# Computational Complexity and Expressive Power in Classical AI Planning

## Part II: Computational Complexity of Planning

*Bernhard Nebel*

Albert-Ludwigs-Universität Freiburg
Department of Computer Science



## Basic STRIPS with First-Order STRIPS

- The state space is **infinite**
- We can use function terms to describe (the index of a) tape cells of a Turing machine
- We can use operator schemata to describe the Turing machine control
- The existence of a plan is then equivalent to the existence of a successful computation
- PLANEX for STRIPS with first-order terms can be used to decide the **Halting problem**

**Theorem**. PLANEX for STRIPS with first-order terms is **undecidable**.



## Turing Machines

Deterministic Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, \sharp, q_0, F\rangle$:

- $Q$: finite set of states
- $\Sigma$: finite input alphabet
- $\Gamma \supseteq \Sigma$: finite tape alphabet
- $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ transition function
- $\sharp \in \Sigma$: blank symbol
- $q_0 \in Q$: initial state
- $F \subseteq Q$: terminal states



## Reduction

Let $w \in \Sigma^*$ be an input string and let $M = \langle Q, \Sigma, \Gamma, \delta, \sharp, q_0, F\rangle$ be a deterministic Turing machine

$\leadsto$ Planning instance $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G}\rangle$:

- Let the signature be $\mathcal{S} = \langle \{\mathsf{s}(\cdot)\}, Q \cup \Gamma \cup \{\ddagger\}, \{\mathsf{acc}, \mathsf{in}(\cdot,\cdot), \mathsf{last}(\cdot), \mathsf{at}(\cdot,\cdot)\}\rangle$
- For given input string $w = a_1 \dots a_n$, the initial state and goal specification is

$$\begin{aligned}\mathbf{I} &= \{\mathsf{in}(\mathsf{s}(\ddagger), a_1), \dots, \mathsf{in}(\mathsf{s}^n(\ddagger), a_n), \mathsf{last}(\mathsf{s}^n(\ddagger)), \mathsf{at}(\mathsf{s}(\ddagger), q_0)\},\\ \mathbf{G} &= \{\mathsf{acc}\}.\end{aligned}$$

- For each transition of the form $\delta(q, a) = \langle q', a', R\rangle$ the following operator schema is included (similar for left moves!)

$$\langle\langle X\rangle,\ \{\mathsf{at}(X, q), \mathsf{in}(X, a)\},\ \{\mathsf{at}(\mathsf{s}(X), q'), \mathsf{in}(X, a'), \neg\mathsf{at}(X, q), \neg\mathsf{in}(X, a)\}\rangle.$$

- Extending the tape on the right side:

$$\langle\langle X\rangle,\ \{\mathsf{last}(X)\},\ \{\mathsf{in}(\mathsf{s}(X), \sharp), \mathsf{last}(\mathsf{s}(X)), \neg\mathsf{last}(X)\}\rangle.$$

- For each terminal state $q_f \in F$: $\langle\langle X\rangle,\ \{\mathsf{at}(X, q_f)\},\ \{\mathsf{acc}\}\rangle$.

## Simplifications

- Only positive effects (deletion-free STRIPS) decidable?

$\leadsto$ Reduction from logic programming: Undecidability!

- PLANLEN decidable?

$\leadsto$ ?

- Function-free STRIPS decidable?

$\leadsto$ Finite state space, but EXPSPACE-complete (idea: generic reduction, use a counter to initialize tape)



## Propositional STRIPS

**Theorem**. PLANEX is **PSPACE-complete** for propositional STRIPS.

$\rightarrow$ Membership follows because we can successively guess operators and compute the resulting states (needs only polynomial space)

$\rightarrow$ Hardness follows using again a generic reduction. Instantiate polynomially many tape cells with no possibility to extend the tape (only poly. space, can all be generated in poly. time)

- PLANLEN is also PSPACE-complete (membership is easy, hardness follows by setting $n = 2^{|\Sigma|}$)



## Restrictions on Plans

- If we restrict the length of the plans to be only **polynomial** in the size of the planning task, PLANEX becomes NP-complete

$\rightarrow$ Membership obvious (guess & check)

$\rightarrow$ Hardness by a straightforward reduction from SAT or by a generic reduction.

$\leadsto$ One source of complexity in planning stems from the fact that plans can become very long

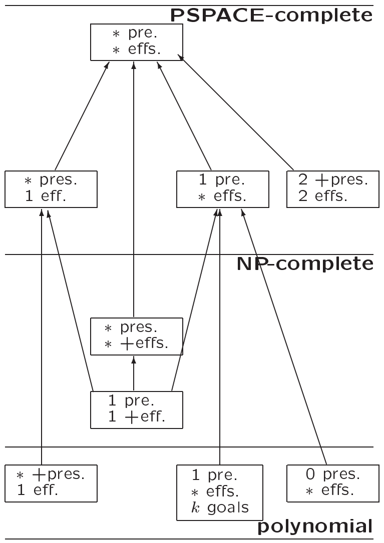
$\leadsto$ We are only interested in short plans!

