Lecture *Hierarchical Planning*

Chapter:
Expressivity Analysis of Planning Formalisms

Dr. Pascal Bercher

Institute of Artificial Intelligence,
Ulm University, Germany

Winter Term 2018/2019

(Compiled on: February 19, 2019)

ulm university universität uulm

Overview:

Motivation

- Given a planning problem with a certain set of constraints,

Motivation

- Given a planning problem with a certain set of constraints, how to decide which planning formalism to choose?

Motivation

- Given a planning problem with a certain set of constraints, how to decide which planning formalism to choose?
- We need to know the influence of formalization choices and solution criteria on the possible solutions.

Motivation

- Given a planning problem with a certain set of constraints, how to decide which planning formalism to choose?
- We need to know the influence of formalization choices and solution criteria on the possible solutions.

$\rightarrow$ Expressivity Analysis: Which *structural* properties may solutions have?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?
- Hierarchical planning?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?
- Hierarchical planning? Under which restrictions?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?
- Hierarchical planning? Under which restrictions?
  - With or without task insertion?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?
- Hierarchical planning? Under which restrictions?
  - With or without task insertion?
  - With or without conditional effects?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?
- Hierarchical planning? Under which restrictions?
  - With or without task insertion?
  - With or without conditional effects?
  - With limited recursion?

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

- Γ, a finite set of non-terminal symbols.

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

- Γ, a finite set of non-terminal symbols.
- Σ, a finite set of terminal symbols.

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

- Γ, a finite set of non-terminal symbols.
- Σ, a finite set of terminal symbols.
- $R \subseteq (\Sigma \cup \Gamma)^*\Gamma(\Sigma \cup \Gamma)^* \times (\Sigma \cup \Gamma)^*$, a finite set of production rules.

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

- Γ, a finite set of non-terminal symbols.
- Σ, a finite set of terminal symbols.
- $R \subseteq (\Sigma \cup \Gamma)^*\Gamma(\Sigma \cup \Gamma)^* \times (\Sigma \cup \Gamma)^*$, a finite set of production rules.
- $S \in \Gamma$, the start symbol.

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

- Γ, a finite set of non-terminal symbols.
- Σ, a finite set of terminal symbols.
- $R \subseteq (\Sigma \cup \Gamma)^*\Gamma(\Sigma \cup \Gamma)^* \times (\Sigma \cup \Gamma)^*$, a finite set of production rules.
- $S \in \Gamma$, the start symbol.

A *word* is a sequence of terminal-symbols $\omega \in \Sigma^*$.

Formal Grammars

Definition (Formal Grammars)

A *formal grammar* is a tuple $G = (\Gamma, \Sigma, R, S)$ consisting of:

- Γ, a finite set of non-terminal symbols.
- Σ, a finite set of terminal symbols.
- $R \subseteq (\Sigma \cup \Gamma)^*\Gamma(\Sigma \cup \Gamma)^* \times (\Sigma \cup \Gamma)^*$, a finite set of production rules.
- $S \in \Gamma$, the start symbol.

A *word* is a sequence of terminal-symbols $\omega \in \Sigma^*$.

The *language* of a grammar, $L(G)$, is the set of words that can be obtained from G's start symbol by applying a sequence of G's production rules.

Formal Grammars, Example

Let $G = (\Gamma, \Sigma, R, S)$ with $\Gamma = \{S, A, B\}$, $\Sigma = \{a, b\}$, and R given by:

- $S \rightarrow aA$
- $A \rightarrow aA$
- $A \rightarrow bB$
- $B \rightarrow bB$
- $B \rightarrow \varepsilon$

Formal Grammars, Example

Let $G = (\Gamma, \Sigma, R, S)$ with $\Gamma = \{S, A, B\}$, $\Sigma = \{a, b\}$, and R given by:

- $S \to aA$
- $A \to aA$
- $A \to bB$
- $B \to bB$
- $B \to \varepsilon$

Question: What is the language of the grammar?

Formal Grammars, Example

Let $G = (\Gamma, \Sigma, R, S)$ with $\Gamma = \{S, A, B\}$, $\Sigma = \{a, b\}$, and R given by:

- $S \rightarrow aA$
- $A \rightarrow aA$
- $A \rightarrow bB$
- $B \rightarrow bB$
- $B \rightarrow \varepsilon$

Question: What is the language of the grammar?

$L(G) = \{a^n b^m \mid n, m \geq 1\}$

Chomsky Hierarchy

Chomsky Hierarchy, ordered from most to least expressive:

Type 0 Unrestricted grammars.

Chomsky Hierarchy

Chomsky Hierarchy, ordered from most to least expressive:

Type 0 Unrestricted grammars.

Type 1 Context-sensitive grammars.

Chomsky Hierarchy

Chomsky Hierarchy, ordered from most to least expressive:

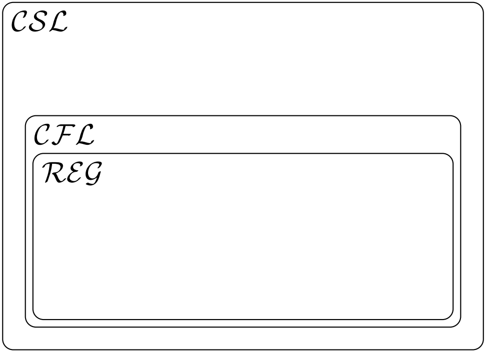
- Type 0 Unrestricted grammars.
- Type 1 Context-sensitive grammars.
- Type 2 Context-free grammars.

Chomsky Hierarchy

Chomsky Hierarchy, ordered from most to least expressive:

- Type 0 Unrestricted grammars.
- Type 1 Context-sensitive grammars.
- Type 2 Context-free grammars.
- Type 3 Regular grammars.

Expressivity via Comparison to Formal Languages



Regular Grammars

Definition:

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:
  - A single terminal symbol.

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:
  - A single terminal symbol.
  - The empty string (ε).

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:
  - A single terminal symbol.
  - The empty string (ε).
  - a terminal symbol followed by a non-terminal or the other way round. These can not be mixed! The one is called *right regular*, the other one is called *left regular*.

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:
  - A single terminal symbol.
  - The empty string (ε).
  - a terminal symbol followed by a non-terminal or the other way round. These can not be mixed! The one is called *right regular*, the other one is called *left regular*.

Properties:

- All finite languages are regular. (But not the other way round.)

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:
  - A single terminal symbol.
  - The empty string (ε).
  - a terminal symbol followed by a non-terminal or the other way round. These can not be mixed! The one is called *right regular*, the other one is called *left regular*.

Properties:

- All finite languages are regular. (But not the other way round.)
- There is an equivalent definition based on DFAs.

