Principles of AI Planning

Computational complexity

Malte Helmert Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

December 15th, 2006

- We have seen that planning in transition systems can be done in time polynomial in the size of the transition system
- This appears not to be true for planning in succinct transition systems (= planning tasks)

1. What is the precise computational complexity of the **planning problem**?
2. How does the computational complexity vary with the expressiveness of the planning language?
3. What is the computational complexity of planning in a particular domain (e.g. blocks world)?

- We have seen that planning in transition systems can be done in time polynomial in the size of the transition system
- This appears not to be true for planning in succinct transition systems (= planning tasks)

1. What is the precise computational complexity of the **planning problem**?
2. How does the computational complexity vary with the expressiveness of the planning language?
3. What is the computational complexity of planning in a particular domain (e.g. blocks world)?

- We have seen that planning in transition systems can be done in time polynomial in the size of the transition system
- This appears not to be true for planning in succinct transition systems (= planning tasks)

1. What is the precise computational complexity of the **planning problem**?
2. How does the computational complexity vary with the expressiveness of the planning language?
3. What is the computational complexity of planning in a particular domain (e.g. blocks world)?

- We have seen that planning in transition systems can be done in time polynomial in the size of the transition system
- This appears not to be true for planning in succinct transition systems (= planning tasks)

1. What is the precise computational complexity of the **planning problem**?
2. How does the computational complexity vary with the expressiveness of the planning language?
3. What is the computational complexity of planning in a particular domain (e.g. blocks world)?

Why Computational Complexity?

- understand the problem
- know what is not possible
- find interesting subproblems that are easier to solve
- distinguish essential features from syntactic sugar

Deterministic planning: NP-hardness

Definition

The decision problem SAT: test whether a given propositional formula ϕ is satisfiable.

Reduction from SAT to deterministic planning

$$\begin{aligned} A &= \text{the set of propositional variables occurring in } \phi \\ I &= \text{any state, e.g. all state variables have value 0} \\ O &= (\{\top\} \times A) \cup (\{\langle \top, \neg a\rangle | a \in A\}) \end{aligned}$$

There is a plan for $\langle A, I, O, \phi \rangle$ if and only if ϕ is satisfiable.

Deterministic planning: NP-hardness

- Because there is a polynomial-time translation from SAT into deterministic planning, and SAT is an NP-complete problem, there is a polynomial time translation from every decision problem in NP into deterministic planning. Hence the problem is NP-hard.

1. Does NP-hardness depend on having Boolean formulae as preconditions?
2. Does deterministic planning have the power of NP, or is it still more powerful?
3. We show that it is more powerful: The decision problem of testing whether a plan exists is PSPACE-complete.

Deterministic planning: NP-hardness

- Because there is a polynomial-time translation from SAT into deterministic planning, and SAT is an NP-complete problem, there is a polynomial time translation from every decision problem in NP into deterministic planning. Hence the problem is NP-hard.

1. Does NP-hardness depend on having Boolean formulae as preconditions?
2. Does deterministic planning have the power of NP, or is it still more powerful?
3. We show that it is more powerful: The decision problem of testing whether a plan exists is PSPACE-complete.

Deterministic planning: NP-hardness

- Because there is a polynomial-time translation from SAT into deterministic planning, and SAT is an NP-complete problem, there is a polynomial time translation from every decision problem in NP into deterministic planning. Hence the problem is NP-hard.

1. Does NP-hardness depend on having Boolean formulae as preconditions?
2. Does deterministic planning have the power of NP, or is it still more powerful?
3. We show that it is more powerful: The decision problem of testing whether a plan exists is PSPACE-complete.

Deterministic planning: NP-hardness

- Because there is a polynomial-time translation from SAT into deterministic planning, and SAT is an NP-complete problem, there is a polynomial time translation from every decision problem in NP into deterministic planning. Hence the problem is NP-hard.

1. Does NP-hardness depend on having Boolean formulae as preconditions?
2. Does deterministic planning have the power of NP, or is it still more powerful?
3. We show that it is more powerful: The decision problem of testing whether a plan exists is PSPACE-complete.

Alternating Turing Machines

Definition: Alternating Turing Machine

Alternating Turing Machine (ATM) $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$:

1. input alphabet Σ and blank symbol $\Box \notin \Sigma$
   - alphabets always non-empty and finite
   - tape alphabet $\Sigma_\Box = \Sigma \cup \{\Box\}$
2. finite set Q of internal states with initial state $q_0 \in Q$
3. state labeling $l : Q \to \{\mathsf{Y}, \mathsf{N}, \exists, \forall\}$
   - accepting, rejecting, existential, universal states Q_Y, Q_N, $Q_\exists$, $Q_\forall$
   - terminal states $Q_\star = Q_\mathsf{Y} \cup Q_\mathsf{N}$
   - nonterminal states $Q' = Q_\exists \cup Q_\forall$
4. transition relation $\delta \subseteq (Q' \times \Sigma_\Box) \times (Q \times \Sigma_\Box \times \{-1, +1\})$

Alternating Turing Machines

Definition: Alternating Turing Machine

Alternating Turing Machine (ATM) $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$:

1. input alphabet Σ and blank symbol $\Box \notin \Sigma$
   - alphabets always non-empty and finite
   - tape alphabet $\Sigma_\Box = \Sigma \cup \{\Box\}$
2. finite set Q of internal states with initial state $q_0 \in Q$
3. state labeling $l : Q \to \{\mathsf{Y}, \mathsf{N}, \exists, \forall\}$
   - accepting, rejecting, existential, universal states Q_Y, Q_N, $Q_\exists$, $Q_\forall$
   - terminal states $Q_\star = Q_\mathsf{Y} \cup Q_\mathsf{N}$
   - nonterminal states $Q' = Q_\exists \cup Q_\forall$
4. transition relation $\delta \subseteq (Q' \times \Sigma_\Box) \times (Q \times \Sigma_\Box \times \{-1, +1\})$

Alternating Turing Machines

Definition: Alternating Turing Machine

Alternating Turing Machine (ATM) $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$:

1. input alphabet Σ and blank symbol $\Box \notin \Sigma$
   - alphabets always non-empty and finite
   - tape alphabet $\Sigma_\Box = \Sigma \cup \{\Box\}$
2. finite set Q of internal states with initial state $q_0 \in Q$
3. state labeling $l : Q \to \{\mathsf{Y}, \mathsf{N}, \exists, \forall\}$
   - accepting, rejecting, existential, universal states Q_Y, Q_N, $Q_\exists$, $Q_\forall$
   - terminal states $Q_\star = Q_\mathsf{Y} \cup Q_\mathsf{N}$
   - nonterminal states $Q' = Q_\exists \cup Q_\forall$
4. transition relation $\delta \subseteq (Q' \times \Sigma_\Box) \times (Q \times \Sigma_\Box \times \{-1, +1\})$

Alternating Turing Machines

Definition: Alternating Turing Machine

Alternating Turing Machine (ATM) $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$:

1. input alphabet Σ and blank symbol $\Box \notin \Sigma$
    - alphabets always non-empty and finite
    - tape alphabet $\Sigma_\Box = \Sigma \cup \{\Box\}$
2. finite set Q of internal states with initial state $q_0 \in Q$
3. state labeling $l : Q \to \{\mathsf{Y}, \mathsf{N}, \exists, \forall\}$
    - accepting, rejecting, existential, universal states Q_Y, Q_N, $Q_\exists$, $Q_\forall$
    - terminal states $Q_\star = Q_\mathsf{Y} \cup Q_\mathsf{N}$
    - nonterminal states $Q' = Q_\exists \cup Q_\forall$
4. transition relation $\delta \subseteq (Q' \times \Sigma_\Box) \times (Q \times \Sigma_\Box \times \{-1, +1\})$

