction before and not to search any further in this branch.

Navigation Problem

10. Without any additional guidance, a very large amount of search is required to solve this task. There are, however, heuristics that can be used to reduce the amount of search. One possible heuristic is the following: when deciding which location to move to next, choose the one that is closest to the goal location. Assume that you have a predicate $Closer(x,\ y,\ z)$ which is true whenever location x is closer to location z than location y is to z (i.e., the distance between x and z is shorter than that between y and z). Explain how the complex actions of GOLOG could be used to define a more restricted search problem which incorporates heuristics like this. Sketch briefly what the GOLOG program should look like. What assumptions are needed to guarantee that this program finds a path to the goal?

Navigation Problem

proc *main*
  while $(\text{location} \neq \text{Depot})$ **do**
    $\pi x.[\exists y.(\text{Connected}(\text{location}, x, y) \wedge$
      $\neg \exists z, w.\text{Connected}(\text{location}, z, w) \wedge$
      $x \neq z \wedge \text{Closer}(x, z, \text{Depot}))?; Moveto(x)]$

proc $Moveto(x)$
  while $(\text{Connected}(\text{location}, x, y) \wedge \text{direction} \neq y)$ **do**
    $turn\_clockwise$;
  $forward$

Navigation Problem—Bonus Question

11. We have treated location and direction as general relations, not as single-valued functions. However, prove that it follows from your formalisation of the domain that if the robot starts at a unique location facing a unique direction in the initial situation that, at any given later situation, the robot is at a unique location facing a unique direction. [Hint: use induction on the number of actions performed.]