Regular Grammars

Definition:

- Regular grammars may only have a single non-terminal symbol as head in the production rules.
- Production rules' right-hand side may only be one of the following three forms:
  - A single terminal symbol.
  - The empty string (ε).
  - a terminal symbol followed by a non-terminal or the other way round. These can not be mixed! The one is called *right regular*, the other one is called *left regular*.

Properties:

- All finite languages are regular. (But not the other way round.)
- There is an equivalent definition based on DFAs.
- Do you know "regular expressions"?

Context-free Grammars

Definition:

Context-free Grammars

Definition:

- The head of each production rule consists of exactly one non-terminal symbol.

Context-free Grammars

Definition:

- The head of each production rule consists of exactly one non-terminal symbol.

Properties:

- Closed under intersection against any regular language.

Context-free Grammars

Definition:

- The head of each production rule consists of exactly one non-terminal symbol.

Properties:

- Closed under intersection against any regular language.
- The language intersection problem for two context-free grammars is undecidable. (Cf. p.202, thm. 8.10. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979)

Context-free Grammars

Definition:

- The head of each production rule consists of exactly one non-terminal symbol.

Properties:

- Closed under intersection against any regular language.
- The language intersection problem for two context-free grammars is undecidable. (Cf. p.202, thm. 8.10. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979)
- Given a context-free grammar, deciding whether it describes a regular language is undecidable. (Cf. p.281 of John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979)

Context-sensitive Grammars

Definition:

Context-sensitive Grammars

Definition:

- Each production rule has the form $\alpha X \beta \rightarrow \alpha \gamma \beta$ or $S \rightarrow \gamma$, where:

Context-sensitive Grammars

Definition:

- Each production rule has the form $\alpha X \beta \rightarrow \alpha \gamma \beta$ or $S \rightarrow \gamma$, where:
  - X is a non-terminal symbol.

Context-sensitive Grammars

Definition:

- Each production rule has the form $\alpha X\beta \rightarrow \alpha\gamma\beta$ or $S \rightarrow \gamma$, where:
  - X is a non-terminal symbol.
  - $\alpha, \beta \in (\Gamma \cup \Sigma)^*$.

Context-sensitive Grammars

Definition:

- Each production rule has the form $\alpha X \beta \to \alpha\gamma\beta$ or $S \to \gamma$, where:
  - X is a non-terminal symbol.
  - $\alpha, \beta \in (\Gamma \cup \Sigma)^*$.
  - $\gamma \in (\Gamma \cup \Sigma)^+$.

Context-sensitive Grammars

Definition:

- Each production rule has the form $\alpha X\beta \rightarrow \alpha\gamma\beta$ or $S \rightarrow \gamma$, where:
  - X is a non-terminal symbol.
  - $\alpha, \beta \in (\Gamma \cup \Sigma)^*$.
  - $\gamma \in (\Gamma \cup \Sigma)^+$.
  - S is not mentioned in any right-hand side.

Unrestricted Grammars

Definition:

Unrestricted Grammars

Definition:

- No restrictions on the production rules.

Expressivity: Example

Consider the (standard example) language $L(G) = \{a^n b^n \mid n \geq 0\}$.

Expressivity: Example

Consider the (standard example) language $L(G) = \{a^n b^n \mid n \geq 0\}$.

- It is context-free! What is its (context-free) grammar?

Expressivity: Example

Consider the (standard example) language $L(G) = \{a^n b^n \mid n \geq 0\}$.

- It is context-free! What is its (context-free) grammar?
- Is is also regular?

Recap: Standard Decision Problems for Formal Languages

We only provide informal definitions here – as they are sufficient for the purpose of this lecture. For formal definitions, please consider any lecture/text book on Formal Grammars/Languages or Complexity Theory.

Recap: Standard Decision Problems for Formal Languages

We only provide informal definitions here – as they are sufficient for the purpose of this lecture. For formal definitions, please consider any lecture/text book on Formal Grammars/Languages or Complexity Theory.

- The *emptiness problem*: Does a grammar G contain any word at all? That is, holds $L(G) = \emptyset$?

Recap: Standard Decision Problems for Formal Languages

We only provide informal definitions here – as they are sufficient for the purpose of this lecture. For formal definitions, please consider any lecture/text book on Formal Grammars/Languages or Complexity Theory.

- The *emptiness problem*: Does a grammar G contain any word at all? That is, holds $L(G) = \emptyset$?
- The *word problem*: Given a grammar G and a word ω, can ω be generated by G, i.e., holds $\omega \in L(G)$?

Recap: Standard Decision Problems for Formal Languages

We only provide informal definitions here – as they are sufficient for the purpose of this lecture. For formal definitions, please consider any lecture/text book on Formal Grammars/Languages or Complexity Theory.

- The *emptiness problem*: Does a grammar G contain any word at all? That is, holds $L(G) = \emptyset$?
- The *word problem*: Given a grammar G and a word ω, can ω be generated by G, i.e., holds $\omega \in L(G)$?
- The *prefix problem*: Given a grammar G and a sequence of terminal symbols ω, is there a word produced by G, $\omega' \in L(G)$, such that ω is the prefix of ω'.

Recap: Standard Decision Problems for Formal Languages

We only provide informal definitions here – as they are sufficient for the purpose of this lecture. For formal definitions, please consider any lecture/text book on Formal Grammars/Languages or Complexity Theory.

- The *emptiness problem*: Does a grammar G contain any word at all? That is, holds $L(G) = \emptyset$?
- The *word problem*: Given a grammar G and a word ω, can ω be generated by G, i.e., holds $\omega \in L(G)$?
- The *prefix problem*: Given a grammar G and a sequence of terminal symbols ω, is there a word produced by G, $\omega' \in L(G)$, such that ω is the prefix of ω'.
- The *language intersection problem*: Given two grammars G and G', do they produce a common word? That is, holds $L(G) \cap L(G') \neq \emptyset$?

Recap: Standard Decision Problems for Formal Languages

We only provide informal definitions here – as they are sufficient for the purpose of this lecture. For formal definitions, please consider any lecture/text book on Formal Grammars/Languages or Complexity Theory.

- The *emptiness problem*: Does a grammar G contain any word at all? That is, holds $L(G) = \emptyset$?
- The *word problem*: Given a grammar G and a word ω, can ω be generated by G, i.e., holds $\omega \in L(G)$?
- The *prefix problem*: Given a grammar G and a sequence of terminal symbols ω, is there a word produced by G, $\omega' \in L(G)$, such that ω is the prefix of ω'.
- The *language intersection problem*: Given two grammars G and G', do they produce a common word? That is, holds $L(G) \cap L(G') \neq \emptyset$?
- The *language classification problem*: Given a set of words (i.e., a language), is there a grammar with certain properties that produces it?