Alternating Turing Machines

Definition: Alternating Turing Machine

Alternating Turing Machine (ATM) $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$:

1. input alphabet Σ and blank symbol $\Box \notin \Sigma$
   - alphabets always non-empty and finite
   - tape alphabet $\Sigma_\Box = \Sigma \cup \{\Box\}$
2. finite set Q of internal states with initial state $q_0 \in Q$
3. state labeling $l : Q \to \{\mathsf{Y}, \mathsf{N}, \exists, \forall\}$
   - accepting, rejecting, existential, universal states Q_Y, Q_N, $Q_\exists$, $Q_\forall$
   - terminal states $Q_\star = Q_\mathsf{Y} \cup Q_\mathsf{N}$
   - nonterminal states $Q' = Q_\exists \cup Q_\forall$
4. transition relation $\delta \subseteq (Q' \times \Sigma_\Box) \times (Q \times \Sigma_\Box \times \{-1, +1\})$

(Non-) Deterministic Turing Machines

Definition: Non-deterministic Turing Machine

A non-deterministic Turing Machine (NTM) is an ATM where all nonterminal states are existential.

- no universal states

Definition: Deterministic Turing Machine

A deterministic Turing Machine (DTM) is an NTM where the transition relation is functional.

- for all $(q, a) \in Q' \times \Sigma_\square$, there is exactly one triple (q', a', Δ) with $((q, a), (q', a', \Delta)) \in \delta$
- notation: $\delta(q, a) = (q', a', \Delta)$

(Non-) Deterministic Turing Machines

Definition: Non-deterministic Turing Machine

A non-deterministic Turing Machine (NTM) is an ATM where all nonterminal states are existential.

- no universal states

Definition: Deterministic Turing Machine

A deterministic Turing Machine (DTM) is an NTM where the transition relation is functional.

- for all $(q, a) \in Q' \times \Sigma_\Box$, there is exactly one triple (q', a', Δ) with $((q, a), (q', a', \Delta)) \in \delta$
- notation: $\delta(q, a) = (q', a', \Delta)$

Turing Machine Configurations

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Configuration

A configuration of M is a triple $(w, q, x) \in \Sigma_\Box^* \times Q \times \Sigma_\Box^+$.

- w: tape contents before tape head
- q: current state
- x: tape contents after and including tape head

Turing Machine Transitions

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Yields relation

A configuration c of M **yields** a configuration c' of M, in symbols $c \vdash c'$, as defined by the following rules, where $a, a', b \in \Sigma_\Box$, $w, x \in \Sigma_\Box^*$, $q, q' \in Q$ and $((q, a), (q', a', \Delta)) \in \delta$:

$$
\begin{aligned}
(w, q, ax) &\vdash (wa', q', x) && \text{if } \Delta = +1, |x| \geq 1 \\
(w, q, a) &\vdash (wa', q', \Box) && \text{if } \Delta = +1 \\
(wb, q, ax) &\vdash (w, q', ba'x) && \text{if } \Delta = -1 \\
(\epsilon, q, ax) &\vdash (\epsilon, q', \Box a'x) && \text{if } \Delta = -1
\end{aligned}
$$

Turing Machine Transitions

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Yields relation

A configuration c of M yields a configuration c' of M, in symbols $c \vdash c'$, as defined by the following rules, where $a, a', b \in \Sigma_\Box$, $w, x \in \Sigma_\Box^*$, $q, q' \in Q$ and $((q, a), (q', a', \Delta)) \in \delta$:

$$
\begin{aligned}
(w, q, ax) &\vdash (wa', q', x) && \text{if } \Delta = +1, |x| \geq 1 \\
(w, q, a) &\vdash (wa', q', \Box) && \text{if } \Delta = +1 \\
(wb, q, ax) &\vdash (w, q', ba'x) && \text{if } \Delta = -1 \\
(\epsilon, q, ax) &\vdash (\epsilon, q', \Box a'x) && \text{if } \Delta = -1
\end{aligned}
$$

Turing Machine Transitions

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Yields relation

A configuration c of M **yields** a configuration c' of M, in symbols $c \vdash c'$, as defined by the following rules, where $a, a', b \in \Sigma_\Box$, $w, x \in \Sigma_\Box^*$, $q, q' \in Q$ and $((q, a), (q', a', \Delta)) \in \delta$:

$$
\begin{aligned}
(w, q, ax) &\vdash (wa', q', x) && \text{if } \Delta = +1, |x| \geq 1 \\
(w, q, a) &\vdash (wa', q', \Box) && \text{if } \Delta = +1 \\
(wb, q, ax) &\vdash (w, q', ba'x) && \text{if } \Delta = -1 \\
(\epsilon, q, ax) &\vdash (\epsilon, q', \Box a'x) && \text{if } \Delta = -1
\end{aligned}
$$

Turing Machine Transitions

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Yields relation

A configuration c of M yields a configuration c' of M, in symbols $c \vdash c'$, as defined by the following rules, where $a, a', b \in \Sigma_\Box$, $w, x \in \Sigma_\Box^*$, $q, q' \in Q$ and $((q, a), (q', a', \Delta)) \in \delta$:

$$
\begin{aligned}
(w, q, ax) &\vdash (wa', q', x) && \text{if } \Delta = +1, |x| \geq 1 \\
(w, q, a) &\vdash (wa', q', \Box) && \text{if } \Delta = +1 \\
(wb, q, ax) &\vdash (w, q', ba'x) && \text{if } \Delta = -1 \\
(\epsilon, q, ax) &\vdash (\epsilon, q', \Box a'x) && \text{if } \Delta = -1
\end{aligned}
$$

Acceptance (Time)

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Acceptance (time)

Let $c = (w, q, x)$ be a configuration of M.

- M accepts $c = (w, q, x)$ with $q \in Q_{\mathsf{Y}}$ in time n for all $n \in \mathbb{N}_0$.
- M accepts $c = (w, q, x)$ with $q \in Q_{\exists}$ in time n iff M accepts some c' with $c \vdash c'$ in time $n - 1$.
- M accepts $c = (w, q, x)$ with $q \in Q_{\forall}$ in time n iff M accepts all c' with $c \vdash c'$ in time $n - 1$.

Acceptance (Time)

Let $M = \langle \Sigma, \square, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Acceptance (time)

Let $c = (w, q, x)$ be a configuration of M.

- M accepts $c = (w, q, x)$ with $q \in Q_{\mathsf{Y}}$ in time n for all $n \in \mathbb{N}_0$.
- M accepts $c = (w, q, x)$ with $q \in Q_{\exists}$ in time n iff M accepts some c' with $c \vdash c'$ in time $n - 1$.
- M accepts $c = (w, q, x)$ with $q \in Q_{\forall}$ in time n iff M accepts all c' with $c \vdash c'$ in time $n - 1$.

Acceptance (Time)

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Acceptance (time)

Let $c = (w, q, x)$ be a configuration of M.

- M accepts $c = (w, q, x)$ with $q \in Q_\mathsf{Y}$ in time n for all $n \in \mathbb{N}_0$.
- M accepts $c = (w, q, x)$ with $q \in Q_\exists$ in time n iff M accepts some c' with $c \vdash c'$ in time $n - 1$.
- M accepts $c = (w, q, x)$ with $q \in Q_\forall$ in time n iff M accepts all c' with $c \vdash c'$ in time $n - 1$.

