Principles of AI Planning

Expressive power

Malte Helmert    Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

January 10th, 2007
Expressive power is the motivation for designing new planning languages.

Often there is the question: *Syntactic sugar* or *essential feature*?

Compiling away or change planning algorithm?

If a feature can be compiled away, then it is apparently only *syntactic sugar*.

Sometimes, however, a compilation can lead to much larger planning domain descriptions or to much longer plans.

This means the planning algorithm will probably choke, i.e., it cannot be considered as a *compilation*.
Motivation: Why Analyzing the Expressive Power?

- **Expressive power** is the motivation for designing new planning languages.
- Often there is the question: *Syntactic sugar* or *essential feature*?

→ *Compiling away* or change planning algorithm?

→ If a feature can be compiled away, then it is apparently only *syntactic sugar*.

- Sometimes, however, a compilation can lead to much larger planning domain descriptions or to much longer plans.

→ This means the planning algorithm will probably choke, i.e., it cannot be considered as a *compilation*.
Motivation: Why Analyzing the Expressive Power?

- **Expressive power** is the motivation for designing new planning languages
- Often there is the question: *Syntactic sugar* or *essential feature*?

⇒ *Compiling away* or change planning algorithm?

→ If a feature can be compiled away, then it is apparently only *syntactic sugar*.

- Sometimes, however, a compilation can lead to much larger planning domain descriptions or to much longer plans.

⇒ This means the planning algorithm will probably choke, i.e., it cannot be considered as a *compilation*. 
Motivation: Why Analyzing the Expressive Power?

Expressive power is the motivation for designing new planning languages.

Often there is the question: *Syntactic sugar* or *essential feature*?

→ *Compiling away* or change planning algorithm?

If a feature can be compiled away, then it is apparently only *syntactic sugar*.

Sometimes, however, a compilation can lead to much larger planning domain descriptions or to much longer plans.

This means the planning algorithm will probably choke, i.e., it cannot be considered as a *compilation*.
Motivation: Why Analyzing the Expressive Power?

- **Expressive power** is the motivation for designing new planning languages.
- Often there is the question: *Syntactic sugar* or *essential feature*?
  - *Compiling away* or change planning algorithm?
  - If a feature can be compiled away, then it is apparently only *syntactic sugar*.
- Sometimes, however, a compilation can lead to much larger planning domain descriptions or to much longer plans.
  - This means the planning algorithm will probably choke, i.e., it cannot be considered as a *compilation*.
Expressive power is the motivation for designing new planning languages.

Often there is the question: *Syntactic sugar* or *essential feature*?

\[ \Rightarrow \text{Compiling away} \] or change planning algorithm?

\[ \rightarrow \text{If a feature can be compiled away, then it is apparently only *syntactic sugar*.} \]

Sometimes, however, a compilation can lead to much larger planning domain descriptions or to much longer plans.

\[ \Rightarrow \text{This means the planning algorithm will probably choke, i.e., it cannot be considered as a *compilation*.} \]
Example: DNF Preconditions

- Assume we have **DNF preconditions** in STRIPS operators
- This can be **compiled away** as follows
- **Split** each operator with a DNF precondition $c_1 \lor \ldots \lor c_n$ into $n$ operators with the same effects and $c_i$ as preconditions

$\Rightarrow$ If there exists a plan for the original planning task there is one for the new planning task and *vice versa*

$\Rightarrow$ The planning task has almost the same size

$\Rightarrow$ The shortest plans have the same size
Example: DNF Preconditions

- Assume we have **DNF preconditions** in STRIPS operators
- This can be **compiled away** as follows
  - **Split** each operator with a DNF precondition $c_1 \lor \ldots \lor c_n$ into $n$ operators with the same effects and $c_i$ as preconditions
  - **If** there exists a plan for the original planning task there is one for the new planning task and **vice versa**
  - The planning task has almost the **same size**
  - The shortest plans have the **same size**
Example: DNF Preconditions

- Assume we have **DNF preconditions** in STRIPS operators.
- This can be **compiled away** as follows.
- **Split** each operator with a DNF precondition $c_1 \lor \ldots \lor c_n$ into $n$ operators with the same effects and $c_i$ as preconditions.

$\implies$ If there exists a plan for the original planning task there is one for the new planning task and *vice versa*.

$\rightarrow$ The planning task has almost the same size.