Expressivity in Planning: Example

The agent (e.g., a robot) acts in an office environment. Constraint: Every door that he opens must be closed afterwards.

By which planning approach can this be expressed?

- Classical planning?
- Non-hierarchical, but also *non-classical* planning?
- Hierarchical planning? Under which restrictions?
  - With or without task insertion?
  - With or without conditional effects?
  - With limited recursion?

Planning: Relationship to Formal Languages

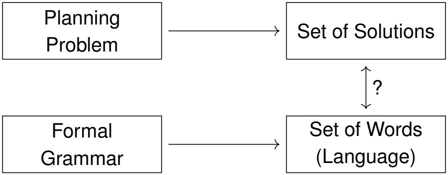
Planning Problem $\longrightarrow$ Set of Solutions

Formal Grammar $\longrightarrow$ Set of Words (Language)

Semantical Correspondence:

- Each planning problem can be interpreted as a compact representation of its solutions.
- Similarly, each formal grammar is a compact representation of its set of words, i.e., its language.

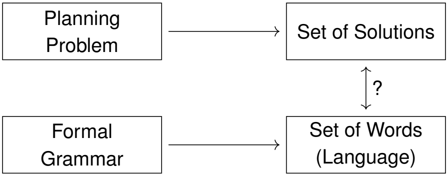
Planning: Relationship to Formal Languages



Semantical Correspondence:

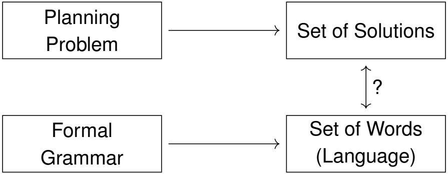
- Each planning problem can be interpreted as a compact representation of its solutions.
- Similarly, each formal grammar is a compact representation of its set of words, i.e., its language.
- So, what is the relationship?

Planning: Relationship to Formal Languages



Syntactic Correspondence:

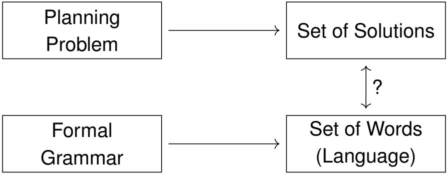
Planning: Relationship to Formal Languages



Syntactic Correspondence:

- Primitive tasks form the *terminal* symbols of a grammar.

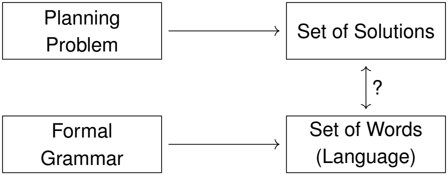
Planning: Relationship to Formal Languages



Syntactic Correspondence:

- Primitive tasks form the *terminal* symbols of a grammar.
- Abstract Tasks form the *non-terminal* symbols.

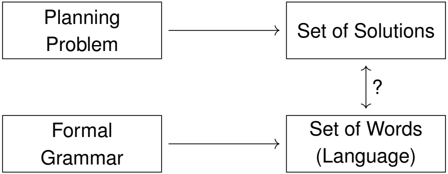
Planning: Relationship to Formal Languages



Syntactic Correspondence:

- Primitive tasks form the *terminal* symbols of a grammar.
- Abstract Tasks form the *non-terminal* symbols.
- Decomposition methods correspond to *production rules*.

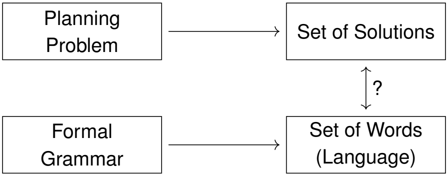
Planning: Relationship to Formal Languages



Syntactic Correspondence:

- Primitive tasks form the *terminal* symbols of a grammar.
- Abstract Tasks form the *non-terminal* symbols.
- Decomposition methods correspond to *production rules*.
- Set of HTN solutions forms the *language* of the problem.

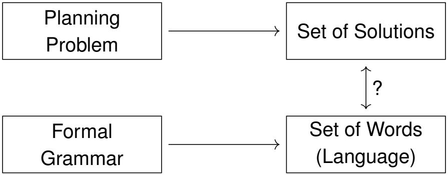
Planning: Relationship to Formal Languages



Syntactic Correspondence:

- Primitive tasks form the *terminal* symbols of a grammar.
- Abstract Tasks form the *non-terminal* symbols.
- Decomposition methods correspond to *production rules*.
- Set of HTN solutions forms the *language* of the problem.
- Analysis also works for TIHTN planning or non-hierarchical planning.

Planning: Relationship to Formal Languages



Further reading, including all of the next results:

- Daniel Höller et al. "Language Classification of Hierarchical Planning Problems". In: *Proc. of the 21st Europ. Conf. on Artificial Intelligence (ECAI 2014)*. IOS Press, 2014, pp. 447–452. DOI: 10.3233/978-1-61499-419-0-447
- Daniel Höller et al. "Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages". In: *Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS 2016)*. AAAI Press, 2016, pp. 158–165

A Closer Look to the Relationship of Planning to Formal Grammars

- *Emptiness problem* $\rightarrow$ *Plan existence problem*, i.e., is the given problem solvable?

A Closer Look to the Relationship of Planning to Formal Grammars

- *Emptiness problem* $\rightarrow$ *Plan existence problem*, i.e., is the given problem solvable?
- *Word Problem* $\rightarrow$ *Plan verification*, i.e., is a given “plan” actually a solution to the given planning problem?

A Closer Look to the Relationship of Planning to Formal Grammars

- *Emptiness problem* $\rightarrow$ *Plan existence problem*, i.e., is the given problem solvable?
- *Word Problem* $\rightarrow$ *Plan verification*, i.e., is a given “plan” actually a solution to the given planning problem?
- *Prefix problem* $\rightarrow$ *Plan recognition*, i.e., which plans could the agent currently be executing given the observed executed actions?

A Closer Look to the Relationship of Planning to Formal Grammars

- *Emptiness problem* $\rightarrow$ *Plan existence problem*, i.e., is the given problem solvable?
- *Word Problem* $\rightarrow$ *Plan verification*, i.e., is a given "plan" actually a solution to the given planning problem?
- *Prefix problem* $\rightarrow$ *Plan recognition*, i.e., which plans could the agent currently be executing given the observed executed actions?

The *language intersection problem* and the *language classification problem* are interesting (and useful) from a theoretical point of view, but there is no immediate correspondence to standard "planning questions".

The Language of a Planning Problem

- Let $\mathcal{P}$ be a planning problem. Then, $L(\mathcal{P}) = \{\omega \mid \omega \text{ is an executable linearization of some solution of } \mathcal{P}\}$.