Acceptance (Space)

Let $M = \langle \Sigma, \square, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Acceptance (space)

Let $c = (w, q, x)$ be a configuration of M.

- M accepts $c = (w, q, x)$ with $q \in Q_{\mathsf{Y}}$ in space n iff $|w| + |x| \leq n$.
- M accepts $c = (w, q, x)$ with $q \in Q_{\exists}$ in space n iff M accepts some c' with $c \vdash c'$ in space n.
- M accepts $c = (w, q, x)$ with $q \in Q_{\forall}$ in space n iff M accepts all c' with $c \vdash c'$ in space n.

Acceptance (Space)

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Acceptance (space)

Let $c = (w, q, x)$ be a configuration of M.

- M accepts $c = (w, q, x)$ with $q \in Q_Y$ in space n iff $|w| + |x| \leq n$.
- M accepts $c = (w, q, x)$ with $q \in Q_\exists$ in space n iff M accepts some c' with $c \vdash c'$ in space n.
- M accepts $c = (w, q, x)$ with $q \in Q_\forall$ in space n iff M accepts all c' with $c \vdash c'$ in space n.

Acceptance (Space)

Let $M = \langle \Sigma, \square, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Acceptance (space)

Let $c = (w, q, x)$ be a configuration of M.

- M accepts $c = (w, q, x)$ with $q \in Q_{\mathsf{Y}}$ in space n iff $|w| + |x| \leq n$.
- M accepts $c = (w, q, x)$ with $q \in Q_{\exists}$ in space n iff M accepts some c' with $c \vdash c'$ in space n.
- M accepts $c = (w, q, x)$ with $q \in Q_{\forall}$ in space n iff M accepts all c' with $c \vdash c'$ in space n.

Accepting Words and Languages

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Accepting words

M accepts the word $w \in \Sigma^*$ in time (space) $n \in \mathbb{N}_0$
iff M accepts (ϵ, q_0, w) in time (space) n.

- Special case: M accepts ϵ in time (space) $n \in \mathbb{N}_0$
  iff M accepts $(\epsilon, q_0, \Box)$ in time (space) n.

Definition: Accepting languages

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$.
M accepts the language $L \subseteq \Sigma^*$ in time (space) f
iff M accepts each word $w \in L$ in time (space) $f(|w|)$,
and M does not accept any word $w \notin L$.

Accepting Words and Languages

Let $M = \langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ be an ATM.

Definition: Accepting words

M accepts the word $w \in \Sigma^*$ in time (space) $n \in \mathbb{N}_0$
iff M accepts (ϵ, q_0, w) in time (space) n.

- Special case: M accepts ϵ in time (space) $n \in \mathbb{N}_0$ iff M accepts $(\epsilon, q_0, \Box)$ in time (space) n.

Definition: Accepting languages

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$.
M accepts the language $L \subseteq \Sigma^*$ in time (space) f
iff M accepts each word $w \in L$ in time (space) $f(|w|)$,
and M does not accept any word $w \notin L$.

Time Complexity

Definition: DTIME, NTIME, ATIME

Let $f : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$.

Complexity class $\mathrm{DTIME}(f)$ contains all languages accepted in time f by some DTM.

Complexity class $\mathrm{NTIME}(f)$ contains all languages accepted in time f by some NTM.

Complexity class $\mathrm{ATIME}(f)$ contains all languages accepted in time f by some ATM.

Time Complexity

Definition: DTIME, NTIME, ATIME

Let $f : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$.

Complexity class DTIME(f) contains all languages accepted in time f by some DTM.

Complexity class NTIME(f) contains all languages accepted in time f by some NTM.

Complexity class ATIME(f) contains all languages accepted in time f by some ATM.

Time Complexity

Definition: DTIME, NTIME, ATIME

Let $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$.

Complexity class $\mathsf{DTIME}(f)$ contains all languages accepted in time f by some DTM.

Complexity class $\mathsf{NTIME}(f)$ contains all languages accepted in time f by some NTM.

Complexity class $\mathsf{ATIME}(f)$ contains all languages accepted in time f by some ATM.

Time Complexity

Definition: DTIME, NTIME, ATIME

Let $f : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$.

Complexity class DTIME(f) contains all languages accepted in time f by some DTM.

Complexity class NTIME(f) contains all languages accepted in time f by some NTM.

Complexity class ATIME(f) contains all languages accepted in time f by some ATM.

Space Complexity

Definition: DSPACE, NSPACE, ASPACE

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$.

Complexity class DSPACE(f) contains all languages accepted in space f by some DTM.

Complexity class NSPACE(f) contains all languages accepted in space f by some NTM.

Complexity class ASPACE(f) contains all languages accepted in space f by some ATM.

Space Complexity

Definition: DSPACE, NSPACE, ASPACE

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$.

Complexity class DSPACE(f) contains all languages accepted in space f by some DTM.

Complexity class NSPACE(f) contains all languages accepted in space f by some NTM.

Complexity class ASPACE(f) contains all languages accepted in space f by some ATM.

Space Complexity

Definition: DSPACE, NSPACE, ASPACE

Let $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$.

Complexity class DSPACE(f) contains all languages accepted in space f by some DTM.

Complexity class NSPACE(f) contains all languages accepted in space f by some NTM.

Complexity class ASPACE(f) contains all languages accepted in space f by some ATM.

Space Complexity

Definition: DSPACE, NSPACE, ASPACE

Let $f : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$.

Complexity class $\text{DSPACE}(f)$ contains all languages accepted in space f by some DTM.

Complexity class $\text{NSPACE}(f)$ contains all languages accepted in space f by some NTM.

Complexity class $\text{ASPACE}(f)$ contains all languages accepted in space f by some ATM.

Polynomial Complexity Classes

Let $\mathcal{P}$ be the set of polynomials $p : \mathbb{N}_0 \rightarrow \mathbb{N}_0$.

Definition: P, NP, ...

$$\begin{aligned}
\mathsf{P} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{DTIME}(p) \\
\mathsf{NP} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{NTIME}(p) \\
\mathsf{AP} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{ATIME}(p) \\
\mathsf{PSPACE} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{DSPACE}(p) \\
\mathsf{NPSPACE} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{NSPACE}(p) \\
\mathsf{APSPACE} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{ASPACE}(p)
\end{aligned}$$

Exponential Complexity Classes

Let $\mathcal{P}$ be the set of polynomials $p : \mathbb{N}_0 \to \mathbb{N}_0$.

Definition: EXP, NEXP, . . .

$$\begin{aligned}
\mathsf{EXP} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{DTIME}(2^p) \\
\mathsf{NEXP} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{NTIME}(2^p) \\
\mathsf{AEXP} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{ATIME}(2^p) \\
\mathsf{EXPSPACE} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{DSPACE}(2^p) \\
\mathsf{NEXPSPACE} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{NSPACE}(2^p) \\
\mathsf{AEXPSPACE} &= \textstyle\bigcup_{p \in \mathcal{P}} \mathsf{ASPACE}(2^p)
\end{aligned}$$

Doubly Exponential Complexity Classes

Let $\mathcal{P}$ be the set of polynomials $p : \mathbb{N}_0 \to \mathbb{N}_0$.

