

Intelligent Agents

Formal Characteristics of Planning

Ute Schmid

Cognitive Systems, Applied Computer Science, Bamberg University

Extensions to the slides for chapter 3 of Dana Nau
with contributions by Michael Mendler

last change: 28. Mai 2014

Outline

- Semantics of classical planning
- Evaluating planning systems
- Basic Results of Decidability
- Basic Results of Complexity

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

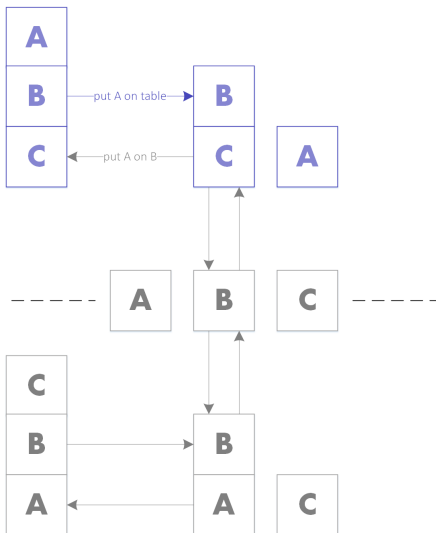
The pair (Σ, I) is a *model* of P if for every state t of P and for every operator instance o of P holds:

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

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_j 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 Depth-First-Search and Breadth-First-Search

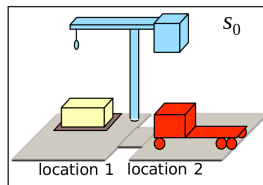
- **Soundness:** A node s is only expanded to such a node s' where (s, s') is an arc in the state space (application of a legal operator whose preconditions are fulfilled in s)
- **Termination:** For finite sets of states guaranteed.
- **Completeness:** If a finite length solution exists.
- **Optimality:** Depth-first no, breadth-first yes
- worst case $O(b^d)$ for both, average case better for depth-first \leftrightarrow 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)

Motivation

- Recall that in classical planning, even simple problems can have *huge search spaces*
 - Example:
 - ⇒ DWR with five locations, three piles, three robots, 100 containers
 - ⇒ 10^{277} states
 - ⇒ About 10^{190} times as many states as there are particles in universe
 - How difficult is it to solve classical planning problems?**
 - The answer depends on which representation scheme we use
 - Classical, set-theoretic, state-variable



Decidability & Complexity Analysis

- Complexity analyses are done on *decision problems* or *language-recognition problems* (yes-or-no answers)
- A language is a set $L \subseteq A^*$ of strings over some alphabet A
- A *recognition procedure* for L is a (possibly *non-deterministic*) Turing machine R , so that for all input strings $x \in A^*$:
 - $R(x)$ terminates (*along some computation path*) **and** returns “yes“ iff $x \in L$
 - *Notice*: If $x \notin L$, then $R(x)$ may return “no“ **or** fail to terminate
- If such a recognition procedure R exists, L is called *Turing-recognizable* (or *semi-decidable* or *recursively enumerable*)
- If R terminates for all input strings, then L is called *decidable*
- If L is (semi-)decidable we can look at the *computational complexity* (time and memory consumption of R)

Formal Languages

- A formal language is a set of strings over some alphabet. Given an alphabet A , language L is a subset of all possible words which can be generated from the alphabet. The set of all possible words is equivalent to the elements of the power set over A , that is, 2^{A^*} , $L \subseteq A^*$
- The empty word (contained in A^*) is often denoted with ϵ .
- The empty language is denoted as \emptyset .
- Example for $A = \{a, b\}$
 $A^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ a language $L_1 \subseteq A^*$ is for example $L_1 = \{a^i b^j \mid i \in \mathbb{N}_0\}$

Planning as a Language-Recognition Problem

- Translate planning into a language-recognition problem
- Fix a *class of planning problems* D
 - ⇒ **Representations**: classical, set-theoretic and state-variable
 - ⇒ **Domains**: Various restrictions and extensions on the language and operators
- Examine the language-recognition problem's decidability and complexity for D :

$PLAN-EXISTENCE(D) =$

$\{P \mid P \text{ is statement of a planning problem in } D \text{ that has a solution}\}$

$PLAN-LENGTH(D) =$

$\{(P, k) \mid P \text{ is the statement of a planning problem that has a solution of length } \leq k\}$

Basic Decidability Results

- Proposition: For classical, set-theoretic, and state-variable planning $\text{PLAN-EXISTENCE}(D)$ is decidable
- Proof idea: If the number of states is finite, we can perform brute-force search to see whether a solution exists
- Proposition: For classical and state-variable planning $\text{PLAN-LENGTH}(D)$ 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)

Basic Decidability Results

Show that Lifted-backward search is sound:

- Let $P = (O, s_0, g)$ be the statement of a classical planning problem
- let k be a non-negative integer
- Modify procedure such that it exits with failure when a plan of length k is reached which is *not* a solution
- This procedure is **sound!**

Basic Decidability Results

Show that Lifted-backward search is complete:

- Let π be any solution for P of length k or less
- If $|\pi| = 0$, then it is empty and the procedure terminates immediately
- Otherwise, let a_1 be the last action in π
- Then a_1 is *relevant* for g , so it must be a substitution instance of some operator o_i chosen in one of the procedures non-deterministic traces
- $\gamma^{-1}(g, a_1)$ is a substitution instance of $\gamma^{-1}(g, o_1)$
- If $|\pi| = 1$ then s_0 satisfies $\gamma^{-1}(g, a_1)$ and also $\gamma^{-1}(g, o_1)$ and the procedure terminates
- Otherwise, let a_2 be the second-last action in π
- Then a_2 is *relevant* for $\gamma^{-1}(g, a_2) \dots$
- We can show that some execution trace will terminate in $|\pi|$ iterations.

Semi-Decidability (Undecidability) Result

- Proposition: If function symbols are allowed, PLAN-EXISTENCE(D) is semi-decidable **but not 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: Provability in 1st-order Horn logic can be reduced to classical planning with only positive effects and preconditions:

- A Horn clause $e:- p_1, \dots, p_n$ corresponds to an operator o with $precond(o) = \{p_1, \dots, p_n\}$ and $effect(o) = \{e\}$
- A goal g is reachable iff g is derivable in Horn logic from the clauses
- *Derivability in Horn logic undecidable* ([Sebelik & Stepanek 1982]: a single constant and single unary function suffices)

Decidability of Planning

Allow function symbols?	Decidability of <i>PLAN-EXISTENCE</i>	Decidability of <i>PLAN-LENGTH</i>
no ¹	decidable	decidable
yes	semidecidable ²	decidable

¹ This is ordinary classical planning.

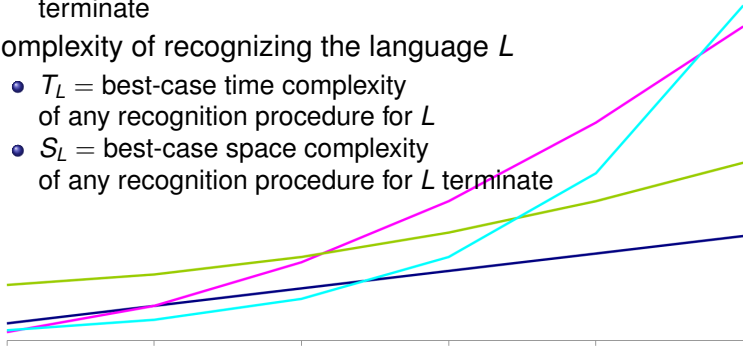
² True even if we make several restrictions.

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” or may fail to terminate
- Translate classical planning in a language-recognition problem
- Examine the language-recognition problem’s complexity

Complexity of Language-Recognition Problems

- Suppose R is a recognition algorithm for a language L
- **Complexity** of *algorithm* R
 - $T_R(n)$ = R 's worst-case time complexity on strings in L of length n
 - $S_R(n)$ = R 's worst-case space complexity on strings in L of length n terminate
- Complexity of recognizing the language L
 - T_L = best-case time complexity of any recognition procedure for L
 - S_L = best-case space complexity of any recognition procedure for L terminate



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

$$\forall n > n_0, n_0 \leq f(n) \leq c \times g(n)$$

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

Complexity Classes

- *Complexity classes:*

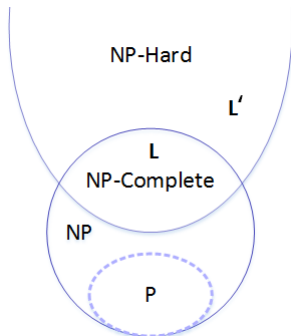
	NLOGSPACE	(non-deterministic procedure, logarithmic space)
⊂	P	(deterministic procedure, polynomial time)
⊂	NP	(non-deterministic procedure, polynomial time)
⊂	PSPACE	(deterministic procedure, polynomial space)
⊂	EXPTIME	(deterministic procedure, exponential time)
⊂	NEXPTIME	(non-deterministic procedure, exponential time)
⊂	EXPSPACE	(deterministic procedure, exponential space)

- Let C be a complexity class and L a language

- L is *C-hard* if for every language $L' \in C$, L' can be *reduced* to L in a *polynomial amount of time*
 - ⇒ NP-hard, PSPACE-hard, etc.
- L is *C-complete* if L is *C-hard* and $L \in C$
 - ⇒ NP-complete, PSPACE-complete, etc.