$\rightarrow$ The shortest plans have the same size.
Example: DNF Preconditions

- Assume we have **DNF preconditions** in STRIPS operators
- This can be **compiled away** as follows
- **Split** each operator with a DNF precondition $c_1 \lor \ldots \lor c_n$ into $n$ operators with the same effects and $c_i$ as preconditions

\[\implies \text{If there exists a plan for the original planning task there is one for the new planning task and } \text{vice versa}\]

\[\implies \text{The planning task has almost the same size}\]

\[\implies \text{The shortest plans have the same size}\]
Example: DNF Preconditions

- Assume we have **DNF preconditions** in STRIPS operators
- This can be **compiled away** as follows
- **Split** each operator with a DNF precondition $c_1 \lor \ldots \lor c_n$ into $n$ operators with the same effects and $c_i$ as preconditions

~~> If there exists a plan for the original planning task there is one for the new planning task and *vice versa*
→ The planning task has almost the same size
→ The shortest plans have the same size
Example: DNF Preconditions

- Assume we have **DNF preconditions** in STRIPS operators
- This can be **compiled away** as follows
- **Split** each operator with a DNF precondition $c_1 \lor \ldots \lor c_n$ into $n$ operators with the same effects and $c_i$ as preconditions
  - If there exists a plan for the original planning task there is one for the new planning task and *vice versa*
  - The **planning task** has almost the **same size**
  - The **shortest plans** have the **same size**
Can we compile away **conditional effects** to STRIPS?

- Example operator: \( \langle a, b \triangleright d \land \neg c \triangleright e \rangle \)
- Can be translated into four operators:
  \( \langle a \land b \land c, d \rangle, \langle a \land b \land \neg c, d \land e \rangle, \ldots \)
- Plan existence and plan size are identical
- **Exponential blowup** of domain description!

→ Can this be avoided?
Example: Conditional effects

- Can we compile away **conditional effects** to STRIPS?
- Example operator: \( \langle a, b \triangleright d \land \neg c \triangleright e \rangle \)
  
  Can be translated into four operators:
  \( \langle a \land b \land c, d \rangle, \langle a \land b \land \neg c, d \land e \rangle, \ldots \)

- Plan existence and plan size are identical
- **Exponential blowup** of domain description!

→ Can this be avoided?
Example: Conditional effects

- Can we compile away **conditional effects** to STRIPS?
- Example operator: $\langle a, b \triangleright d \land \neg c \triangleright e \rangle$
- Can be translated into four operators:
  $\langle a \land b \land c, d \rangle, \langle a \land b \land \neg c, d \land e \rangle, \ldots$
- Plan existence and plan size are identical
- **Exponential blowup** of domain description!
  $\rightarrow$ Can this be avoided?
Example: Conditional effects

- Can we compile away \textit{conditional effects} to STRIPS?
- Example operator: \( \langle a, b \triangleright d \land \neg c \triangleright e \rangle \)
- Can be translated into four operators:
  \( \langle a \land b \land c, d \rangle, \langle a \land b \land \neg c, d \land e \rangle, \ldots \)
- Plan \textit{existence} and plan \textit{size} are identical
- \textit{Exponential blowup} of domain description!
- \( \rightarrow \) Can this be avoided?
Example: Conditional effects

- Can we compile away **conditional effects** to STRIPS?
- Example operator: \( \langle a, b \triangleright d \land \neg c \triangleright e \rangle \)
- Can be translated into four operators:
  \( \langle a \land b \land c, d \rangle, \langle a \land b \land \neg c, d \land e \rangle, \ldots \)
- Plan **existence** and plan **size** are identical
- **Exponential blowup** of domain description!
  \[\rightarrow\] Can this be avoided?
Example: Conditional effects

- Can we compile away **conditional effects** to STRIPS?
- Example operator: $\langle a, b \triangleright d \land \neg c \triangleright e \rangle$
- Can be translated into four operators:
  $\langle a \land b \land c, d \rangle, \langle a \land b \land \neg c, d \land e \rangle, \ldots$
- Plan **existence** and plan **size** are identical
- **Exponential blowup** of domain description!
- $\rightarrow$ Can this be avoided?
In the following we will only consider propositional STRIPS and some variants of it.

**Planning task:**

\[ T = \langle A, I, O, G \rangle. \]

**Often we refer to domain structures** \( \mathcal{D} = \langle A, O \rangle. \)
In the following we will only consider **propositional STRIPS** and some variants of it.

**Planning task:**

\[ T = \langle A, I, O, G \rangle. \]

Often we refer to **domain structures** \( D = \langle A, O \rangle. \)
In the following we will only consider **propositional STRIPS** and some variants of it.

**Planning task:**

\[ T = \langle A, I, O, G \rangle. \]

Often we refer to **domain structures** \( D = \langle A, O \rangle \).
Kambhampati et al [ECP 97] and Gazen & Knoblock [ECP 97]: Disjunctive preconditions are trivial – since they can be translated to basic STRIPS (DNF-preconditions)

Bäckström [AIJ 95]: Disjunctive preconditions are probably essential – since they can not easily be translated to basic STRIPS (CNF-preconditions)

Anderson et al [AIPS 98]: “[D]isjunctive preconditions . . . are . . . essential prerequisites for handling conditional effects” ~ conditional effects imply disjunctive preconditions (?) (General Boolean preconditions)
Disjunctive Preconditions: Trivial or Essential?

