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: 19. Mai 2015
Outline

- Semantics of classical planning
- Evaluating planning systems
- Basic Results of Decidability
- Basic Results of Complexity
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 \( \Sigma \)
  - For every operator instance \( o \) of \( P \), \( I(o) \) is an action of \( \Sigma \)

The pair \((\Sigma, I)\) 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 $\Sigma$ is a model of $P$ is quite trivial; for extended representation it can get complicated (see lectures on deductive planning in FOL)
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

- The worst case $O(b^d)$ for both, average case better for depth-first $\rightarrow$ 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:
- Can we decide for a set of problems $D$ (e.g., all classical planning problems)
  - whether a plan exists ($\text{PLAN-EXISTENCE}(D)$)
  - whether a solution contains no more than $k$ steps ($\text{PLAN-LENGTH}(D)$)

Complexity:
- How much time or space does it need to decide $\text{PLAN-EXISTENCE}(D)$ and $\text{PLAN-LENGTH}(D)$
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
### Number of States for DWR

An example calculation (our estimate is a bit lower than that of Nau)

5 locations, 3 piles, 3 robots, 100 containers

<table>
<thead>
<tr>
<th>State</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>occupied(l)</td>
<td>$2^5 = 32$</td>
</tr>
<tr>
<td>at(r,l)</td>
<td>$5 \times 4 \times 3 = 60$</td>
</tr>
<tr>
<td>loaded(r,c), unloaded(r)</td>
<td>each robot can hold none or one of the containers</td>
</tr>
<tr>
<td>holding(k,c), empty(k)</td>
<td>“n over k” $\binom{101!}{98! \times 3!} = 166.650$</td>
</tr>
<tr>
<td>in(c,p), on(c,p), top(c,p)</td>
<td>$\binom{101!}{98! \times 5!} = 8332$</td>
</tr>
</tbody>
</table>

$\sum \quad 10^{205}$
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 $R$).
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 $\epsilon$.

The empty language is denoted as $\emptyset$.

Example for $A = \{a, b\}$

$A^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$ a language $L_1 \subseteq A^*$ is for example $L_1 = \{a^i b^i \mid i \in N_0\}$
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 $D$:
  - $PLAN\text{-}EXISTENCE(D) = \{ P \mid P$ is statement of a planning problem in $D$ that has a solution $\}$
  - $PLAN\text{-}LENGTH(D) = \{ (P, k) \mid P$ 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 exits

- 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 $\pi$)
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 $\pi$ 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 $\pi$
- 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 $\pi$
- Then $a_2$ is relevant for $\gamma^{-1}(g, a_2)$
- 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 $P$ is solvable and that never returns “yes“ if $P$ is unsolvable. However, if $P$ 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 $e: - p_1, \ldots, p_n$ corresponds to an operator $o$ with $\text{precond}(o) = \{p_1, \ldots, p_1\}$ and $\text{effect}(o) = \{e\}$
- 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

<table>
<thead>
<tr>
<th>Allow function symbols?</th>
<th>Decidability of PLAN-EXISTENCE</th>
<th>Decidability of PLAN-LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>no(^1)</td>
<td>decidable</td>
<td>decidable</td>
</tr>
<tr>
<td>yes</td>
<td>semidecidable(^2)</td>
<td>decidable</td>
</tr>
</tbody>
</table>

\(^1\) 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.
Basic Complexity Results

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: $f(n) \in O(c^n)$
Basic Complexity Results

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*
    - $\Rightarrow$ NP-hard, PSPACE-hard, etc.
  - $L$ is *C-complete* if $L$ is C-hard and $L \in C$
    - $\Rightarrow$ NP-complete, PSPACE-complete, etc.
**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
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).

If there are no negative effects, then in searching for a solution plan \ldots
\ldots whenever a closed atom (ground predicate instance) enters the state, it remains part of the state for throughout the plan
\ldots no operator needs to be instantiated more than once

Since the number of possible operator instances is at most exponential
\ldots we can stop the (depth-first) search after at most exponentially many steps

\textit{PLANEXISTENCE}(D) is in \textit{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) is in EXPTIME.
D = CP with at Most 1 Precondition 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) is in NPSPACE = PSPACE
**Basic Complexity Results**

- **PLAN-LENGTH** is never worse than **NEXPTIME-complete**
  - length may be double-exponential, but we can cut off every search path at depth $k = O(2^n)$

<table>
<thead>
<tr>
<th>Kind of representation</th>
<th>How the operators are given</th>
<th>Allow negative effects?</th>
<th>Allow negative preconditions?</th>
<th>Complexity of <strong>PLAN-EXISTENCE</strong></th>
<th>Complexity of <strong>PLAN-LENGTH</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>classical rep.</td>
<td>in the input</td>
<td>yes</td>
<td>yes/no</td>
<td>EXPSPACE-complete</td>
<td>NEXPTIME-complete</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
<td></td>
<td>NEXPTIME-complete</td>
<td>NEXPTIME-complete</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>no</td>
<td></td>
<td>EXPTIME-complete</td>
<td>NEXPTIME-complete</td>
</tr>
<tr>
<td></td>
<td>no$^\alpha$</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
<td>PSPACE$^\gamma$</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>yes/no</td>
<td>PSPACE$^\gamma$</td>
<td>NP$^\gamma$</td>
<td>NP$^\gamma$</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
<td>NP$^\gamma$</td>
<td>P</td>
<td>NP$^\gamma$</td>
</tr>
<tr>
<td></td>
<td>no$^\alpha$</td>
<td>P</td>
<td>NP$^\gamma$</td>
<td>NLOGSPACE</td>
<td>NP</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Kind of representation</th>
<th>How the operators are given</th>
<th>Allow negative effects?</th>
<th>Allow negative preconditions?</th>
<th>Complexity of PLAN-EXISTENCE</th>
<th>Complexity of PLAN-LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>classical rep.</td>
<td>in the input</td>
<td>yes</td>
<td>yes/no</td>
<td>EXPSPACE-complete</td>
<td>NEXPTIME-complete</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td></td>
<td></td>
<td>NEXPTIME-complete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
<td></td>
<td>EXPSPACE-complete</td>
<td>NEXPTIME-complete</td>
</tr>
<tr>
<td></td>
<td>noα</td>
<td></td>
<td></td>
<td>PSPACE-complete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>yes/no</td>
<td></td>
<td>PSPACE(^\gamma)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
<td></td>
<td>NP(^\gamma)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>noα</td>
<td></td>
<td></td>
<td>NLOGSPACE</td>
<td></td>
</tr>
</tbody>
</table>

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

no operator has > 1 preconditions
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.