Hardness and Completeness

- C-complete:**
 L is known to be in C , is a “typical representative”
 (e.g., boolean satisfiability problem, travelling salesman problem)
- C-hard:**
 every L' can be transformed into $L \in C$ in polynomial time;
 that is: L' is in C or a higher complexity class



Classical Planning is 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 non-deterministically choose an operator and apply it until we reach the goal. This is NEXPSPACE which is equal to EXPSPACE.
- This holds, if operators are part of the input, that is, not defined in advance (to enable the use of a domain-specific algorithm).

Proof see chap 3.4 in Ghallab, Malik, Dana Nau, and Paolo Traverso. Automated planning: theory & practice. Elsevier, 2004.

D = CP without Negative Effects

If there are *no negative effects*, then in searching for a solution plan ...

- ... whenever a closed atom (ground predicate instance) enters the state, it remains part of the state for throughout the plan
- ... no operator needs to be instantiated more than once

Since the *number of possible operator instances* is at most **exponential**

- ... we can stop the (depth-first) search after at most **exponentially many steps**

PLANEXISTENCE(D) 2 NEXPTIME

D = CP without Negative Effects or Negative Preconditions

If, there are neither negative effects *nor negative preconditions*, then

- ... *operators do not conflict* with each other
- ... once an operator is applicable, it remains applicable for throughout the plan

Since the *order* in which the operators are applied *does not matter*

- ... in forward search, simply apply any operator instance that is applicable in the current state and consistent with goal
- ... we **do not need to backtrack**

PLANEXISTENCE(D) 2 EXPTIME

D = CP with $\cdot 1$ Preconditions or Operators Fixed in Advance

If each operator has **at most one precondition**, then

- ... in backward search, the number of literals in the goal never increases
- ... size of the *current goal* remains *polynomial*

If the operators are **fixed in advance** (arity of operators and predicates), then

- ... there are at most *polynomially many ground atoms*
- ... the size of a state or sub-goal is at most polynomial

PLANEXISTENCE(D) \leq NPSPACE = PSPACE

- PLAN-LENGTH is never worse than NEXPTIME-complete

- length may be double-exponential, but we can cut off every search path at depth $k = O(2^n)$

Kind of representation	How the operators are given	Allow negative effects?	Allow negative preconditions?	Complexity of <i>PLAN-EXISTENCE</i>	Complexity of <i>PLAN-LENGTH</i>
classical rep.	in the input	yes	yes/no	EXPSpace-complete	NEXPTIME-complete
		no	yes	NEXPTIME-complete	NEXPTIME-complete
			no	EXPTIME-complete	NEXPTIME-complete
			no^{α}	PSPACE-complete	PSPACE-complete
	in advance	yes	yes/no	PSPACE $^{\gamma}$	PSPACE $^{\gamma}$
		no	yes	NP $^{\gamma}$	NP $^{\gamma}$
			no	P	NP $^{\gamma}$
			no^{α}	NLOGSPACE	NP

Here PLAN-LENGTH is harder than PLAN-EXISTENCE

- In this case, we can write domain-specific algorithms

- e.g., DWR and Blocks World:

PLAN-EXISTENCE is in P and PLAN-LENGTH is NP-complete

Kind of representation	How the operators are given	Allow negative effects?	Allow negative preconditions?	Complexity of <i>PLAN-EXISTENCE</i>	Complexity of <i>PLAN-LENGTH</i>
classical rep.	in the input	yes	yes/no	EXSPACE-complete	NEXPTIME-complete
		no	yes	<i>NEXPTIME-complete</i>	<i>NEXPTIME-complete</i>
			no	<i>EXPTIME-complete</i>	<i>NEXPTIME-complete</i>
			no ^α	<i>PSPACE-complete</i>	<i>PSPACE-complete</i>
	in advance	yes	yes/no	PSPACE ^γ	PSPACE ^γ
		no	yes	NP ^γ	NP ^γ
			no	P	NP ^γ
			no ^α	<i>NLOGSPACE</i>	NP

no operator has > 1 preconditions

PSPACE-complete or NP-complete for some sets of operators

Remarks

- Often, plan length is harder than plan existence, but it is easier for classical planning (NEXPTIME-complete):
we can cut off 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

Summary

- Semantics of classical planning is based on interpreting problem representations in a state-space model
- Planning is a calculus, i.e. a syntactic way to compute sound and complete solutions
- If *classical planning* is extended to allow *function symbols*
 - Then we can *encode arbitrary computations* as planning problems
 - ⇒ Plan existence is *only* semi-decidable
 - ⇒ *Only* plan length is decidable
- Ordinary classical planning is quite complex
 - ⇒ Plan existence is *EXPSPACE-complete*
 - ⇒ Plan length is *NEXPTIME-complete*
 - But those are *worst case* results
 - ⇒ If we can write *domain-specific algorithms*, most well-known planning problems are *much easier*