- Kambhampati et al [ECP 97] and Gazen & Knoblock [ECP 97]: Disjunctive preconditions are trivial – since they can be translated to basic STRIPS (DNF-preconditions)
- Bäckström [AIJ 95]: Disjunctive preconditions are probably essential – since they can not easily be translated to basic STRIPS (CNF-preconditions)
- Anderson et al [AIPS 98]: “[D]isjunctive preconditions . . . are . . . essential prerequisites for handling conditional effects” ⟷ conditional effects imply disjunctive preconditions (?) (General Boolean preconditions)
Kambhampati et al [ECP 97] and Gazen & Knoblock [ECP 97]: Disjunctive preconditions are trivial – since they can be translated to basic STRIPS (DNF-preconditions)

Bäckström [AIJ 95]: Disjunctive preconditions are probably essential – since they can not easily be translated to basic STRIPS (CNF-preconditions)

Anderson et al [AIPS 98]: “[D]isjunctive preconditions . . . are . . . essential prerequisites for handling conditional effects” ⟷ conditional effects imply disjunctive preconditions (?) (General Boolean preconditions)
More “Expressive Power”

\[ \text{STRIPS}_N : \text{plain strips with negative literals} \]
\[ \text{STRIPS}_{Bd} : \text{precondition in disjunctive normal form} \]
\[ \text{STRIPS}_{Bc} : \text{precondition in conjunctive normal form} \]
\[ \text{STRIPS}_B : \text{Boolean expressions as preconditions} \]
\[ \text{STRIPS}_C : \text{conditional effects} \]
\[ \text{STRIPS}_{C,N} : \text{conditional effects & negative literals} \]

...
More “Expressive Power”

\[
\begin{align*}
\text{STRIPS}_N &: \text{plain strips with negative literals} \\
\text{STRIPS}_{Bd} &: \text{precondition in disjunctive normal form} \\
\text{STRIPS}_{Bc} &: \text{precondition in conjunctive normal form} \\
\text{STRIPS}_B &: \text{Boolean expressions as preconditions} \\
\text{STRIPS}_C &: \text{conditional effects} \\
\text{STRIPS}_{C,N} &: \text{conditional effects & negative literals} \\
\ldots
\end{align*}
\]
More “Expressive Power”

\[
\begin{align*}
\text{STRIPS}_N &: \text{plain strips with negative literals} \\
\text{STRIPS}_{Bd} &: \text{precondition in disjunctive normal form} \\
\text{STRIPS}_{Bc} &: \text{precondition in conjunctive normal form} \\
\text{STRIPS}_B &: \text{Boolean expressions as preconditions} \\
\text{STRIPS}_C &: \text{conditional effects} \\
\text{STRIPS}_{C,N} &: \text{conditional effects & negative literals} \\
\ldots &
\end{align*}
\]
More “Expressive Power”

$\text{STRIPS}_N$ : plain strips with negative literals
$\text{STRIPS}_{Bd}$ : precondition in disjunctive normal form
$\text{STRIPS}_{Bc}$ : precondition in conjunctive normal form
$\text{STRIPS}_B$ : Boolean expressions as preconditions
$\text{STRIPS}_C$ : conditional effects
$\text{STRIPS}_{C,N}$ : conditional effects & negative literals

...
More “Expressive Power”

\[ \text{STRIPS}_N \] : plain strips with negative literals
\[ \text{STRIPS}_{Bd} \] : precondition in disjunctive normal form
\[ \text{STRIPS}_{Bc} \] : precondition in conjunctive normal form
\[ \text{STRIPS}_B \] : Boolean expressions as preconditions
\[ \text{STRIPS}_C \] : conditional effects
\[ \text{STRIPS}_{C,N} \] : conditional effects & negative literals

...
More “Expressive Power”

$\text{STRIPS}_{N}$: plain strips with negative literals
$\text{STRIPS}_{Bd}$: precondition in disjunctive normal form
$\text{STRIPS}_{Bc}$: precondition in conjunctive normal form
$\text{STRIPS}_{B}$: Boolean expressions as preconditions
$\text{STRIPS}_{C}$: conditional effects
$\text{STRIPS}_{C,N}$: conditional effects & negative literals
More “Expressive Power”

\[ \text{STRIPS}_N \] : plain strips with negative literals
\[ \text{STRIPS}_{Bd} \] : precondition in disjunctive normal form
\[ \text{STRIPS}_{Bc} \] : precondition in conjunctive normal form
\[ \text{STRIPS}_B \] : Boolean expressions as preconditions
\[ \text{STRIPS}_C \] : conditional effects
\[ \text{STRIPS}_{C,N} \] : conditional effects & negative literals
Theorem

**PLANEX is PSPACE-complete for STRIPS\(_N\), STRIPS\(_C,B\), and for all formalisms “between” the two.**

Proof.

Follows from theorems proved in the previous lecture.
Theorem

PLANEX is PSPACE-complete for $\text{STRIPS}_N$, $\text{STRIPS}_{C,B}$, and for all formalisms “between” the two.

Proof.

