G53DIA: Designing Intelligent Agents

Lecture 5: Deliberative Architectures

Brian Logan
School of Computer Science
bsl@cs.nott.ac.uk

Outline of this lecture

- deliberative architectures
- the role of representation
- advantages of deliberation
- examples
  - travelling salesman problem
  - Shakey the robot

Agent architecture

- the *architecture* defines a (real or virtual) machine which runs the agent program
- defines the *atomic operations* of the agent program
- determines which operations happen *automatically*, without the agent program having to do anything
- the atomic operations and automatic operations determine whether an architecture is *reactive* or *deliberative*

Limitations of reactive architectures

- there are many things agents with reactive architectures can't do:
  - they can't represent or reason about hypothetical objects and times;
  - they don't do well in domains where plausible actions can't be ignored or undone if they prove to be unwise;
  - it is difficult for purely reactive agents to organise their own activities over time or to coordinate their behaviour with that of other agents in a non-trivial way
- it's difficult to get *intelligent* behaviour from a purely reactive agent

Deliberation

- deliberation is the *explicit* consideration of *alternative courses of action*
- deliberation involves *generating alternatives* and *choosing an alternative*
- an agent can deliberate about:
  - **means:** how to achieve a goal (this lecture)
  - **ends:** whether to achieve (intend) a goal (next lecture)

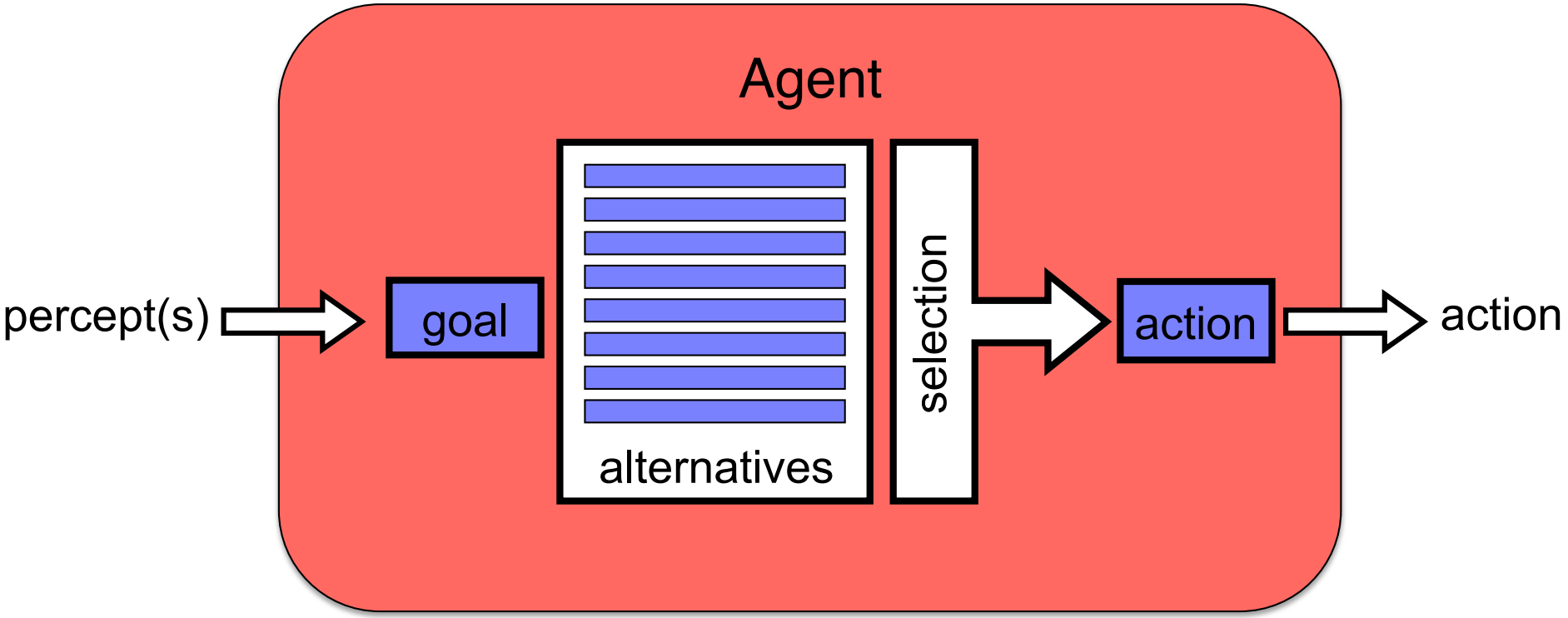
Examples: deliberating about means

- problem solving (search)
- planning
- scheduling
- theorem proving
- constraint satisfaction
- etc