The Language of a Planning Problem

- Let $\mathcal{P}$ be a planning problem. Then, $L(\mathcal{P}) = \{\omega \mid \omega \text{ is an executable linearization of some solution of } \mathcal{P}\}$.
- Note that this definition abstracts from various problem classes and algorithms:

The Language of a Planning Problem

- Let $\mathcal{P}$ be a planning problem. Then, $L(\mathcal{P}) = \{\omega \mid \omega$ is an executable linearization of some solution of $\mathcal{P}\}$.
- Note that this definition abstracts from various problem classes and algorithms:
  - STRIPS problems: correspondence is trivial (1-to-1).

The Language of a Planning Problem

- Let $\mathcal{P}$ be a planning problem. Then, $L(\mathcal{P}) = \{\omega \mid \omega \text{ is an executable linearization of some solution of } \mathcal{P}\}$.
- Note that this definition abstracts from various problem classes and algorithms:
  - STRIPS problems: correspondence is trivial (1-to-1).
  - POCL problems: for each POCL solution, *every action linearization* is in the language.

The Language of a Planning Problem

- Let $\mathcal{P}$ be a planning problem. Then, $L(\mathcal{P}) = \{\omega \mid \omega$ is an executable linearization of some solution of $\mathcal{P}\}$.
- Note that this definition abstracts from various problem classes and algorithms:
  - STRIPS problems: correspondence is trivial (1-to-1).
  - POCL problems: for each POCL solution, *every action linearization* is in the language.
  - For standard HTN planning, *every executability witness* of any solution is in the language.

The Language of a Planning Problem

- Let $\mathcal{P}$ be a planning problem. Then, $L(\mathcal{P}) = \{\omega \mid \omega$ is an executable linearization of some solution of $\mathcal{P}\}$.
- Note that this definition abstracts from various problem classes and algorithms:
  - STRIPS problems: correspondence is trivial (1-to-1).
  - POCL problems: for each POCL solution, *every action linearization* is in the language.
  - For standard HTN planning, *every executability witness* of any solution is in the language.
  - For HTN planning with *all executability semantics*, every linearization of any solution is in the language.

The Language of a Planning Problem, cont'd

- With $\mathcal{X}$ we denote the set of all languages of all planning problems of type *X*. For instance, $\mathcal{STRIPS}$ and $\mathcal{HTN}$ represent all STRIPS and HTN languages, respectively.

The Language of a Planning Problem, cont'd

- With $\mathcal{X}$ we denote the set of all languages of all planning problems of type X. For instance, $\mathcal{STRIPS}$ and $\mathcal{HTN}$ represent all STRIPS and HTN languages, respectively.
- Formally: $\mathcal{X} := \{L(\mathcal{P}) \mid \mathcal{P}$ is a planning problem of type $X\}$

The Language of a Planning Problem, cont'd

- With $\mathcal{X}$ we denote the set of all languages of all planning problems of type *X*. For instance, $\mathcal{STRIPS}$ and $\mathcal{HTN}$ represent all STRIPS and HTN languages, respectively.
- Formally: $\mathcal{X} := \{L(\mathcal{P}) \mid \mathcal{P}$ is a planning problem of type $X\}$
- Example: $\mathcal{STRIPS} = \{L(\mathcal{P}) \mid \mathcal{P}$ is a STRIPS planning problem$\}$

The *EXE* "Planning Problem"

- Let $\mathcal{P}$ be a STRIPS planning planning problem with empty goal description.

The *EXE* "Planning Problem"

- Let $\mathcal{P}$ be a STRIPS planning planning problem with empty goal description.
- The set of solutions of this *EXE (executablity) problem* is exactly the set of executable action sequences.

The *EXE* "Planning Problem"

- Let $\mathcal{P}$ be a STRIPS planning planning problem with empty goal description.
- The set of solutions of this *EXE (executablity) problem* is exactly the set of executable action sequences.
- With $\mathcal{EXE}$ we refer to the language of the respective problem class.

The *EXE* "Planning Problem"

- Let $\mathcal{P}$ be a STRIPS planning planning problem with empty goal description.
- The set of solutions of this *EXE (executablity) problem* is exactly the set of executable action sequences.
- With $\mathcal{EXE}$ we refer to the language of the respective problem class.
- Because of the missing goal description, EXE problems are less expressive than the regular languages.

The *EXE* “Planning Problem”, cont’d

Theorem

$\mathcal{EXE} \subsetneq \mathcal{REG}$

Proof:

The *EXE* "Planning Problem", cont'd

Theorem

$\mathcal{EXE} \subsetneq \mathcal{REG}$

Proof:

1. Show for all $L \in \mathcal{EXE}$ that $L \in \mathcal{REG}$. How?

The *EXE* "Planning Problem", cont'd

Theorem

$\mathcal{EXE} \subsetneq \mathcal{REG}$

Proof:

1. Show for all $L \in \mathcal{EXE}$ that $L \in \mathcal{REG}$. How? Construct an automaton.

The *EXE* "Planning Problem", cont'd

Theorem

$\mathcal{EXE} \subsetneq \mathcal{REG}$

Proof:

1. Show for all $L \in \mathcal{EXE}$ that $L \in \mathcal{REG}$. How? Construct an automaton.
2. Provide a language $L \in \mathcal{REG}$ with $L \notin \mathcal{EXE}$. How?

The *EXE* “Planning Problem”, cont’d

Theorem

$\mathcal{EXE} \subsetneq \mathcal{REG}$

Proof:

1. Show for all $L \in \mathcal{EXE}$ that $L \in \mathcal{REG}$. How? Construct an automaton.
2. Provide a language $L \in \mathcal{REG}$ with $L \notin \mathcal{EXE}$. How? Exploit an important property: If some plan is executable, than every prefix is as well (due to the missing goal description).

STRIPS

Theorem

$\mathcal{STRIPS} \subsetneq \mathcal{REG}$.

Proof:

STRIPS

Theorem

$\mathcal{STRIPS} \subsetneq \mathcal{REG}$.

Proof:

1. Show for all $L \in \mathcal{STRIPS}$ that $L \in \mathcal{REG}$. How?

STRIPS

Theorem

$\mathcal{STRIPS} \subsetneq \mathcal{REG}$.

Proof:

1. Show for all $L \in \mathcal{STRIPS}$ that $L \in \mathcal{REG}$. How? As before.

STRIPS

Theorem

$\mathcal{STRIPS} \subsetneq \mathcal{REG}$.

Proof:

1. Show for all $L \in \mathcal{STRIPS}$ that $L \in \mathcal{REG}$. How? As before.
2. Provide a language $L \in \mathcal{REG}$ with $L \notin \mathcal{STRIPS}$. How?