Follows from theorems proved in the previous lecture.
Consider **mappings** between planning problems in different formalisms

- that **preserve**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- that **are limited**
  - in the *size* of the result (poly. size)
  - in the *computational resources* (poly. time)

- that **transform**
  - entire planning instances
  - domain structure and states in isolation
Measuring Expressive Power

Consider **mappings** between planning problems in different formalisms

- that **preserve**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- that **are limited**
  - in the *size* of the result (poly. size)
  - in the *computational resources* (poly. time)

- that **transform**
  - entire planning instances
  - domain structure and states in isolation
Measuring Expressive Power

Consider **mappings** between planning problems in different formalisms

- that **preserve**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- that **are limited**
  - in the size of the result (poly. size)
  - in the *computational resources* (poly. time)

- that **transform**
  - entire planning instances
  - domain structure and states in isolation
Method 1: Polynomial Transformation

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

→ all formalisms have the same expressiveness (?)
Method 1: Polynomial Transformation

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

→ all formalisms have the same expressiveness (?)
Method 1: Polynomial Transformation

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the *computational resources* (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇒ all formalisms have the **same expressiveness** (?)
Method 1: Polynomial Transformation

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

→ all formalisms have the same expressiveness (?)
Method 2: Bäckström’s ESP-reductions

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

However, **expressiveness** is independent of the computational resources needed to compute the mapping
Method 2: Bäckström’s ESP-reductions

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

However, expressiveness is independent of the computational resources needed to compute the mapping
Method 2: Bäckström’s ESP-reductions

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

~~> However, expressiveness is independent of the computational resources needed to compute the mapping
Method 2: Bäckström’s ESP-reductions

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - **in the computational resources (poly. time)**

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇝ However, expressiveness is independent of the computational resources needed to compute the mapping
Method 3: Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇝ All formalisms are trivially equivalent (because planning is PSPACE-complete for all propositional STRIPS formalisms)
Method 3: Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

~~> All formalisms are trivially equivalent (because planning is PSPACE-complete for all propositional STRIPS formalisms)
Method 3: Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇒ All formalisms are **trivially equivalent** (because planning is PSPACE-complete for all propositional STRIPS formalisms)
Method 3: Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇒ All formalisms are **trivially equivalent** (because planning is PSPACE-complete for all propositional STRIPS formalisms)
Method 4: Modular & Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

When measuring the expressiveness of planning formalisms, domain structures should be considered independently from states.
Method 4: Modular & Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇝ When measuring the expressiveness of planning formalisms, domain structures should be considered independently from states
Method 4: Modular & Polysize Mappings

- **preserving**
  - solution existence
  - **plan size linearly or polynomially etc.**
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the *computational resources* (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

⇝ When measuring the expressiveness of planning formalisms, domain structures should be considered independently from states.
Method 4: Modular & Polysize Mappings

- **preserving**
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan “structure”
  - the solutions/plans themselves

- **limiting**
  - in the size of the result (poly. size)
  - in the computational resources (poly. time)

- **transforming**
  - entire planning instances
  - domain structure and states in isolation

When measuring the expressiveness of planning formalisms, domain structures should be considered independently from states.
The Right Method: Compilation Schemes (Simplified)

- Transform **domain structure** 
  \( D = \langle A, O \rangle \) (with polynomial blowup) to \( D' \) preserving solution existence
- Only trivial changes to **states** (independent of operator set)
- Resulting **plans** \( \pi' \) should not grow too much (additive constant, linear growth, polynomial growth)

\( \Rightarrow \) Similar to **knowledge compilation**, with operators as the **fixed part** and initial states & goals as the **varying part**
The Right Method: Compilation Schemes (Simplified)

- Transform **domain structure** $\mathcal{D} = \langle A, O \rangle$ (with polynomial blowup) to $\mathcal{D}'$ preserving solution existence
- Only trivial changes to **states** (independent of operator set)
- Resulting **plans** $\pi'$ should not grow too much (additive constant, linear growth, polynomial growth)

→ Similar to **knowledge compilation**, with operators as the **fixed part** and initial states & goals as the **varying part**
The Right Method: Compilation Schemes (Simplified)

- Transform **domain structure** $\mathcal{D} = \langle A, O \rangle$ (with polynomial blowup) to $\mathcal{D}'$ preserving solution existence
- Only trivial changes to **states** (independent of operator set)
- Resulting **plans** $\pi'$ should not grow too much (additive constant, linear growth, polynomial growth)

⇝ Similar to **knowledge compilation**, with operators as the **fixed part** and initial states & goals as the **varying part**
The Right Method: Compilation Schemes (Simplified)

- Transform **domain structure** $\mathcal{D} = \langle A, O \rangle$ (with polynomial blowup) to $\mathcal{D}'$ preserving solution existence
- Only trivial changes to **states** (independent of operator set)
- Resulting **plans** $\pi'$ should not grow too much (additive constant, linear growth, polynomial growth)

Similarly to **knowledge compilation**, with operators as the **fixed part** and initial states & goals as the **varying part**
Compilability

\( \mathcal{Y} \preceq \mathcal{X} \) (\( \mathcal{Y} \) is compilable to \( \mathcal{X} \))

iff

there exists a compilation scheme from \( \mathcal{Y} \) to \( \mathcal{X} \).