$\leadsto$ We can use methods for NP-complete problems if we are only looking for "short" plans.



## Syntactic Restrictions for STRIPS$_N$

- $n$ pres./effs. means that the number of *preconditions* resp. *effect* is restricted to $n$ ("*" means no restriction)
- +pres./+effs. means only positive *preconditions* resp. *effects*
- $k$ goals means that at most $k$ *goals* are allowed



## Propositional, Precondition-free STRIPS$_N$

**Theorem**. The problem of deciding plan existence for precondition-free, propositional STRIPS$_N$ is polynomial.

**Proof**. Do a backward greedy plan generation. Choose all operators that make some goals true and that do not make goals false. Remove the satisfied goals and the operators from further consideration and iterate the step until all remaining goals are satisfied by the initial state (succeed) or no more operators can be applied (fail).



## Propositional, Precondition-free STRIPS$_N$ and Plan Length

**Theorem** The problem of deciding whether there exists a plan of length $k$ for precondition-free, propositional STRIPS$_N$ is NP-complete, even if all effects are positive.

**Proof**. Membership in NP is obvious. Hardness follows from a straightforward reduction from the MINIMUM-COVER problem [Garey & Johnson 79]:

> Given a collection $C$ of subsets of a finite set $S$ and a positive integer $k$, does there exist a cover for $S$ of size $k$ or less, i.e., a subset $C' \subseteq C$ such that $\bigcup C' \supseteq S$ and $|C'| \leq k$?

This is unfortunate because we could use this problem for estimating the distance to the goal when using heuristic search in planning (HSP, FF).



## NP-complete Planning: One Precondition and one Positive Effect

**Membership** in P follows because all effects are positive so that an operator should be applied only once $\rightsquigarrow$ easy guess & check in poly. time.

**Hardness** follows by a straightforward reduction from SAT (assuming CNF).

- Propositional atoms in the planning instance: $T_i$ (the $i$th atom is true), $F_i$ (... false), and $C_j$ ($j$th clause is true)
- Initial state $\mathbf{I} = \emptyset$, goal $\mathbf{G} = \{C_1, \ldots, C_m\}$
- Truth value choosers: $\langle\{\neg F_i\}, \{T_i\}\rangle$, $\langle\{\neg T_i\}, \{F_i\}\rangle$.
- Literal evaluator (positive occurence of $i$th atom in $j$th clause): $\langle\{T_i\}, \{C_j\}\rangle$
- Negative occurrence: $\langle\{F_i\}, \{C_j\}\rangle$

$\rightsquigarrow$ There exists a (poly size) plan if the formula is satisfiable.



## Domain-Dependent Planning

Planning (and its complexity) for particular domains is interesting – since we may decide to go for **domain-dependent** planning.

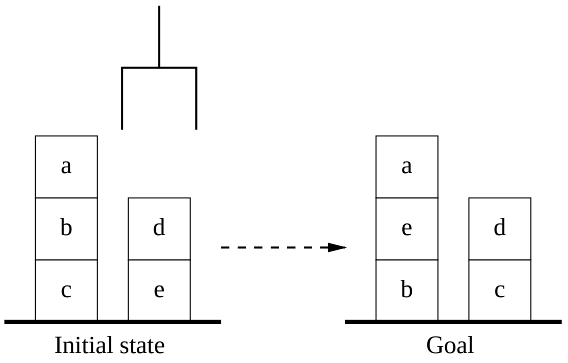
**Theorem**. DATALOG-STRIPS-PLANEX with a fixed set of operators is PSPACE-complete.

**Proof**. Membership follows because the arity of predicates in the operator schemata is fixed in advance. For this reason, we have to generate only polynomially many (relevant) operators and ground atoms for a given set of constants and planning can be reduced to the propositional case.

Hardness follows for the fixed set of operator schemata for a fixed PSPACE-complete language constructed in the PSPACE-completeness proof for propositional STRIPS.

Propositional STRIPS with a fixed operator set is trivial!