Definition: 2-EXP, . . .

$$\text{2-EXP} = \bigcup_{p \in \mathcal{P}} \text{DTIME}(2^{2^p})$$

$\dots$

Standard Complexity Classes Relationships

Theorem

$$
\begin{array}{rcccl}
\mathsf{P} & \subseteq & \mathsf{NP} & \subseteq & \mathsf{AP} \\
\mathsf{PSPACE} & \subseteq & \mathsf{NPSPACE} & \subseteq & \mathsf{APSPACE} \\
\mathsf{EXP} & \subseteq & \mathsf{NEXP} & \subseteq & \mathsf{AEXP} \\
\mathsf{EXPSPACE} & \subseteq & \mathsf{NEXPSPACE} & \subseteq & \mathsf{AEXPSPACE} \\
\mathsf{2\text{-}EXP} & \subseteq & \ldots & &
\end{array}
$$

The Power of Nondeterministic Space

Theorem (Savitch 1970)

$\mathsf{NSPACE}(f) \subseteq \mathsf{DSPACE}(f^2)$, and thus:

$$\begin{aligned} \mathsf{PSPACE} &= \mathsf{NPSPACE} \\ \mathsf{EXPSPACE} &= \mathsf{NEXPSPACE} \end{aligned}$$

The Power of Alternation

Theorem (Chandra et al. 1981)

$$\begin{aligned} \mathrm{AP} &= \mathrm{PSPACE} \\ \mathrm{APSPACE} &= \mathrm{EXP} \\ \mathrm{AEXP} &= \mathrm{EXPSPACE} \\ \mathrm{AEXPSPACE} &= \text{2-EXP} \end{aligned}$$

The Hierarchy of complexity classes

2-EXPSPACE = 2-NEXPSPACE

2-NEXP

2-EXP = AEXPSPACE

EXPSPACE = NEXPSPACE = AEXP

NEXP

EXP = APSPACE

PSPACE = NPSPACE = AP

NP

P

The Hierarchy of complexity classes

2-EXPSPACE= 2-NEXPSPACE

2-NEXP

2-EXP= AEXPSPACE

EXPSPACE= NEXPSPACE= AEXP

NEXP

EXP= APSPACE

PSPACE = NPSPACE = AP

NP

P

The Hierarchy of complexity classes

2-EXPSPACE= 2-NEXPSPACE

2-NEXP

2-EXP= AEXPSPACE

EXPSPACE= NEXPSPACE= AEXP

NEXP

EXP= APSPACE

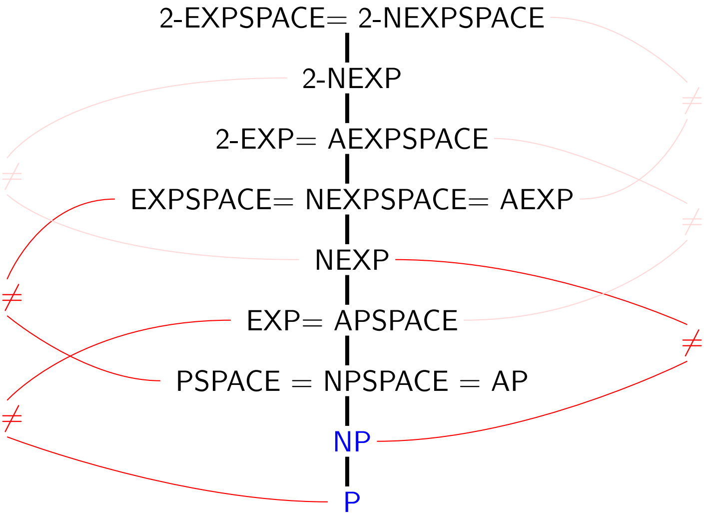
PSPACE = NPSPACE = AP

NP

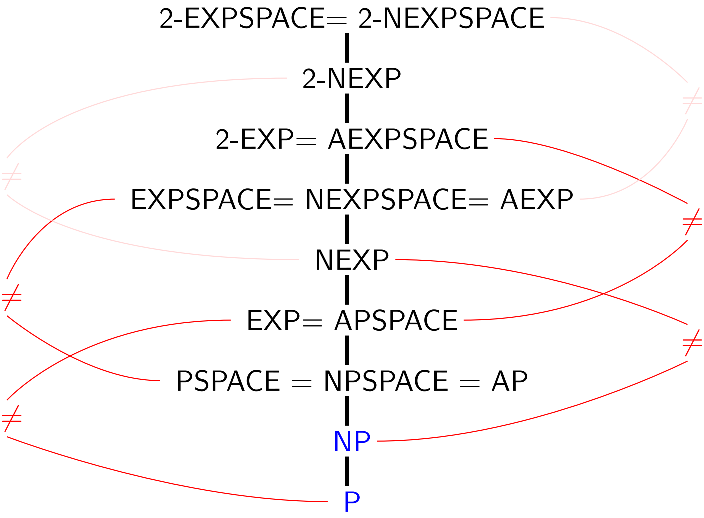
P

$\neq$ $\neq$ $\neq$ $\neq$ $\neq$ $\neq$

The Hierarchy of complexity classes



The Hierarchy of complexity classes



The Hierarchy of complexity classes

2-EXPSPACE= 2-NEXPSPACE

2-NEXP

2-EXP= AEXPSPACE

EXPSPACE= NEXPSPACE= AEXP

NEXP

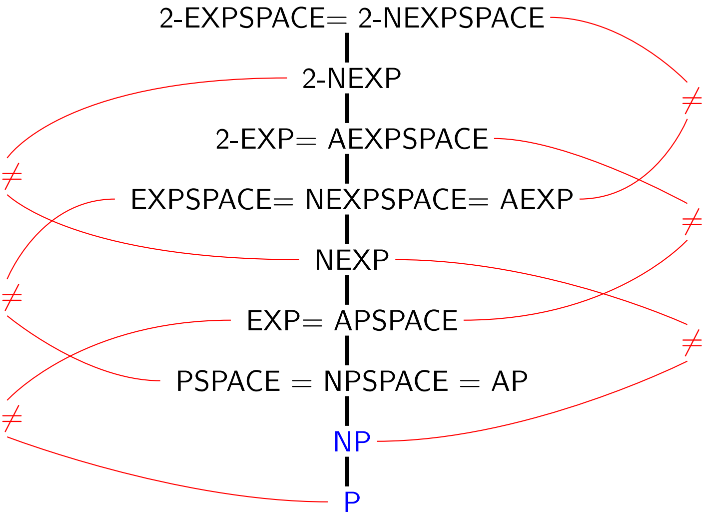
EXP= APSPACE

PSPACE = NPSPACE = AP

NP

P

The Hierarchy of complexity classes



The Planning Problem

PlanEx (Plan Existence)

Given: Planning task $\langle A, I, O, G\rangle$

Question: Is there a plan for $\langle A, I, O, G\rangle$?

PlanLen (Bounded Plan Existence)

Given: Planning task $\langle A, I, O, G\rangle$, bound $K \in \mathbb{N}_0$

Question: Is there a plan for $\langle A, I, O, G\rangle$ of length at most K?

Plan Existence vs. Bounded Plan Existence

PLANEX $\leq_{\mathsf{p}}$ PLANLEN

A planning task with n state variables has a plan
iff it has a plan of length at most $2^n - 1$.
$\leadsto$ polynomial reduction

Membership in PSPACE

PlanLen $\in$ PSPACE

Show PlanLen $\in$ NPSPACE and use Savitch's theorem.