\( \mathcal{Y} \preceq^1 \mathcal{X} \): preserving plan size exactly (modulo additive constants)

\( \mathcal{Y} \preceq^c \mathcal{X} \): preserving plan size linearly (in \(|\pi|\))

\( \mathcal{Y} \preceq^p \mathcal{X} \): preserving plan size polynomially (in \(|\pi|\) and \(|D|\))

\( \mathcal{Y} \preceq^{x,p} \mathcal{X} \): polynomial-time compilability

Theorem

For all \( x, y \), the relations \( \preceq_{x,y} \) are transitive and reflexive.
Compilability

\( \mathcal{Y} \preceq \mathcal{X} \) (\( \mathcal{Y} \) is compilable to \( \mathcal{X} \))

iff

there exists a compilation scheme from \( \mathcal{Y} \) to \( \mathcal{X} \).

\( \mathcal{Y} \preceq^1 \mathcal{X} \): preserving plan size **exactly** (modulo additive constants)

\( \mathcal{Y} \preceq^c \mathcal{X} \): preserving plan size **linearly** (in \(|\pi|\))

\( \mathcal{Y} \preceq^p \mathcal{X} \): preserving plan size **polynomially** (in \(|\pi|\) and \(|\mathcal{D}|\))

\( \mathcal{Y} \preceq_{x^p} \mathcal{X} \): **polynomial-time** compilability

**Theorem**

For all \( x, y \), the relations \( \preceq_{x^y} \) are transitive and reflexive.
$\mathcal{Y} \leq \mathcal{X}$ ($\mathcal{Y}$ is compilable to $\mathcal{X}$) iff there exists a compilation scheme from $\mathcal{Y}$ to $\mathcal{X}$.

- $\mathcal{Y} \leq^1 \mathcal{X}$: preserving plan size exactly (modulo additive constants)
- $\mathcal{Y} \leq^c \mathcal{X}$: preserving plan size linearly (in $|\pi|$)
- $\mathcal{Y} \leq^p \mathcal{X}$: preserving plan size polynomially (in $|\pi|$ and $|D|$)
- $\mathcal{Y} \leq^{xp} \mathcal{X}$: polynomial-time compilability

Theorem

For all $x, y$, the relations $\leq^x_y$ are transitive and reflexive.
Y ≤ X (Y is compilable to X) iff
there exists a compilation scheme from Y to X.

Y ≤_1^X: preserving plan size **exactly** (modulo additive constants)
Y ≤_c^X: preserving plan size **linearly** (in |π|)
Y ≤_p^X: preserving plan size **polynomially** (in |π| and |D|)
Y ≤_x^p, X: **polynomial-time** compilability

**Theorem**

For all x, y, the relations ≤_x^y are transitive and reflexive.
\( \mathcal{Y} \preceq \mathcal{X} \) (\( \mathcal{Y} \) is compilable to \( \mathcal{X} \))

iff

does not exist a compilation scheme from \( \mathcal{Y} \) to \( \mathcal{X} \).

\( \mathcal{Y} \preceq^1 \mathcal{X} \): preserving plan size **exactly** (modulo additive constants)

\( \mathcal{Y} \preceq^c \mathcal{X} \): preserving plan size **linearly** (in \(|\pi|\))

\( \mathcal{Y} \preceq^p \mathcal{X} \): preserving plan size ** polynomially** (in \(|\pi|\) and \(|D|\))

\( \mathcal{Y} \preceq^x_p \mathcal{X} \): **polynomial-time** compilability

**Theorem**

*For all \( x, y \), the relations \( \preceq^x_y \) are transitive and reflexive.*
Back-Translatability

- Shouldn’t we also require that plans in the compiled instance can be *translated back* to the original formalism?
- Yes, if we want to use this technique, one should require that!
- In all *positive cases*, there was never any problem to translate the plan back.
- For the *negative case*, it is easier to prove *non-existence*
- So, in order to prove negative results, we do not need it, for positive it never had been a problem.

⇝ So, similarly to the concentration on *decision problems* when determining complexity, we simplify things here.
Shouldn’t we also require that plans in the compiled instance can be *translated back* to the original formalism?

Yes, if we want to use this technique, one should require that!

In all *positive cases*, there was never any problem to translate the plan back.

For the *negative case*, it is easier to prove *non-existence*.

So, in order to prove negative results, we do not need it, for positive it never had been a problem.

So, similarly to the concentration on *decision problems* when determining complexity, we simplify things here.
Shouldn’t we also require that plans in the compiled instance can be *translated back* to the original formalism?

Yes, if we want to use this technique, one should require that!

In all *positive cases*, there was never any problem to translate the plan back.

For the *negative case*, it is easier to prove *non-existence*.

So, in order to prove negative results, we do not need it, for positive it never had been a problem.

So, similarly to the concentration on *decision problems* when determining complexity, we simplify things here.
Shouldn’t we also require that plans in the compiled instance can be translated back to the original formalism?

Yes, if we want to use this technique, one should require that!

In all positive cases, there was never any problem to translate the plan back.

For the negative case, it is easier to prove non-existence.

So, in order to prove negative results, we do not need it, for positive it never had been a problem.