## Blocks-World Planning






## Blocks-World: A Formalization

- All blocks are represented by **constant symbols**
- **Predicates**: on$(\cdot,\cdot)$, ontable$(\cdot)$, clear$(\cdot)$, holding$(\cdot)$, and arm-empty.
- **Operators**:

$$
\begin{aligned}
\mathrm{stack}(x,y) &= \langle\langle x,y\rangle, \\
&\quad \{\mathrm{clear}(y), \mathrm{holding}(x)\}, \\
&\quad \{\mathrm{on}(x,y), \text{arm-empty}, \mathrm{clear}(x), \neg\mathrm{clear}(y), \neg\mathrm{holding}(x)\}\rangle, \\
\mathrm{unstack}(x,y) &= \langle\langle x,y\rangle, \\
&\quad \{\mathrm{on}(x,y), \mathrm{clear}(x), \text{arm-empty}\}, \\
&\quad \{\mathrm{holding}(x), \mathrm{clear}(y), \neg\mathrm{on}(x,y), \neg\mathrm{clear}(x), \neg\text{arm-empty}\}\rangle, \\
\mathrm{pickup}(x) &= \langle\langle x\rangle, \\
&\quad \{\mathrm{clear}(x), \mathrm{ontable}(x), \text{arm-empty}\}, \\
&\quad \{\mathrm{holding}(x), \neg\mathrm{clear}(x), \neg\mathrm{ontable}(x), \neg\text{arm-empty}\}\rangle, \\
\mathrm{putdown}(x) &= \langle\langle x\rangle, \\
&\quad \{\mathrm{holding}(x)\}, \\
&\quad \{\mathrm{ontable}(x), \mathrm{clear}(x), \neg\mathrm{holding}(x), \neg\text{arm-empty}\}\rangle.
\end{aligned}
$$



## A Simple Blocks-World Planning Algorithm

**Algorithm 1** simple blocks-world planner

**Input:** Blocks-world instance
**Output:** Blocks-world plan
1. Place each block on the table (using a sequence of unstack/putdown)
2. Build up the goal stacks block by block (using a sequence of pickup/stack

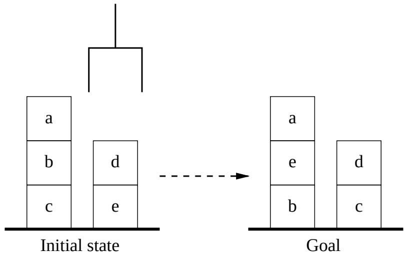
$\leadsto$ Instances are always solvable
$\leadsto$ Plans of poly. size
$\leadsto$ How difficult is it to generate a minimal length plan?



## Deadlocks etc.

- A block $b_1$ is **above** block $b_k$ in a state $S$ if there exists a set of atoms $\{\mathrm{on}(b_1, b_2), \ldots, \mathrm{on}(b_{k-1}, b_k\} \subseteq S$.
- Similarly, **below**
- A block $b$ is in its **final position** in state $S$ if all the blocks below it are already in the right order as required in the goal description and the block itself appears exactly in this position in the goal description.
- A set of blocks $\{b_1, \ldots, b_p\}$ is **deadlocked** in state $S$ if there exists a set of blocks $\{d_1, \ldots, d_p\}$ such that
  1. In state $S$, $b_i$ is above $d_i$ for $i = 1, 2, \ldots, p$,
  2. in $S$, no $b_i$ is in its final position for $i = 1, 2, \ldots, p$, and
  3. in the goal description $\mathbf{G}$, $b_i$ is above $d_{i+1}$ for $i = 1, 2, \ldots, p-1$, and $b_p$ is above $d_1$.

## Deadlock Example




- The set {a, d} is deadlocked because a is above c and d is above e in the initial state and a is above e and d is above c in the goal description.
- The set {a} is deadlocked because a is above b in the initial state and in the goal description.



## A Fast, Nondeterministic Blocks-World Planning Algorithm

**Algorithm 2** block-world planner

**Input:** Blocks-world instance
**Output:** Blocks-world plan

1. Test whether all blocks in $\mathbf{G}$ appear also in $\mathbf{I}$ and whether $\mathbf{I}$ and $\mathbf{G}$ are physically possible state descriptions. If not, fail.
2. Test whether $S \models \mathbf{G}$. If so, exit with success.
3. If a block $b$ can be directly moved to its final position, then do it using two operations. Go to step (2).
4. If this step is reached, all blocks that are clear and not in their final positions participate in some deadlocked set. Resolve one of those deadlocks and go to step (2).

Let $B$ be the number of blocks and $F$ be the number of blocks already in their final position in $\mathbf{I}$. Then the above algorithm produces plans of length $4 \times (B - F)$ or less.



## What Makes Blocks-World Planning Difficult?

- The deadlocks make it difficult. Use **Feedback-Arc-Set** to prove that the problem is indeed NP-hard.
- Blocks-world planning served as a **standard problem** for many years, but appears to be very special – for a number of reasons.
- But then, we do not have a classification of planning systems yet ...