Nondeterministic algorithm:

def plan($\langle A, I, O, G\rangle$, K):
  $s := I$
  $k := K$
  repeat until $s \models G$:
    guess $o \in O$
    reject if o not applicable in s
    set $s :=$ *app*$_o(s)$
    reject if $k = 0$
    set $k := k - 1$
  accept

Hardness for PSPACE

Idea: generic reduction

- For a fixed polynomial p, given DTM M and input w, generate planning task which is solvable iff M accepts w in space $p(|w|)$
- For simplicity, restrict to TMs which never move to the left of the initial head position (no loss of generality)

Hardness for PSPACE

Idea: generic reduction

- For a fixed polynomial p, given DTM M and input w, generate planning task which is solvable iff M accepts w in space $p(|w|)$
- For simplicity, restrict to TMs which never move to the left of the initial head position (no loss of generality)

Hardness for PSPACE

Idea: generic reduction

- For a fixed polynomial p, given DTM M and input w, generate planning task which is solvable iff M accepts w in space $p(|w|)$
- For simplicity, restrict to TMs which never move to the left of the initial head position (no loss of generality)

Reduction: State Variables

Let p be the space-bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

State variables

- state_q for all $q \in Q$
- head_i for all $i \in X \cup \{0, p(n) + 1\}$
- $\text{content}_{i,a}$ for all $i \in X$, $a \in \Sigma_\square$

Reduction: State Variables

Let p be the space-bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

State variables

- state$_q$ for all $q \in Q$
- head$_i$ for all $i \in X \cup \{0, p(n) + 1\}$
- content$_{i,a}$ for all $i \in X$, $a \in \Sigma_\square$

Reduction: State Variables

Let p be the space-bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

State variables

- state_q for all $q \in Q$
- head_i for all $i \in X \cup \{0, p(n) + 1\}$
- $\text{content}_{i,a}$ for all $i \in X$, $a \in \Sigma_\square$

Reduction: State Variables

Let p be the space-bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

State variables

- state_q for all $q \in Q$
- head_i for all $i \in X \cup \{0, p(n) + 1\}$
- $\text{content}_{i,a}$ for all $i \in X$, $a \in \Sigma_\Box$

Reduction: Initial State

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Initial state

Initially true:

- state_{q_0}
- head_1
- $\mathrm{content}_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- $\mathrm{content}_{i,\square}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Initial State

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Initial state

Initially true:

- state$_{q_0}$
- head$_1$
- content$_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- content$_{i,\Box}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Initial State

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Initial state

Initially true:

- state$_{q_0}$
- head$_1$
- content$_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- content$_{i,\square}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Initial State

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Initial state

Initially true:

- state_{q_0}
- head_1
- $\text{content}_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- $\text{content}_{i,\Box}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Initial State

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Initial state

Initially true:

- state_{q_0}
- head_1
- $\text{content}_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- $\text{content}_{i,\square}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Initial State

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Initial state

Initially true:

- state_{q_0}
- head_1
- $\text{content}_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- $\text{content}_{i,\Box}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Operators

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Operators

One operator for each transition rule $\delta(q, a) = (q', a', \Delta)$ and each cell position $i \in X$:

- precondition: $\text{state}_q \wedge \text{head}_i \wedge \text{content}_{i,a}$
- effect: $\neg\text{state}_q \wedge \neg\text{head}_i \wedge \neg\text{content}_{i,a}$
  $\wedge\, \text{state}_{q'} \wedge \text{head}_{i+\Delta} \wedge \text{content}_{i,a'}$

Reduction: Operators

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Operators

One operator for each transition rule $\delta(q, a) = (q', a', \Delta)$ and each cell position $i \in X$:

- precondition: $\text{state}_q \land \text{head}_i \land \text{content}_{i,a}$
- effect: $\neg\text{state}_q \land \neg\text{head}_i \land \neg\text{content}_{i,a}$
  $\land\, \text{state}_{q'} \land \text{head}_{i+\Delta} \land \text{content}_{i,a'}$

Reduction: Operators

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Operators

One operator for each transition rule $\delta(q, a) = (q', a', \Delta)$ and each cell position $i \in X$:

- precondition: $\text{state}_q \wedge \text{head}_i \wedge \text{content}_{i,a}$
- effect: $\neg\text{state}_q \wedge \neg\text{head}_i \wedge \neg\text{content}_{i,a}$ $\wedge\, \text{state}_{q'} \wedge \text{head}_{i+\Delta} \wedge \text{content}_{i,a'}$

Reduction: Operators

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Operators

One operator for each transition rule $\delta(q, a) = (q', a', \Delta)$ and each cell position $i \in X$:

- precondition: $\mathsf{state}_q \wedge \mathsf{head}_i \wedge \mathsf{content}_{i,a}$
- effect: $\neg\mathsf{state}_q \wedge \neg\mathsf{head}_i \wedge \neg\mathsf{content}_{i,a}$
  $\wedge\, \mathsf{state}_{q'} \wedge \mathsf{head}_{i+\Delta} \wedge \mathsf{content}_{i,a'}$

Reduction: Goal

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \square, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Goal

$\bigvee_{q \in Q_Y} \mathsf{state}_q$

Reduction: Goal

Let p be the space bound polynomial.

Given DTM $\langle \Sigma, \Box, Q, q_0, l, \delta \rangle$ and input $w_1 \dots w_n$, define relevant tape positions $X = \{1, \dots, p(n)\}$.

Goal

$\bigvee_{q \in Q_\mathsf{Y}} \mathsf{state}_q$

PSPACE-completeness for STRIPS

Theorem (PSPACE-completeness (Bylander))

PlanEx *and* PlanLen *are PSPACE-complete even if the planning task is given in STRIPS form (preconditions and goals are conjunctions of literals and no conditional effects).*

Proof.

Hardness and membership for the general formalism follows from the above. Hardness holds for STRIPS as well because of the style of the reduction: only simple preconditions and no conditional effects. The only problem is the disjunction in the goal formula. This can be eliminated by transforming the TM beforehand, though. □

PSPACE-completeness for STRIPS

Theorem (PSPACE-completeness (Bylander))

PlanEx *and* PlanLen *are PSPACE-complete even if the planning task is given in STRIPS form (preconditions and goals are conjunctions of literals and no conditional effects).*

Proof.

Hardness and membership for the general formalism follows from the above. Hardness holds for STRIPS as well because of the style of the reduction: only simple preconditions and no conditional effects. The only problem is the disjunction in the goal formula. This can be eliminated by transforming the TM beforehand, though. □

PSPACE-completeness for STRIPS

Theorem (PSPACE-completeness (Bylander))

PLANEX *and* PLANLEN *are PSPACE-complete even if the planning task is given in STRIPS form (preconditions and goals are conjunctions of literals and no conditional effects).*

Proof.

Hardness and membership for the general formalism follows from the above. Hardness holds for STRIPS as well because of the style of the reduction: only simple preconditions and no conditional effects. The only problem is the disjunction in the goal formula. This can be eliminated by transforming the TM beforehand, though. □

PSPACE-completeness for STRIPS

Theorem (PSPACE-completeness (Bylander))

PlanEx and PlanLen are PSPACE-complete even if the planning task is given in STRIPS form (preconditions and goals are conjunctions of literals and no conditional effects).

Proof.

Hardness and membership for the general formalism follows from the above. Hardness holds for STRIPS as well because of the style of the reduction: only simple preconditions and no conditional effects. The only problem is the disjunction in the goal formula. This can be eliminated by transforming the TM beforehand, though. □

Polynomial plan size

The PSPACE result depends on the fact that plans can become exponentially long. What if restrict them to be only of polynomial length?