Example: search problems

- a search problem consists of:
  - a set of *states*: complete descriptions of the world for the purposes of problem-solving
  - a set of *operators*: actions that transform one state into another state
  - the *state space* is the set of all states reachable from the initial state by any sequence of actions
- a *path* in the state space is any sequence of actions leading from one state to another
- a *solution* is a path from an initial state to a goal state
- a *path cost function*, *g(n),* can be used to assign a quality measure to a path

Deliberative architecture



Deliberative architectures (means)

- in a deliberative architecture, percepts (or communication) give rise to *goals*—representations of a state to be achieved
- the agent *deliberates* about how to achieve the goal
  - deliberation involves (usually systematic) *exploration of alternative courses* of action
  - a *deliberative architecture* typically includes *automatic generation and comparison of alternatives*
- result of deliberation is a *representation* of of the action(s) to be performed

The role of representations

- deliberation involves the manipulation of a *model of the world* and *possible courses of action*, rather than the world itself

- requires the ability to represent actions and derive the consequences of actions *without actually performing them,* e.g.:

  – by remembering their effects in previous, similar situations

  – by reference to a causal model of the world

Counterfactual representations

- to represent desired states and the consequences of actions:
  - some states of the agent must be *counterfactual* in the sense of referring to hypothetical future states (goals) or as yet unexecuted actions (plans)
  - some of the basic operations of the architecture should generate such counterfactual states
  - such states must be influential in the choice of actions
- to represent hypothetical situations, a deliberative agent requires representations with *compositional semantics*

Advantages of deliberation

- useful when the penalty for incorrect actions is high, e.g., when the environment is hazardous
- allows us to code a *general procedure* for finding a solution to a *class of problems*
  - may be better than reactive systems at coping with novel problems
  - we may be able to get a correct or even an optimal answer, e.g., decision theoretic approaches

Example: Traveling Salesman Problem

- given a set of cities and routes between them, find the shortest tour that visits each city exactly once
- often formulated of finding the shortest Hamiltonian cycle in a completely connected undirected graph
- however there are many variants, e.g., graph may be incomplete or directed, non-metric distances, etc.
- finding an optimal solution is NP-hard

TSP algorithms

- many TSP algorithms use *iterative refinement*, i.e., they require the representation of at least two *alternative* tours:
  - the *current best tour*
  - the *candidate tour* (often a modification of the current best tour)
- iterative algorithms typically stop when no modification of the current best tour has lower cost or when there has been no improvement for *n* iterations
- agent then executes the steps in the tour

More complex deliberation

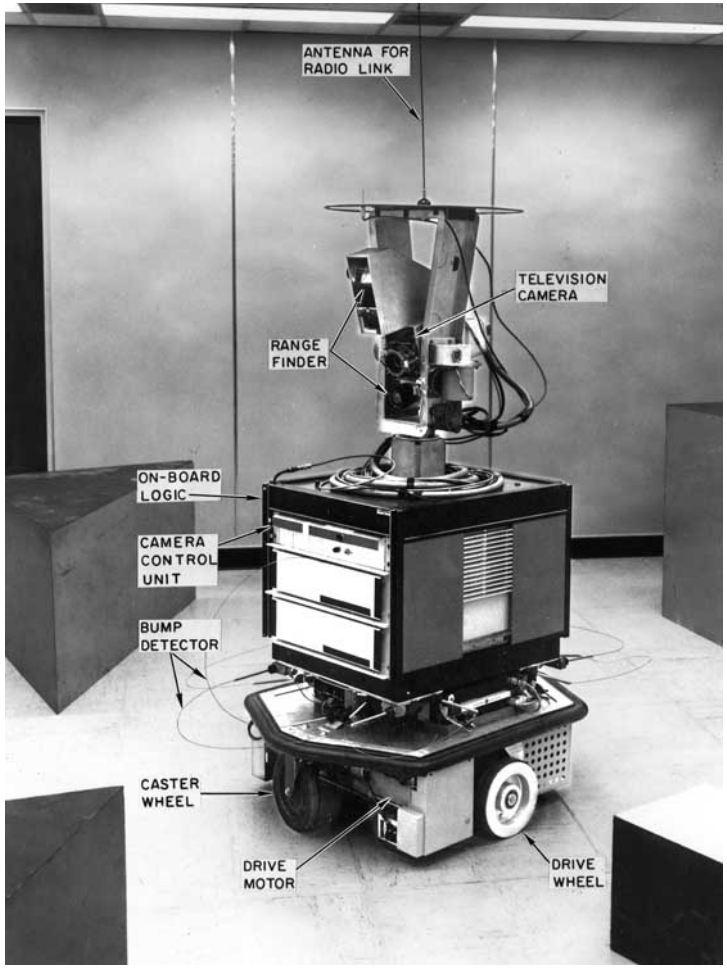
- iterative improvement TSP algorithms implement a very simple form of deliberation