So, similarly to the concentration on decision problems when determining complexity, we simplify things here.
Shouldn’t we also require that plans in the compiled instance can be *translated back* to the original formalism?

Yes, if we want to use this technique, one should require that!

In all *positive cases*, there was never any problem to translate the plan back.

For the *negative case*, it is easier to prove *non-existence*.

So, in order to prove negative results, we do not need it, for positive it never had been a problem.

So, similarly to the concentration on *decision problems* when determining complexity, we simplify things here.
Back-Translatability

- Shouldn’t we also require that plans in the compiled instance can be *translated back* to the original formalism?
- Yes, if we want to use this technique, one should require that!
- In all *positive cases*, there was never any problem to translate the plan back
- For the *negative case*, it is easier to prove *non-existence*
- So, in order to prove negative results, we do not need it, for positive it never had been a problem

So, similarly to the concentration on *decision problems* when determining complexity, we simplify things here.
A (Trivial) Positive Result: $\text{STRIPS}_{Bd} \preceq_1^p \text{STRIPS}_N$

DNF preconditions can be "compiled away."

Assume operator $o = \langle c, e \rangle$ and

$$c = L_1 \lor \ldots \lor L_k$$

with $L_i$ being a conjunction of literals. Create $k$ operators $o_i = \langle L_i, e \rangle$

1. compilation is solution-preserving,
2. $D'$ is only polynomially larger than $D$,
3. compilation can be computed in polynomial time,
4. resulting plans do not grow at all.

$\Rightarrow$ $\text{STRIPS}_{Bd} \preceq_1^p \text{STRIPS}_N$
A (Trivial) Positive Result: $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$

DNF preconditions can be “compiled away.”

Assume operator $o = \langle c, e \rangle$ and

$$c = L_1 \lor \ldots \lor L_k$$

with $L_i$ being a conjunction of literals. Create $k$ operators $o_i = \langle L_i, e \rangle$;

- compilation is solution-preserving,
- $D'$ is only polynomially larger than $D$,
- compilation can be computed in polynomial time,
- resulting plans do not grow at all.

$\Rightarrow$ $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$
A (Trivial) Positive Result: $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$

DNF preconditions can be "compiled away."

Assume operator $o = \langle c, e \rangle$ and

$$c = L_1 \lor \ldots \lor L_k$$

with $L_i$ being a conjunction of literals. Create $k$ operators $o_i = \langle L_i, e \rangle$

1. compilation is solution-preserving,
2. $D'$ is only polynomially larger than $D$,
3. compilation can be computed in polynomial time,
4. resulting plans do not grow at all.

$\Rightarrow$ $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$
A (Trivial) Positive Result: \( \text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N \)

DNF preconditions can be "compiled away."

Assume operator \( o = \langle c, e \rangle \) and

\[
c = L_1 \lor \ldots \lor L_k
\]

with \( L_i \) being a conjunction of literals. Create \( k \) operators

\[
o_i = \langle L_i, e \rangle
\]

1. compilation is solution-preserving,
2. \( D' \) is only polynomially larger than \( D \),
3. compilation can be computed in polynomial time,
4. resulting plans do not grow at all.

\( \Rightarrow \) \( \text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N \)
A (Trivial) Positive Result: $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$

DNF preconditions can be "compiled away."

Assume operator $o = \langle c, e \rangle$ and

$$c = L_1 \lor \ldots \lor L_k$$

with $L_i$ being a conjunction of literals. Create $k$ operators $o_i = \langle L_i, e \rangle$

1. compilation is solution-preserving,
2. $D'$ is only polynomially larger than $D$,
3. compilation can be computed in polynomial time,
4. resulting plans do not grow at all.

$\Rightarrow$ $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$
A (Trivial) Positive Result: $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$

DNF preconditions can be “compiled away.”
Assume operator $o = \langle c, e \rangle$ and

$$ c = L_1 \lor \ldots \lor L_k $$

with $L_i$ being a conjunction of literals. Create $k$ operators $o_i = \langle L_i, e \rangle$

1. compilation is solution-preserving,
2. $D'$ is only polynomially larger than $D$,
3. compilation can be computed in polynomial time,
4. resulting plans do not grow at all.

$\Rightarrow$ $\text{STRIPS}_{Bd} \preceq^1_p \text{STRIPS}_N$
A (Trivial) Positive Result: \( \text{STRIPS}_{Bd} \preceq_p^1 \text{STRIPS}_N \)

DNF preconditions can be “compiled away.”
Assume operator \( o = \langle c, e \rangle \) and

\[
c = L_1 \lor \ldots \lor L_k
\]

with \( L_i \) being a conjunction of literals. Create \( k \) operators
\( o_i = \langle L_i, e \rangle \)

1. compilation is solution-preserving,
2. \( D' \) is only polynomially larger than \( D \),
3. compilation can be computed in polynomial time,
4. resulting plans do not grow at all.