Theorem (NP-completeness for polynomial plan size)

PLANEX and PLANLEN are NP-complete even if the planning task is given in STRIPS form provided only plans of length polynomial in the size of the planning task are permitted.

Membership follows easily by using a guess-and-check algorithm.

Hardness needs a bit more ...

Note: Answers earlier questions whether we need the Boolean precondition formula for NP-hardness

Polynomial plan size

The PSPACE result depends on the fact that plans can become exponentially long. What if restrict them to be only of polynomial length?

Theorem (NP-completeness for polynomial plan size)

PLANEX *and* PLANLEN *are NP-complete even if the planning task is given in STRIPS form provided only plans of length polynomial in the size of the planning task are permitted.*

Membership follows easily by using a guess-and-check algorithm.

Hardness needs a bit more ...

Note: Answers earlier questions whether we need the Boolean precondition formula for NP-hardness

Polynomial plan size

The PSPACE result depends on the fact that plans can become exponentially long. What if restrict them to be only of polynomial length?

Theorem (NP-completeness for polynomial plan size)

PLANEX and PLANLEN are NP-complete even if the planning task is given in STRIPS form provided only plans of length polynomial in the size of the planning task are permitted.

Membership follows easily by using a guess-and-check algorithm.

Hardness needs a bit more ...

Note: Answers earlier questions whether we need the Boolean precondition formula for NP-hardness

Polynomial plan size

The PSPACE result depends on the fact that plans can become exponentially long. What if restrict them to be only of polynomial length?

Theorem (NP-completeness for polynomial plan size)

PLANEX *and* PLANLEN *are NP-complete even if the planning task is given in STRIPS form provided only plans of length polynomial in the size of the planning task are permitted.*

Membership follows easily by using a guess-and-check algorithm.

Hardness needs a bit more ...

Note: Answers earlier questions whether we need the Boolean precondition formula for NP-hardness

Polynomial plan size

The PSPACE result depends on the fact that plans can become exponentially long. What if restrict them to be only of polynomial length?

Theorem (NP-completeness for polynomial plan size)

PLANEX *and* PLANLEN *are NP-complete even if the planning task is given in STRIPS form provided only plans of length polynomial in the size of the planning task are permitted.*

Membership follows easily by using a guess-and-check algorithm.

Hardness needs a bit more ...

Note: Answers earlier questions whether we need the Boolean precondition formula for NP-hardness

First-Order Tasks

- we considered
  propositional state variables (0-ary predicates) and
  grounded operators (0-ary schematic operators)
- reasonable: most planning algorithms
  work on grounded representations
- predicate arity is typically small (a constant?)

How do the complexity results change if we introduce first-order predicates and schematic operators?

First-Order Tasks

- we considered
  propositional state variables (0-ary predicates) and
  grounded operators (0-ary schematic operators)
- reasonable: most planning algorithms
  work on grounded representations
- predicate arity is typically small (a constant?)

How do the complexity results change if we introduce first-order predicates and schematic operators?

Membership in EXPSPACE

$\textsc{PlanEx}, \textsc{PlanLen} \in$ EXPSPACE

- input size n
- $\rightsquigarrow$ at most 2^n grounded state variables
- $\rightsquigarrow$ at most 2^n grounded operators
- can ground the task in exponential time, then use the earlier PSPACE algorithms

Hardness for EXPSPACE

Idea: Adapt the earlier reduction from PLANEX to encode Turing Machine contents more succinctly. Assume relevant tape positions are now $X = \{1, \dots, 2^n\}$. We need to encode the computation as a planning task in polynomial time!

Objects

0, 1

Predicates

- $\mathrm{state}_q()$ for all $q \in Q$
- $\mathrm{head}(?b_1, \dots, ?b_n)$
- $\mathrm{content}_a(?b_1, \dots, ?b_n)$ for all $a \in \Sigma_\square$

Hardness for EXPSPACE

Idea: Adapt the earlier reduction from PLANEX to encode Turing Machine contents more succinctly. Assume relevant tape positions are now $X = \{1, \dots, 2^n\}$. We need to encode the computation as a planning task in polynomial time!

Objects

0, 1

Predicates

- state$_q$() for all $q \in Q$
- head$(?b_1, \dots, ?b_n)$
- content$_a(?b_1, \dots, ?b_n)$ for all $a \in \Sigma_\square$

Hardness for EXPSPACE

Idea: Adapt the earlier reduction from PLANEX to encode Turing Machine contents more succinctly. Assume relevant tape positions are now $X = \{1, \dots, 2^n\}$. We need to encode the computation as a planning task in polynomial time!

Objects

$0, 1$

Predicates

- $\mathrm{state}_q()$ for all $q \in Q$
- $\mathrm{head}(?b_1, \dots, ?b_n)$
- $\mathrm{content}_a(?b_1, \dots, ?b_n)$ for all $a \in \Sigma_\square$

Hardness for EXPSPACE

Idea: Adapt the earlier reduction from PLANEX to encode Turing Machine contents more succinctly. Assume relevant tape positions are now $X = \{1, \ldots, 2^n\}$. We need to encode the computation as a planning task in polynomial time!

Objects

$0, 1$

Predicates

- $\text{state}_q()$ for all $q \in Q$
- $\text{head}(?b_1, \ldots, ?b_n)$
- $\text{content}_a(?b_1, \ldots, ?b_n)$ for all $a \in \Sigma_\square$

Hardness for EXPSPACE

Idea: Adapt the earlier reduction from PLANEX to encode Turing Machine contents more succinctly. Assume relevant tape positions are now $X = \{1, \dots, 2^n\}$. We need to encode the computation as a planning task in polynomial time!

Objects

0, 1

Predicates

- state$_q()$ for all $q \in Q$
- head$(?b_1, \dots, ?b_n)$
- content$_a(?b_1, \dots, ?b_n)$ for all $a \in \Sigma_\square$

Reduction: Example Operator

Operator example

Schematic operator for transition rule $\delta(q, a) = (q', a', +1)$

- parameters: $?b_1, \dots, ?b_n$
- precondition:

  state_q
  $\wedge\, \mathsf{head}(?b_1, \dots, ?b_n)$
  $\wedge\, \mathsf{content}_a(?b_1, \dots, ?b_n)$
- effect:

  $\neg\mathsf{state}_q$
  $\wedge\, \neg\mathsf{head}(?b_1, \dots, ?b_n)$
  $\wedge\, \neg\mathsf{content}_a(?b_1, \dots, ?b_n)$
  $\wedge\, \mathsf{state}_{q'}$
  $\wedge$ advance-head
  $\wedge\, \mathsf{content}_{a'}(?b_1, \dots, ?b_n)$

Reduction: Example Operator (continued)

Operator example (ctd.)

$$
\begin{aligned}
\textsf{advance-head} = {} & ((?b_n = 0) \\
& \quad \rhd \textsf{head}(?b_1, \dots, ?b_{n-1}, 1)) \\
& \wedge ((?b_{n-1} = 0 \wedge ?b_n = 1) \\
& \quad \rhd \textsf{head}(?b_1, \dots, ?b_{n-2}, 1, 0)) \\
& \wedge ((?b_{n-2} = 0 \wedge ?b_{n-1} = 1 \wedge ?b_n = 1) \\
& \quad \rhd \textsf{head}(?b_1, \dots, ?b_{n-3}, 1, 0, 0)) \\
& \wedge \dots \\
& \wedge ((?b_1 = 0 \wedge ?b_2 = 1 \wedge \dots \wedge ?b_n = 1) \\
& \quad \rhd \textsf{head}(1, 0, \dots, 0))
\end{aligned}
$$

Plan Existence vs. Bounded Plan Existence

- Our earlier reduction from PlanEx to PlanLen no longer works: the shortest plan can have length doubly exponentially in the input size, so that the bound cannot be written down in polynomial time.
- Indeed, PlanLen is actually easier than PlanEx for this planning formalism (NEXP-complete).