STRIPS

Theorem

$\mathcal{STRIPS} \subsetneq \mathcal{REG}$.

Proof:

1. Show for all $L \in \mathcal{STRIPS}$ that $L \in \mathcal{REG}$. How? As before.
2. Provide a language $L \in \mathcal{REG}$ with $L \notin \mathcal{STRIPS}$. How? Again, provide a finite language that cannot be expressed as a STRIPS planning problem.

STRIPS, cont'd

For the second step in the previous proof, exploit:

Theorem

Let $s \in S$ be a state and $a \in A$ an action. If a is applicable in s' (resulting from applying a in s), then a is applicable arbitrarily often.

STRIPS, cont'd

For the second step in the previous proof, exploit:

Theorem

Let $s \in S$ be a state and $a \in A$ an action. If a is applicable in s' (resulting from applying a in s), then a is applicable arbitrarily often.

Proof:
Exercise (just show it directly via playing with preconditions and effects).

STRIPS with Conditional Effects

Theorem

The language of STRIPS problems with conditional effects, $\mathcal{STRIPS}\text{–}\mathcal{CE}$, is equivalent to the regular languages, $\mathcal{REG}$.

Proof:

STRIPS with Conditional Effects

Theorem

The language of STRIPS problems with conditional effects, $\mathcal{STRIPS{-}CE}$, is equivalent to the regular languages, $\mathcal{REG}$.

Proof:

1. For every SCE planning problem, there is an equivalent regular language.

STRIPS with Conditional Effects

Theorem

The language of STRIPS problems with conditional effects, $\mathcal{STRIPS}{-}\mathcal{CE}$, is equivalent to the regular languages, $\mathcal{REG}$.

Proof:

1. For every SCE planning problem, there is an equivalent regular language.
2. For every regular language, there is a SCE problem generating it.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.
- We define:

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \rightarrow S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.
- We define:
  - $\Sigma = A$.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.
- We define:
  - $\Sigma = A$.
  - $S = 2^V$ (in planning, the set of states is also defined as S).

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.
- We define:
  - $\Sigma = A$.
  - $S = 2^V$ (in planning, the set of states is also defined as S).
  - d is given by:

$$d(s, a) = \begin{cases} s', & \textit{iff } (\tau(a, s) \wedge \gamma(a, s) = s') \\ \textit{undefined}, & \textit{else} \end{cases}$$

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.
- We define:
  - $\Sigma = A$.
  - $S = 2^V$ (in planning, the set of states is also defined as S).
  - d is given by:

$$d(s, a) = \begin{cases} s', & \textit{iff } (\tau(a, s) \wedge \gamma(a, s) = s') \\ \textit{undefined}, & \textit{else} \end{cases}$$

  - $i = s_I$.

STRIPS with Conditional Effects, cont'd

- Let $\mathcal{P} = (V, A, s_I, g)$ be a planning problem.
- We define a Deterministic Finite Automaton (Σ, S, d, i, F) with
  - Σ is its finite input alphabet.
  - S its finite set of states.
  - $d : S \times \Sigma \to S$ its state-transition function.
  - i its initial state.
  - $F \subseteq S$ its set of final states.
- We define:
  - $\Sigma = A$.
  - $S = 2^V$ (in planning, the set of states is also defined as S).
  - d is given by:

$$d(s, a) = \begin{cases} s', & \textit{iff } (\tau(a, s) \wedge \gamma(a, s) = s') \\ \textit{undefined}, & \textit{else} \end{cases}$$

  - $i = s_I$.
  - Every goal state $s \supseteq g$ is included in F.

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$,

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

$$\forall a \in A : prec(a)$$

$$add(a)$$

$$del(a)$$

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

$$\forall a \in A : prec(a) = \emptyset$$

$$add(a)$$

$$del(a)$$

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

$$\forall a \in A : prec(a) = \emptyset$$

$$add(a) = \{(\{s\} \rightarrow \{s'\} \qquad) \mid d(s,a) = s'\}$$

$$del(a)$$

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

$$\forall a \in A : prec(a) = \emptyset$$

$$add(a) = \{(\{s\} \rightarrow \{s'\} \cup G') \mid d(s,a) = s'\}$$

$$del(a)$$

Language of STRIPS with Conditional Effects

- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

$$\forall a \in A : prec(a) = \emptyset$$

$$add(a) = \{(\{s\} \rightarrow \{s'\} \cup G') \mid d(s,a) = s'\}$$

$$\text{with } G' = \begin{cases} \{g\}, & \text{if } s' \in F \\ \emptyset, & \text{else} \end{cases}$$

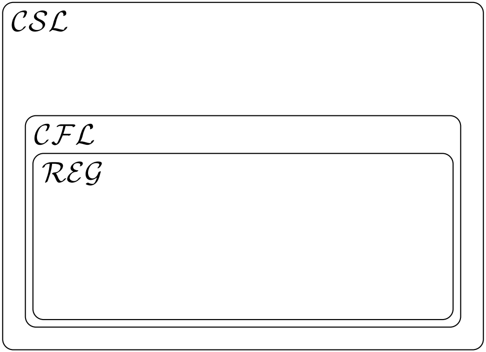
$$del(a)$$

Language of STRIPS with Conditional Effects

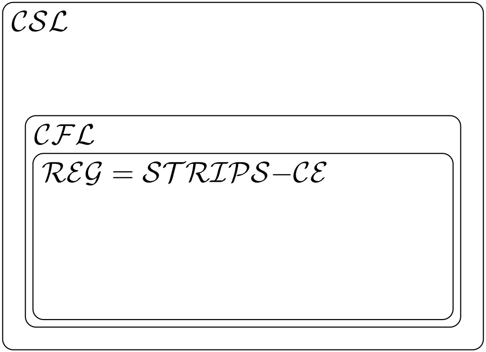
- Let (Σ, S, d, i, F) be a Deterministic Finite Automaton.
- We define a planning problem $\mathcal{P} = (V, A, s_I, g)$ with:
  - $V = S \cup \{g\}$ and $g \notin S$.
  - $s_I = \{i\}$, $g \in s_I$ iff $i \in F$.
  - A equals the alphabet Σ and

$$
\begin{aligned}
\forall a \in A : \mathit{prec}(a) &= \emptyset \\
\mathit{add}(a) &= \{(\{s\} \to \{s'\} \cup G') \mid d(s,a) = s'\} \\
\text{with } G' &= \begin{cases} \{g\}, & \text{if } s' \in F \\ \emptyset, & \text{else} \end{cases} \\
\mathit{del}(a) &= \{(\emptyset \to V)\}
\end{aligned}
$$

Expressivity via Comparison to Formal Languages

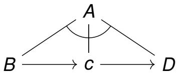


Expressivity via Comparison to Formal Languages



Totally Ordered HTN Planning Problems

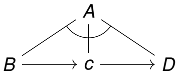
- Decomposition in totally ordered HTN planning problems is similar to rule application in context-free grammars.



$$A \to BcD$$

Totally Ordered HTN Planning Problems

- Decomposition in totally ordered HTN planning problems is similar to rule application in context-free grammars.