\( \implies \) \( \text{STRIPS}_{Bd} \preceq_p^1 \text{STRIPS}_N \)
Another Positive Result: $\text{STRIPS}_{C,Bc} \preceq^p_{C} \text{STRIPS}_{C,N}$

CNF preconditions can be “\textit{compiled away}” – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a \textit{special evaluation operator} with conditional effects that make new “clause atoms” true.
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators.

$\Rightarrow$ Operator sets grow only \textit{polynomially}.

$\Rightarrow$ Plans are \textit{double as long} as the original plans.

$\Rightarrow$ Anderson et al's conjecture holds in a weak version.
CNF preconditions can be "compiled away" – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a special evaluation operator with conditional effects that make new "clause atoms" true.
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators.

$\Rightarrow$ Operator sets grow only polynomially

$\Rightarrow$ Plans are double as long as the original plans

$\Rightarrow$ Anderson et al's conjecture holds in a weak version
Another Positive Result: $\text{STRIPS}_{C,Bc} \preceq_p^C \text{STRIPS}_{C,N}$

CNF preconditions can be “compiled away” – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a special evaluation operator with conditional effects that make new “clause atoms” true.
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators.

$\Rightarrow$ Operator sets grow only polynomially

$\Rightarrow$ Plans are double as long as the original plans

$\Rightarrow$ Anderson et al’s conjecture holds in a weak version.
Another Positive Result: \( \text{STRIPS}_{C,Bc} \preceq^C_p \text{STRIPS}_{C,N} \)

CNF preconditions can be \textbf{“compiled away”} – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a \textbf{special evaluation operator} with conditional effects that make new “clause atoms” true.
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators.

\( \Rightarrow \) Operator sets grow only \textbf{polynomially}.

\( \Rightarrow \) Plans are \textbf{double as long} as the original plans.

\( \Rightarrow \) \textbf{Anderson et al’s conjecture} holds in a weak version.
Another Positive Result: $\text{STRIPS}_{C,Bc} \preceq_p^C \text{STRIPS}_{C,N}$

CNF preconditions can be “\textit{compiled away}” – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a \textit{special evaluation operator} with conditional effects that make new “clause atoms” true
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators

\[\Rightarrow\] Operator sets grow only \textit{polynomially}

\[\Rightarrow\] Plans are \textit{double as long} as the original plans

\[\Rightarrow\] Anderson et al’s conjecture holds in a weak version
Another Positive Result: \( \text{STRIPS}_{C,Bc} \preceq^C_p \text{STRIPS}_{C,N} \)

CNF preconditions can be “compiled away” – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a special evaluation operator with conditional effects that make new “clause atoms” true.
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators.

\( \rightarrow \) Operator sets grow only polynomially.

\( \rightarrow \) Plans are double as long as the original plans.

\( \rightarrow \) Anderson et al’s conjecture holds in a weak version.
CNF preconditions can be "compiled away" – provided we have already conditional effects.

- Evaluate the truth value of all disjunctions appearing in operators by using a **special evaluation operator** with conditional effects that make new “clause atoms” true
- Alternate between executing original operators (clauses replaced by new atoms) and evaluation operators

- Operator sets grow only **polynomially**
- Plans are **double as long** as the original plans

**Anderson et al’s conjecture** holds in a weak version
A First Negative Result: Conditional Effects Cannot be Compiled into Boolean Preconditions

Consider domain \( D \) with only one (STRIPS\(_{C,B} \)) operator \( o \):

\[
\langle \top, (p_1 \triangleright \neg p_1) \land (\neg p_1 \triangleright p_1) \land \ldots \land (p_k \triangleright \neg p_k) \land (\neg p_k \triangleright p_k) \rangle ,
\]

which “inverts” a given state. For all \((I, G)\) with

\[
G = \bigwedge \{ \neg v \mid v \in A, I \models v \} \land \bigwedge \{ v \mid v \in A, I \not\models v \} ,
\]

there exists a STRIPS\(_{C,B} \) one-step plan.

Assume there exists a compilation preserving plan size linearly leading to a STRIPS\(_B \) domain structure \( D' \). There are exponentially many possible initial states, but only polynomially many different \( c \)-step plans for \( D' \). Some STRIPS\(_B \) plan \( \pi \) is used for different initial states \( I_1, I_2 \) (for large enough \( k \)). Let \( v \) be a variable with \( I_1(v) \neq I_2(v) \).

\( \Rightarrow \) In one case, \( v \) must be set by \( \pi \), in the other case, it must be cleared.

\( \Rightarrow \) This is not possible in an unconditional plan.

\( \Rightarrow \) The transformation is not solution preserving!

\( \Rightarrow \) Conditional effects cannot be compiled away (if plan size can grow only linearly)
Consider domain $\mathcal{D}$ with only one (STRIPS$_{C,B}$) operator $o$:

$$\langle \top, (p_1 \triangleright \neg p_1) \land (\neg p_1 \triangleright p_1) \land \ldots \land (p_k \triangleright \neg p_k) \land (\neg p_k \triangleright p_k) \rangle,$$

which “inverts” a given state. For all $(I, G)$ with

$$G = \bigwedge \{ \neg v \mid v \in A, I \models v \} \land \bigwedge \{ v \mid v \in A, I \nvdash v \},$$

there exists a STRIPS$_{C,B}$ one-step plan.