Plan Existence vs. Bounded Plan Existence

- Our earlier reduction from PLANEX to PLANLEN no longer works: the shortest plan can have length doubly exponentially in the input size, so that the bound cannot be written down in polynomial time.
- Indeed, PLANLEN is actually easier than PLANEX for this planning formalism (NEXP-complete).

Planning with function terms

- If we allow in addition function terms with arity > 0, then planning becomes undecidable.
- The state space is infinite: $s(0), s(s(0)), s(s(s(0))), \ldots$
- We can use function terms to describe (the index of) tape cells of a Turing machine.
- We can use operators to describe the Turing machine control.
- The existence of a plan is then equivalent to the existence of a successful computation on the Turing machine.
- PLANEX for planning tasks with function terms can be used to decide the Halting problem.

Theorem

PLANEX *for planning tasks with function terms is undecidable.*

Domain-Dependent Planning

Planning (and its complexity) for particular domains is interesting, since we want to judge planning benchmarks

. . . and perhaps want to go for domain-dependent planning.

Consider fixed domains and determine complexity.

Domain-Dependent Planning

Planning (and its complexity) for particular domains is interesting, since we want to judge planning benchmarks

. . . and perhaps want to go for domain-dependent planning.

Consider fixed domains and determine complexity.

Domain-Dependent Planning

Planning (and its complexity) for particular domains is interesting, since we want to judge planning benchmarks

. . . and perhaps want to go for domain-dependent planning.

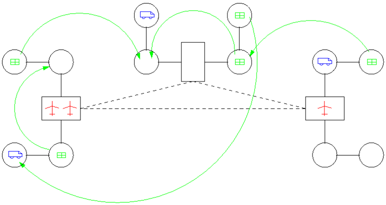
Consider fixed domains and determine complexity.

A Concrete Domain: Logistics

There are several cities, each containing several locations, some of which are airports. There are also trucks, which drive within a single city, and airplanes, which can fly between airports. The goal is to get some packages from various locations to various new locations [McDermott, 1998].

A Concrete Domain: Logistics

There are several cities, each containing several locations, some of which are airports. There are also trucks, which drive within a single city, and airplanes, which can fly between airports. The goal is to get some packages from various locations to various new locations [McDermott, 1998].



Plan Existence for Logistics

Theorem

PLANEX for Logistics can be decided in polynomial time.

Proof.

Consider the subgraphs formed by the connected airport networks (for planes) and city networks (for trucks). If at least one vehicle (truck or plane) is in one of the subgraphs, all nodes in the subgraph are internally reachable, otherwise only the externally connected nodes can be reached. Check for each package delivery, whether there are connected subgraphs such that the package can pass through the subgraphs to the target node. This is a simple reachability test, which can be done in poly. time. □

Plan Existence for Logistics

Theorem

PLANEX for Logistics can be decided in polynomial time.

Proof.

Consider the subgraphs formed by the connected airport networks (for planes) and city networks (for trucks). If at least one vehicle (truck or plane) is in one of the subgraphs, all nodes in the subgraph are internally reachable, otherwise only the externally connected nodes can be reached. Check for each package delivery, whether there are connected subgraphs such that the package can pass through the subgraphs to the target node. This is a simple reachability test, which can be done in poly. time. □

Plan Existence for Logistics

Theorem

PLANEX for Logistics can be decided in polynomial time.

Proof.

Consider the subgraphs formed by the connected airport networks (for planes) and city networks (for trucks). If at least one vehicle (truck or plane) is in one of the subgraphs, all nodes in the subgraph are internally reachable, otherwise only the externally connected nodes can be reached. Check for each package delivery, whether there are connected subgraphs such that the package can pass through the subgraphs to the target node. This is a simple reachability test, which can be done in poly. time. □

Plan Existence for Logistics

Theorem

PLANEX for Logistics can be decided in polynomial time.

Proof.

Consider the subgraphs formed by the connected airport networks (for planes) and city networks (for trucks). If at least one vehicle (truck or plane) is in one of the subgraphs, all nodes in the subgraph are internally reachable, otherwise only the externally connected nodes can be reached. Check for each package delivery, whether there are connected subgraphs such that the package can pass through the subgraphs to the target node. This is a simple reachability test, which can be done in poly. time. □

Plan Existence for Logistics

Theorem

PLANEX for Logistics can be decided in polynomial time.

Proof.

Consider the subgraphs formed by the connected airport networks (for planes) and city networks (for trucks). If at least one vehicle (truck or plane) is in one of the subgraphs, all nodes in the subgraph are internally reachable, otherwise only the externally connected nodes can be reached. Check for each package delivery, whether there are connected subgraphs such that the package can pass through the subgraphs to the target node. This is a simple reachability test, which can be done in poly. time. □

Optimizing Delivery: Shortest Plans for Logistics

Definition (Feedback Vertex Set)

- Given: a directed graph $G = (V, A)$ and a natural number k
- Question: Does there exists a subset $V' \subseteq V$ with $|V'| \leq k$ such that removing V' results in an acyclic graph?

This problem is NP-complete and can be used to prove the following result:

Theorem

PLANLEN for Logistics is NP-complete, even if there is only one complete city graph and one truck in this graph.

Proof.

Membership follows because there is an obvious polynomial upper bound of moves for all solvable instances.
Hardness is shown using a reduction from FVS.

Optimizing Delivery: Shortest Plans for Logistics

Definition (Feedback Vertex Set)

- Given: a directed graph $G = (V, A)$ and a natural number k
- Question: Does there exists a subset $V' \subseteq V$ with $|V'| \leq k$ such that removing V' results in an acyclic graph?

This problem is NP-complete and can be used to prove the following result:

Theorem

PLANLEN for Logistics is NP-complete, even if there is only one complete city graph and one truck in this graph.

Proof.

Membership follows because there is an obvious polynomial upper bound of moves for all solvable instances.
Hardness is shown using a reduction from FVS.

Optimizing Delivery: Shortest Plans for Logistics

Definition (Feedback Vertex Set)

- Given: a directed graph $G = (V, A)$ and a natural number k
- Question: Does there exists a subset $V' \subseteq V$ with $|V'| \leq k$ such that removing V' results in an acyclic graph?

This problem is NP-complete and can be used to prove the following result:

Theorem

PLANLEN for Logistics is NP-complete, even if there is only one complete city graph and one truck in this graph.

Proof.

Membership follows because there is an obvious polynomial upper bound of moves for all solvable instances.

Hardness is shown using a reduction from FVS.

Optimizing Delivery: Shortest Plans for Logistics

Definition (Feedback Vertex Set)

- Given: a directed graph $G = (V, A)$ and a natural number k
- Question: Does there exists a subset $V' \subseteq V$ with $|V'| \leq k$ such that removing V' results in an acyclic graph?

This problem is NP-complete and can be used to prove the following result:

Theorem

PLANLEN for Logistics is NP-complete, even if there is only one complete city graph and one truck in this graph.