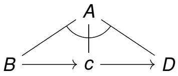


$$A \rightarrow BcD$$

- The encoding of (totally ordered) HTN decomposition as (context-free) grammar rules and vice versa is straightforward.

Totally Ordered HTN Planning Problems

- Decomposition in totally ordered HTN planning problems is similar to rule application in context-free grammars.

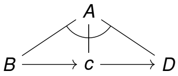


$$A \rightarrow BcD$$

- The encoding of (totally ordered) HTN decomposition as (context-free) grammar rules and vice versa is straightforward.
- $\mathcal{HTN}-\mathcal{ORD} \supseteq \mathcal{CFL}$ is trivial, since no states are required.

Totally Ordered HTN Planning Problems

- Decomposition in totally ordered HTN planning problems is similar to rule application in context-free grammars.

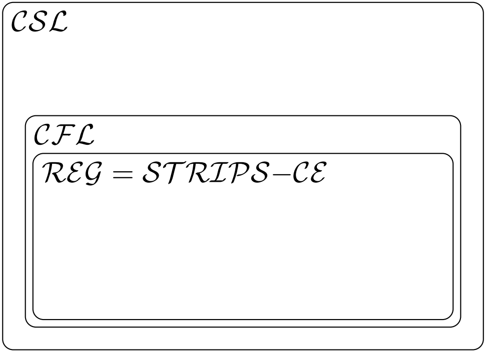


$$A \rightarrow BcD$$

- The encoding of (totally ordered) HTN decomposition as (context-free) grammar rules and vice versa is straightforward.
- $\mathcal{HTN}{-}\mathcal{ORD} \supseteq \mathcal{CFL}$ is trivial, since no states are required.
- Constraints introduced by preconditions and effects can be treated via intersection with a regular language:

  Remember that the intersection of any context-free language with any regular language is still context-free. Thus, we can intersect the language representing the hierarchy (which is context-free) with one of the regular languages $\mathcal{EXE}$ or $\mathcal{STRIPS}$ (do we feature a goal description?) to show $\mathcal{HTN}{-}\mathcal{ORD} \subseteq \mathcal{CFL}$.

Expressivity via Comparison to Formal Languages



Expressivity via Comparison to Formal Languages

$\mathcal{CSL}$

$\mathcal{CFL} = \mathcal{HTN}\text{-}\mathcal{ORD}$

$\mathcal{REG} = \mathcal{STRIPS}\text{-}\mathcal{CE}$

Acyclic HTN Problems

- Informally/intuitively, *acyclic HTN/TIHTN problems* are problems where no recursion is possible.
- There are many equivalent formal definitions, some of them will be covered later. For instance: For every task network that is reachable via decomposition from the initial task network holds: Let *dt* be its decomposition tree. Then, no path from its root node to any of its leafs contains the same task more than once.

TIHTN and Acyclic HTN Problems

The following results can easily be shown:

- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$

TIHTN and Acyclic HTN Problems

The following results can easily be shown:

- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$
- $\mathcal{HTN}{-}\mathcal{AC} \subsetneq \mathcal{REG}$

TIHTN and Acyclic HTN Problems

The following results can easily be shown:

- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$
- $\mathcal{HTN}{-}\mathcal{AC} \subsetneq \mathcal{REG}$
- There exist the following languages *L*:

TIHTN and Acyclic HTN Problems

The following results can easily be shown:

- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$
- $\mathcal{HTN}{-}\mathcal{AC} \subsetneq \mathcal{REG}$
- There exist the following languages L:
  - $L \in \mathcal{STRIPS} \cap \mathcal{HTN}{-}\mathcal{AC}$

TIHTN and Acyclic HTN Problems

The following results can easily be shown:

- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$
- $\mathcal{HTN}{-}\mathcal{AC} \subsetneq \mathcal{REG}$
- There exist the following languages L:
  - $L \in \mathcal{STRIPS} \cap \mathcal{HTN}{-}\mathcal{AC}$
  - $L \in \mathcal{TIHTN}$ and $L \in \cap\mathcal{HTN}{-}\mathcal{AC}$ and $L \notin \cap\mathcal{STRIPS}$

TIHTN and Acyclic HTN Problems

The following results can easily be shown:

- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$
- $\mathcal{HTN}{-}\mathcal{AC} \subsetneq \mathcal{REG}$
- There exist the following languages L:
  - $L \in \mathcal{STRIPS} \cap \mathcal{HTN}{-}\mathcal{AC}$
  - $L \in \mathcal{TIHTN}$ and $L \in \cap\mathcal{HTN}{-}\mathcal{AC}$ and $L \notin \cap\mathcal{STRIPS}$
  - $L \in \mathcal{TIHTN}$ and $L \notin \cap\mathcal{HTN}{-}\mathcal{AC}$ and $L \notin \cap\mathcal{STRIPS}$

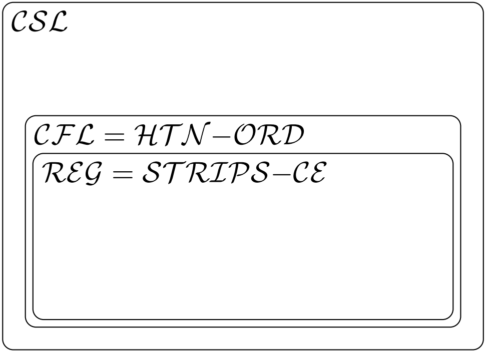
TIHTN and Acyclic HTN Problems

The following results can easily be shown:

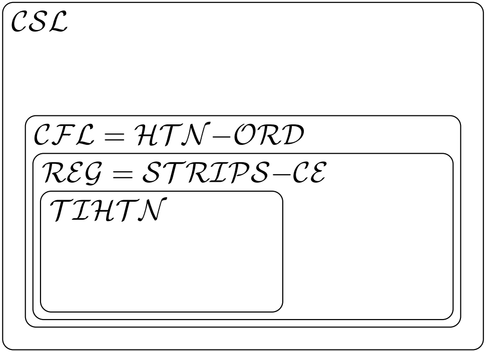
- $\mathcal{STRIPS} \subsetneq \mathcal{TIHTN} \subsetneq \mathcal{REG}$
- $\mathcal{HTN}{-}\mathcal{AC} \subsetneq \mathcal{REG}$
- There exist the following languages L:
  - $L \in \mathcal{STRIPS} \cap \mathcal{HTN}{-}\mathcal{AC}$
  - $L \in \mathcal{TIHTN}$ and $L \in \cap\mathcal{HTN}{-}\mathcal{AC}$ and $L \notin \cap\mathcal{STRIPS}$
  - $L \in \mathcal{TIHTN}$ and $L \notin \cap\mathcal{HTN}{-}\mathcal{AC}$ and $L \notin \cap\mathcal{STRIPS}$

These results rely on the presence of goal descriptions! More details in the exercises.