- they are specific to a particular kind of problem, i.e., TSP problems (though many interesting real world problems can be formulated as a TSP problems)

- viewed as search problems, their representational requirements and the number of operators are minimal

Shakey the robot (1966–1972)

- Shakey was the first mobile robot to reason about its actions
- multiple sensors (TV camera, a triangulating range finder, and bump sensors)
- connected to DEC PDP-10 and PDP-15 computers via radio and video links
- programs for perception, world-modeling, and acting (simple motion, turning, and route planning)



Shakey's impelmentation

- programmed in Fortran and Lisp
- problem solving (planning) was implemented using STRIPS (as of 1969)
- ran on a "large" PDP-10 with 192K 36-bit words of memory
- Shakey's programs occupied "over 300,000 36-bit words" (i.e., about 1.35MB)

Shakey's problem solving

- given the command (in English) "push the block off the platform"
- Shakey looks around, identifies a platform with a block on it and locates a ramp
- Shakey
  - pushes the ramp over to the platform
  - rolls up the ramp onto the platform
  - pushes the block off the platform

Planning is a hard problem

- the agent's knowledge of the initial state of the world is often incomplete or mistaken
- the world is continually changing and continues to change while the agent is planning
- actions don't always have the intended effect—actions can fail or just have unpredictable outcomes
- the agent may make errors executing the plan
- the time available to find a solution may be limited

Simplifying assumptions

- the agent has perfect knowledge of the world, including the location and properties of all the objects in the world
- the world is static—i.e., it doesn't change unless the agent changes it
- the world is deterministic—we know in advance the effect of performing an action in the world and each action has a single outcome
- plan execution is error-free
- it doesn't matter how long it takes the agent to find a plan

Classical planning

- if we make these assumptions we get *classical planning*
  - production of a complete, totally ordered set of actions, which, when executed in a given initial situation, will achieve a goal
- resulting sequence of actions will typically only work if the simplifying assumptions hold
- implies some way of coping with inaccurate world models, non-deterministic actions, plan execution errors, etc. (later lecture)

Shakey's world model

- Shakey planned using the STRIPS planner
- STRIPS used information stored in a symbolic world model to determine what actions to take to achieve the robot's goal
- Shakey had an initial world model containing information about the positions of walls and doors in the environment and (possibly partial) information about other objects
- Shakey's percepts provided the information to update the representations in the world model

STRIPS

- *states* are represented as conjunctions of (function-free) ground literals
- *goals* are represented by conjunctions of literals, possibly containing existentially quantified variables
- actions are represented by *operators* which specify
  - the *name* of the action
  - the *precondition*—a conjunction of positive literals specifying what must be true for the action to be applicable
  - the *postcondition*—a conjunction of literals specifying how the situation changes when the operator is applied

STRIPS continued

- for example, an operator which stacks one block on top of another in the blocks world could be specified as

$$[Clear(x), Clear(y)]\ STACK(x, y)\ [\ On(x, y), \neg Clear(y)\ ]$$

- the precondition implicitly refers to the situation, s, immediately before the action, and the postcondition implicitly refers to the situation, s', which results from performing the action

- in s', all the positive literals in the postcondition hold, as do all the literals that held at s, except for those that are negative literals in the postcondition

Regression planning

- one way to solve STRIPS problems is to search backwards from the goal in world (situation) space
- operator preconditions become *subgoals*—we stop when the operator preconditions are satisfied in the current state
- resulting plan is a series of instantiated operators which, if applied in the initial state, result in the goal state
- searching backwards often reduces the branching factor
- in typical problems the goal state has a small number of conjuncts, each of which is made true by a small number of operators, while there are many operators that can be applied in the intial state

The next lecture

Further Deliberation

Suggested reading:

- Russell & Norvig (2003), chapter 12 (chapter 13 of the 1st edition)
- Wooldridge (2002), chapter 4.