Proof.

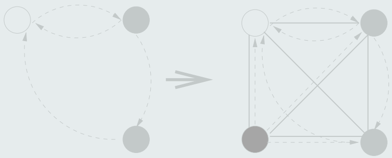
Membership follows because there is an obvious polynomial upper bound of moves for all solvable instances.
Hardness is shown using a reduction from FVS.

PLANLEN for Logistics: NP-hardness contd. (1)

Proof. (continued)

Let $G = (V, A)$ be a directed graph and k a natural number. Then G contains a FVS of size k iff the logistics problem constructed below has a plan of length at most $3|V| + 2|A| + k$.

Construct a Logistics task with just one truck and one city network which is a complete graph containing V and an extra node v_0, where the truck starts. The truck has to deliver one package from v_0 to each other location and one package from u to v for each $(u, v) \in A$.

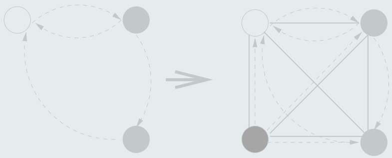


PLANLEN for Logistics: NP-hardness contd. (1)

Proof. (continued)

Let $G = (V, A)$ be a directed graph and k a natural number. Then G contains a FVS of size k iff the logistics problem constructed below has a plan of length at most $3|V| + 2|A| + k$.

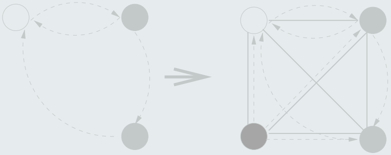
Construct a Logistics task with just one truck and one city network which is a complete graph containing V and an extra node v_0, where the truck starts. The truck has to deliver one package from v_0 to each other location and one package from u to v for each $(u, v) \in A$.



PLANLEN for Logistics: NP-hardness contd. (1)

Proof. (continued)

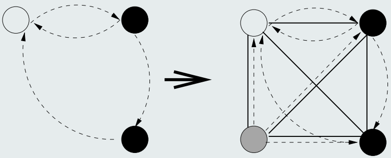
Let $G = (V, A)$ be a directed graph and k a natural number. Then G contains a FVS of size k iff the logistics problem constructed below has a plan of length at most $3|V| + 2|A| + k$. Construct a Logistics task with just one truck and one city network which is a complete graph containing V and an extra node v_0, where the truck starts. The truck has to deliver one package from v_0 to each other location and one package from u to v for each $(u, v) \in A$.



PLANLEN for Logistics: NP-hardness contd. (1)

Proof. (continued)

Let $G = (V, A)$ be a directed graph and k a natural number. Then G contains a FVS of size k iff the logistics problem constructed below has a plan of length at most $3|V| + 2|A| + k$. Construct a Logistics task with just one truck and one city network which is a complete graph containing V and an extra node v_0, where the truck starts. The truck has to deliver one package from v_0 to each other location and one package from u to v for each $(u, v) \in A$.



PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$.

Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$. Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$. Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$.
Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$.

Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$.
Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$. Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$.
Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

PLANLEN for Logistics: NP-hardness contd. (2)

Proof. (continued).

Let $V' \subseteq V$ the feedback vertex set. Solve task by moving to V' in any order, then to all $V - V'$ using a topological ordering on these nodes, and finally to V' again. Requires $|A| + |V|$ load and unload actions each, and $|V'| + |V - V'| + |V'|$ movements, i.e., $3|V| + 2|A| + k$ actions if $|V'| = k$.
Conversely, at least $3|V| + 2|A|$ actions are needed. If a plan contains not more than $3|V| + 2|A| + k$, then no more than k nodes are visited twice. These nodes form a FVS of size k. □

Note: It is not route planning that makes the task difficult, but the interaction of sub-goals!

Other Domains

- Generalizations of all the domains that have been used at the international planning competition have been analyzed.
- Many show a similar behavior as Logistics: PLANEX is in P, PLANLEN is NP-complete, e.g., Blocks world.
- Some are already NP-complete for PLANEX, e.g. Freecell.
- A few are even PSPACE-complete for PLANEX, e.g. Airport.

Other Domains

- Generalizations of all the domains that have been used at the international planning competition have been analyzed.
- Many show a similar behavior as Logistics: PLANEX is in P, PLANLEN is NP-complete, e.g., Blocks world.
- Some are already NP-complete for PLANEX, e.g. Freecell.
- A few are even PSPACE-complete for PLANEX, e.g. Airport.

Other Domains

- Generalizations of all the domains that have been used at the international planning competition have been analyzed.
- Many show a similar behavior as Logistics: PLANEX is in P, PLANLEN is NP-complete, e.g., Blocks world.
- Some are already NP-complete for PLANEX, e.g. Freecell.
- A few are even PSPACE-complete for PLANEX, e.g. Airport.

Other Domains

- Generalizations of all the domains that have been used at the international planning competition have been analyzed.
- Many show a similar behavior as Logistics: PLANEX is in P, PLANLEN is NP-complete, e.g., Blocks world.
- Some are already NP-complete for PLANEX, e.g. Freecell.
- A few are even PSPACE-complete for PLANEX, e.g. Airport.

Summary

- Planning using general first-order terms is undecidable.
- Planning using a function free language is EXPSPACE-complete.
- Planning with a propositional language (no schema variables) is PSPACE-complete.
- If we consider only "short" plans, the complexity comes down to NP-completeness.
- Domain-dependent planning can be easier.
- For Logistics, the existence problem is in P, while the optimization problem is NP-complete, which holds for many other domains as well.

Summary

- Planning using general first-order terms is undecidable.
- Planning using a function free language is EXPSPACE-complete.
- Planning with a propositional language (no schema variables) is PSPACE-complete.
- If we consider only "short" plans, the complexity comes down to NP-completeness.
- Domain-dependent planning can be easier.
- For Logistics, the existence problem is in P, while the optimization problem is NP-complete, which holds for many other domains as well.

Summary

- Planning using general first-order terms is undecidable.
- Planning using a function free language is EXPSPACE-complete.
- Planning with a propositional language (no schema variables) is PSPACE-complete.
- If we consider only "short" plans, the complexity comes down to NP-completeness.
- Domain-dependent planning can be easier.
- For Logistics, the existence problem is in P, while the optimization problem is NP-complete, which holds for many other domains as well.

Summary

- Planning using general first-order terms is undecidable.
- Planning using a function free language is EXPSPACE-complete.
- Planning with a propositional language (no schema variables) is PSPACE-complete.
- If we consider only "short" plans, the complexity comes down to NP-completeness.
- Domain-dependent planning can be easier.
- For Logistics, the existence problem is in P, while the optimization problem is NP-complete, which holds for many other domains as well.

Summary

- Planning using general first-order terms is undecidable.
- Planning using a function free language is EXPSPACE-complete.
- Planning with a propositional language (no schema variables) is PSPACE-complete.
- If we consider only "short" plans, the complexity comes down to NP-completeness.
- Domain-dependent planning can be easier.
- For Logistics, the existence problem is in P, while the optimization problem is NP-complete, which holds for many other domains as well.

Summary

- Planning using general first-order terms is undecidable.
- Planning using a function free language is EXPSPACE-complete.
- Planning with a propositional language (no schema variables) is PSPACE-complete.
- If we consider only "short" plans, the complexity comes down to NP-completeness.
- Domain-dependent planning can be easier.
- For Logistics, the existence problem is in P, while the optimization problem is NP-complete, which holds for many other domains as well.