Intelligent Agents

Formal Characteristics of State-Space Planning

Ute Schmid

Cognitive Systems, Applied Computer Science, Bamberg University

Extensions to the slides for chapter 3 of Dana Nau

last change: 7. Juni 2010

Semantics of Classical Planning

- Distinction of a syntactical planning problem and what it means
- Analogous to distinction between a logical theory and its models
- Statement of a planning problem: $P = (O, s_0, g)$
- Problem: $\mathcal{P}$ with a state-transition system $\Sigma = (S, A, \gamma)$

Semantics cont.

- Denotational semantics: Interpretation function I such that
  - For every state t of P, $I(t)$ is a state of Σ
  - For every operator instance o of P, $I(o)$ is an action of Σ

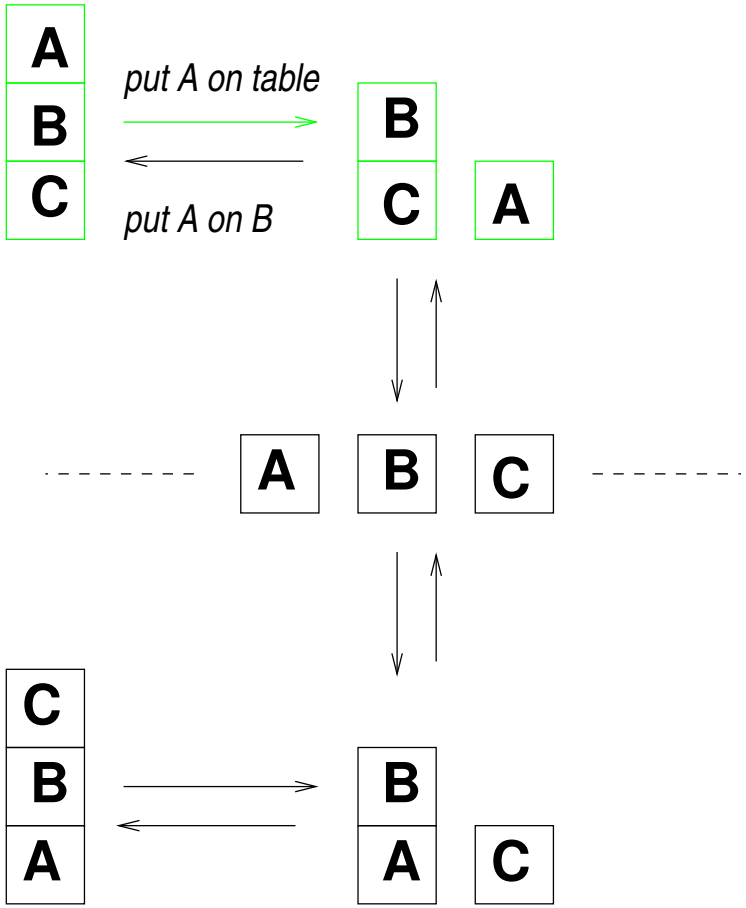
  The pair (Σ, I) is a *model* of P if for every state t of P and for every operator instance o of P holds:

$$\gamma(I(s), I(o)) = I((s - \mathit{effects}^-(o)) \cup \mathit{effects}^+(o))$$

Semantics cont.

- Distinction between syntax and semantics:
  - Syntactic calculus (automated computation)
  - with *soundness* property
- For classical representation the question whether Σ is a model of P is quite trivial; for extended representation it can get complicated (see lectures on deductive planning in FOL)

State-Space Model



- Arc from state s_i to s_i iff s_j can be reached from s_i by performing a single action.

Evaluating Planners

- **Termination** (critical case: no solution exists)
- **Soundness**: every plan returned is a legal sequence of actions to achieve the goal
  implies consistency: each intermediate state appearing in the plan is a legal state of the domain
- **Completeness**: the planner finds a solution, if one exists.
- Optimality: the returned plans are optimal (shortest) solutions (typically not considered)
- Expressiveness of the planning language
- Complexity of planning problems in a given representation formalism
- Efficiency of algorithms

Evaluation of DFS and BFS

- **Soundness:** A node s is only expanded to such a node s' where (s, s') is an arc in the state space (application of a legal operator whose preconditions are fulfilled in s)
- **Termination:** For finite sets of states guaranteed.
- **Completeness:** If a finite length solution exists.
- **Optimality:** Depth-first no, breadth-first yes
- worst case $O(b^d)$ for both, average case better for depth-first $\hookrightarrow$ If you know that there exist many solutions, that the average solution length is rather short and if the branching factor is rather high, use depth-first search, if you are not interested in the optimal but just in some admissible solution.
- Prolog is based on a depth-first search-strategy.
- Typical planning algorithms are depth-first.