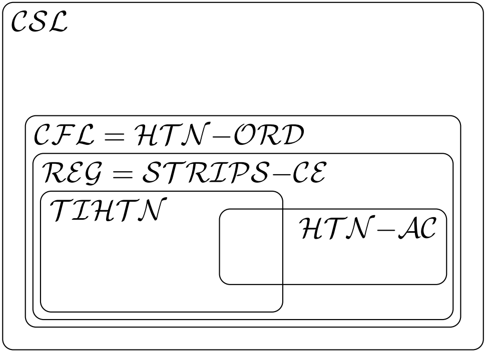
Expressivity via Comparison to Formal Languages



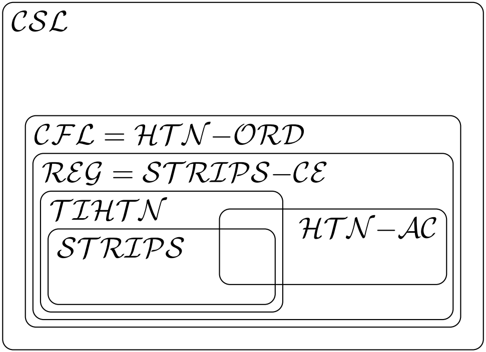
Expressivity via Comparison to Formal Languages



Expressivity via Comparison to Formal Languages



Expressivity via Comparison to Formal Languages



Noop HTN Planning Problems

- Subtasks of the problem's methods may be partially ordered.

Noop HTN Planning Problems

- Subtasks of the problem's methods may be partially ordered.
- First class we look at:

Noop HTN Planning Problems

- Subtasks of the problem's methods may be partially ordered.
- First class we look at:
  $\mathcal{HTN}{-}\mathcal{NOOP}$ – actions have no preconditions and effects.

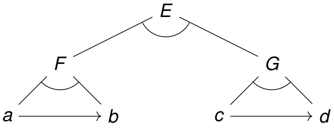
Noop HTN Planning Problems

- Subtasks of the problem's methods may be partially ordered.
- First class we look at:

  $\mathcal{HTN}{-}\mathcal{NOOP}$ – actions have no preconditions and effects.
- Can a partially ordered method be transformed into a set of totally ordered methods?

Noop HTN Planning Problems, cont'd I

HTN

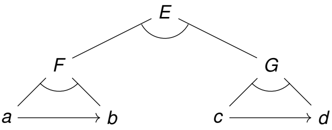


Grammar

$$E \rightarrow FG \quad F \rightarrow ab$$
$$E \rightarrow GF \quad G \rightarrow cd$$

Noop HTN Planning Problems, cont'd I

HTN



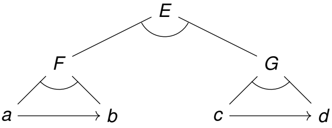
Grammar

$$E \rightarrow FG \quad F \rightarrow ab$$
$$E \rightarrow GF \quad G \rightarrow cd$$

Word 1 *cdab*

Noop HTN Planning Problems, cont'd I

HTN



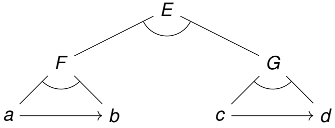
Grammar

$$E \rightarrow FG \quad F \rightarrow ab$$
$$E \rightarrow GF \quad G \rightarrow cd$$

Word 1 *cdab* ✓

Noop HTN Planning Problems, cont'd I

HTN



Grammar

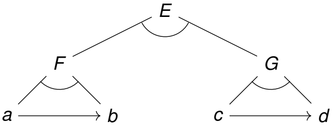
$E \rightarrow FG \quad F \rightarrow ab$

$E \rightarrow GF \quad G \rightarrow cd$

Word 1 *cdab* ✓ Word 2 *acbd*

Noop HTN Planning Problems, cont'd I

HTN



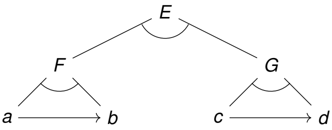
Grammar

$$E \rightarrow FG \quad F \rightarrow ab$$
$$E \rightarrow GF \quad G \rightarrow cd$$

Word 1 *cdab* ✓ Word 2 *acbd* ✗

Noop HTN Planning Problems, cont'd I

HTN



Grammar

$$E \rightarrow FG \quad F \rightarrow ab$$
$$E \rightarrow GF \quad G \rightarrow cd$$

Word 1 *cdab* ✓

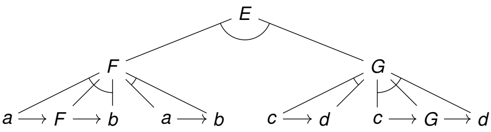
$ab||cd$

Word 2 *acbd* ✗

$\{abcd\} \cup \{cdab\}$

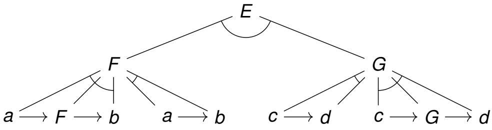
Noop HTN Planning Problems, cont'd II

- The HTN depicted below generates the language $a^n b^n || c^m d^m$.



Noop HTN Planning Problems, cont'd II

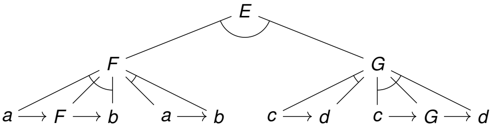
- The HTN depicted below generates the language $a^n b^n || c^m d^m$.
- Using the *Pumping Lemma* for context-free languages, it can be shown that this language is not context-free.



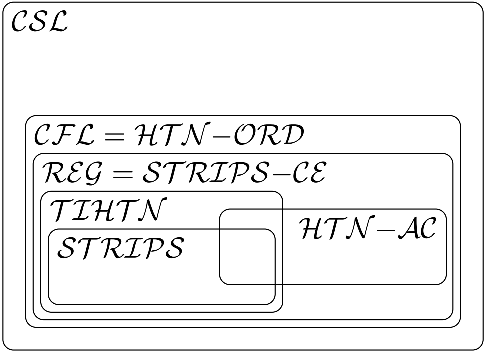
Noop HTN Planning Problems, cont'd II

- The HTN depicted below generates the language $a^n b^n || c^m d^m$.
- Using the *Pumping Lemma* for context-free languages, it can be shown that this language is not context-free.

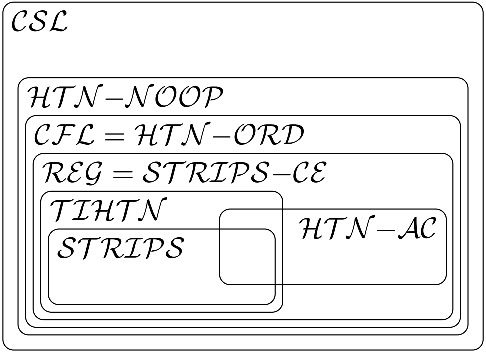
$\rightarrow$ $\mathcal{CFL} \subsetneq \mathcal{HTN}\text{–}\mathcal{NOOP}$



Expressivity via Comparison to Formal Languages



Expressivity via Comparison to Formal Languages



(Unrestricted) HTN Planning Problems

- $\mathcal{HTN} \subseteq \mathcal{CSL}$ can be shown by providing a linear space-bounded Turing machine (also called: LBA, linear-bounded automaton) that decides the word problem for every HTN problem.

(Unrestricted) HTN Planning Problems

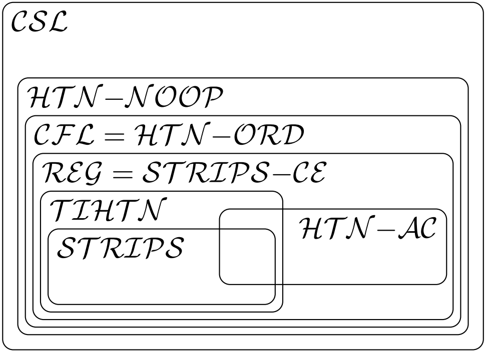
- $\mathcal{HTN} \subseteq \mathcal{CSL}$ can be shown by providing a linear space-bounded Turing machine (also called: LBA, linear-bounded automaton) that decides the word problem for every HTN problem.
- $\mathcal{HTN} \subsetneq \mathcal{CSL}$ can be shown by the language $\{a^p \mid p \text{ prime}\}$, which cannot be produced by an HTN problem.

(Unrestricted) HTN Planning Problems

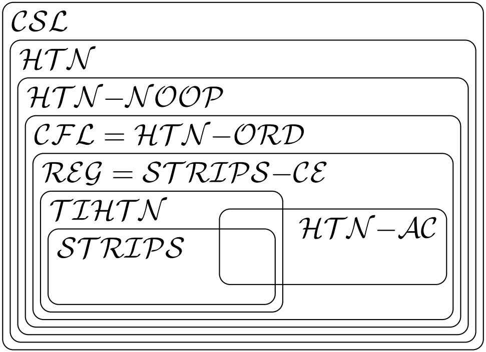
- $\mathcal{HTN} \subseteq \mathcal{CSL}$ can be shown by providing a linear space-bounded Turing machine (also called: LBA, linear-bounded automaton) that decides the word problem for every HTN problem.
- $\mathcal{HTN} \subsetneq \mathcal{CSL}$ can be shown by the language $\{a^p \mid p \text{ prime}\}$, which cannot be produced by an HTN problem.

$\rightarrow$ These results are just mentioned for the sake of completeness. Proofs are omitted.

Expressivity via Comparison to Formal Languages



Expressivity via Comparison to Formal Languages



Extensions of Expressivity Analysis

Several results could still be investigated, e.g.:

Extensions of Expressivity Analysis

Several results could still be investigated, e.g.:

- Conditional effects in all classes, not just in STRIPS.

Extensions of Expressivity Analysis

Several results could still be investigated, e.g.:

- Conditional effects in all classes, not just in STRIPS.
- No-ops in all classes, not just in non-restricted HTNs.

Extensions of Expressivity Analysis

Several results could still be investigated, e.g.:

- Conditional effects in all classes, not just in STRIPS.
- No-ops in all classes, not just in non-restricted HTNs.
- Further restrictions on hierarchy (e.g., tail-recursive problems), cf. chapter on complexity theory.

Extensions of Expressivity Analysis

Several results could still be investigated, e.g.:

- Conditional effects in all classes, not just in STRIPS.
- No-ops in all classes, not just in non-restricted HTNs.
- Further restrictions on hierarchy (e.g., tail-recursive problems), cf. chapter on complexity theory.
- Even higher language features, e.g., functions.

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.
- Expressivity analysis studies the structural properties of the solutions that can be generated.

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.
- Expressivity analysis studies the structural properties of the solutions that can be generated.
- Analysis abstracts from the problem size and tells little about how hard a problem is to solve.

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.
- Expressivity analysis studies the structural properties of the solutions that can be generated.
- Analysis abstracts from the problem size and tells little about how hard a problem is to solve.
  - No-op HTNs are more expressive than STRIPS problems.

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.
- Expressivity analysis studies the structural properties of the solutions that can be generated.
- Analysis abstracts from the problem size and tells little about how hard a problem is to solve.
  - No-op HTNs are more expressive than STRIPS problems.
  - Yet No-op HTNs can be decided (plan existence) in $\mathbb{P}$, whereas STRIPS problems are $\mathbb{PSPACE} - complete$ (see chapter on complexity theory).

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.
- Expressivity analysis studies the structural properties of the solutions that can be generated.
- Analysis abstracts from the problem size and tells little about how hard a problem is to solve.
  - No-op HTNs are more expressive than STRIPS problems.
  - Yet No-op HTNs can be decided (plan existence) in $\mathbb{P}$, whereas STRIPS problems are $\mathbb{PSPACE} -$ *complete* (see chapter on complexity theory).
- The comparison to formal grammars is independent of lifting/grounding!

Summary

- To choose an adequate formalism for a problem at hand, we need to know the expressivity of the different formalisms.
- Expressivity analysis studies the structural properties of the solutions that can be generated.
- Analysis abstracts from the problem size and tells little about how hard a problem is to solve.
  - No-op HTNs are more expressive than STRIPS problems.
  - Yet No-op HTNs can be decided (plan existence) in $\mathbb{P}$, whereas STRIPS problems are $\mathbb{PSPACE} - \textit{complete}$ (see chapter on complexity theory).
- The comparison to formal grammars is independent of lifting/grounding!
- Our analysis reveals interesting relationships between standard problems in formal grammars/languages and planning.