Decidability and Complexity

- Decidability:
  - Can we decide for a set of problems D (e.g., all classical planning problems)
    - ⋆ whether a plan exists (PLAN-EXISTENCE(D))
    - ⋆ whether a solution contains no more than k steps (PLAN-LENGTH(D))
- Complexity:
  - How much time or space does it need to decide PLAN-EXISTENCE(D) and PLAN-LENGTH(D)

Decidability Results

- Proposition: For classical, set-theoretic, and state-variable planning PLAN-EXISTENCE(D) is decidable
- Proof idea: If the number of states is finite, we can perform brute-force search to see whether a solution exits
- Proposition: For classical and state-variable planning PLAN-LENGTH(D) is decidable, even if function symbols are allowed
- Proof idea: Do Lifted-backward-search which exits with failure if it reaches a plan of length k which is not a solution. This is a sound procedure which will always terminate. The procedure is also complete: Some execution trace will terminate in $|\Pi|$ iterations (proof by induction over the length of Π)

Semi-Decidability Result

- Proposition: If function symbols are allowed, PLAN-EXISTENCE(D) is semi-decidable
- Semi-decidability: It is possible to write a procedure that always terminates with "yes" if $\mathcal{P}$ is solvable and that never returns "yes" if $\mathcal{P}$ is unsolvable. However, if $\mathcal{P}$ is unsolvable, there is no guarantee that the procedure will terminate.
- Proof idea: It is a well-known result of logic programming that it is not decidable whether a set of Horn clauses is consistent. A planning problem P can be reformulated in a set of Horn clauses H_P. For example: `e :- p1, ..., pn` for operators with at most one positive and no negative effects. P and H_P are equivalent: If H_P is consistent, P has a solution.

O-Calculus

- Computational complexity of procedures are typically analyzed with the O-calculus
- A function $f(n)$ is in the set $O(g(n))$ if there are numbers c and n_0 such that

$$\forall n > n_0, n_0 \leq f(n) \leq cg(n)$$

- logarithmically bounded: $f(n) \in O(logn)$
- polynomially bounded: $f(n) \in O(n^c)$
- exponentially bounded: $f(n) \in O(c^n)$

Language Recognition Problems

- Complexity analyses are done on decision problems or language-recognition problems
- A language is a set L of strings over some alphabet A
- Recognition procedure:
  - $R(x)$ returns "yes" iff x is in L
  - If x is not in L, the $R(x)$ may return "no" ore may fail to terminate
- Translate classical planning in a language-recognition problem
- Examine the language-recognition problem's complexity

Complexity Classes

	NLOGSPACE	(nondeterministic procedure, logarithmic space)
$\subseteq$	P	(deterministic procedure, polynomial time)
$\subseteq$	NP	(nondeterministic procedure, polynomial time)
$\subseteq$	PSPACE	(deterministic procedure, polynomial space)
$\subseteq$	EXPTIME	(deterministic procedure, exponential time)
$\subseteq$	NEXPTIME	(nondeterministic procedure, exponential time)
$\subseteq$	EXPSPACE	(deterministic procedure, exponential space)

Let C be a complexity class and L a language

- Recognizing L is C-hard if for every language L' in C, L' can be reduced to L in polynomial time
- Recognizing L is C-complete if L is C-hard and L is also in C

CP EXPSPACE-Complete

- Unrestricted classical Planning is EXPSPACE-complete.
- Proof that PLANEXISTENCE is in EXPSPACE: Number of ground instances of predicates is exponential in terms of input length. Hence, size of a state is at most exponential. Starting with an initial state, we can nondeterministically choose an operator an apply it until we reach the goal. This is NEXPSPACE which is equal to EXPSPACE.
- Proof that PLANEXISTENCE is EXPSPACE-hard: by defining a polynomial reduction from the EXPSPACE-bounded Turing-machine problem (see page 62 of Ghallab, Nau and Traveso)

Remarks

- Often, plan length is harder than plan existance, but it is easier for classical planning (NEXPTIME-complete): we can cut of any search path at depth n
- For Tower of Hanoi, the length of the shortest plan can be found in low-order polynomial time, but producing a plan of that length requires exponential time and space