Assume there exists a compilation preserving plan size linearly leading to a STRIPS$_B$ domain structure $\mathcal{D}'$. There are exponentially many possible initial states, but only polynomially many different $c$-step plans for $\mathcal{D}'$. Some STRIPS$_B$ plan $\pi$ is used for different initial states $I_1, I_2$ (for large enough $k$). Let $v$ be a variable with $I_1(v) \neq I_2(v)$.

$\implies$ In one case, $v$ must be set by $\pi$, in the other case, it must be cleared.

$\implies$ This is not possible in an unconditional plan.

$\implies$ The transformation is not solution preserving!

$\implies$ Conditional effects cannot be compiled away (if plan size can grow only linearly)
A First Negative Result: Conditional Effects Cannot be Compiled into Boolean Preconditions

Consider domain $\mathcal{D}$ with only one (STRIPS$_{C,B}$) operator $o$:

$$
\langle \top, (p_1 \triangleright \neg p_1) \land (\neg p_1 \triangleright p_1) \land \ldots \land (p_k \triangleright \neg p_k) \land (\neg p_k \triangleright p_k) \rangle,
$$

which “inverts” a given state. For all $(I, G)$ with

$$
G = \bigwedge \{ \neg v \mid v \in A, I \models v \} \land \bigwedge \{ v \mid v \in A, I \not\models v \},
$$

there exists a STRIPS$_{C,B}$ one-step plan.

Assume there exists a compilation preserving plan size linearly leading to a STRIPS$_B$ domain structure $\mathcal{D}'$. There are exponentially many possible initial states, but only polynomially many different $c$-step plans for $\mathcal{D}'$. Some STRIPS$_B$ plan $\pi$ is used for different initial states $I_1, I_2$ (for large enough $k$). Let $v$ be a variable with $I_1(v) \not\equiv I_2(v)$.

$\Rightarrow$ In one case, $v$ must be set by $\pi$, in the other case, it must be cleared.

$\Rightarrow$ This is not possible in an unconditional plan.

$\Rightarrow$ The transformation is not solution preserving!

$\Rightarrow$ Conditional effects cannot be compiled away (if plan size can grow only linearly)
Consider domain $\mathcal{D}$ with only one (STRIPS$_{C,B}$) operator $o$:

$$\langle \top, (p_1 \triangleright \neg p_1) \land (\neg p_1 \triangleright p_1) \land \ldots \land (p_k \triangleright \neg p_k) \land (\neg p_k \triangleright p_k) \rangle,$$

which “inverts” a given state. For all $(I, G)$ with

$$G = \bigwedge \{ \neg v \mid v \in A, I \models v \} \land \bigwedge \{ v \mid v \in A, I \not\models v \},$$

there exists a STRIPS$_{C,B}$ one-step plan.

Assume there exists a compilation preserving plan size linearly leading to a STRIPS$_B$ domain structure $\mathcal{D}'$. There are exponentially many possible initial states, but only polynomially many different $c$-step plans for $\mathcal{D}'$. Some STRIPS$_B$ plan $\pi$ is used for different initial states $I_1, I_2$ (for large enough $k$). Let $v$ be a variable with $I_1(v) \neq I_2(v)$.

$\Rightarrow$ In one case, $v$ must be set by $\pi$, in the other case, it must be cleared.

$\Rightarrow$ This is not possible in an unconditional plan.

$\Rightarrow$ The transformation is **not solution preserving**!

$\Rightarrow$ **Conditional effects** cannot be compiled away (if plan size can grow only linearly)
A First Negative Result: Conditional Effects Cannot be Compiled into Boolean Preconditions

Consider domain $\mathcal{D}$ with only one (STRIPS$_{C,B}$) operator $o$:

$$\langle \top, (p_1 \triangleright \neg p_1) \land (\neg p_1 \triangleright p_1) \land \ldots \land (p_k \triangleright \neg p_k) \land (\neg p_k \triangleright p_k) \rangle,$$

which “inverts” a given state. For all $(I, G)$ with

$$G = \bigwedge \{ \neg v \mid v \in A, I \models v \} \land \bigwedge \{ v \mid v \in A, I \not\models v \},$$

there exists a STRIPS$_{C,B}$ one-step plan.

Assume there exists a compilation preserving plan size linearly leading to a STRIPS$_B$ domain structure $\mathcal{D}'$. There are exponentially many possible initial states, but only polynomially many different $c$-step plans for $\mathcal{D}'$. Some STRIPS$_B$ plan $\pi$ is used for different initial states $I_1, I_2$ (for large enough $k$). Let $v$ be a variable with $I_1(v) \neq I_2(v)$.

$\Rightarrow$ In one case, $v$ must be set by $\pi$, in the other case, it must be cleared.

$\Rightarrow$ This is not possible in an unconditional plan.

$\Rightarrow$ The transformation is not solution preserving!

$\Rightarrow$ Conditional effects cannot be compiled away (if plan size can grow only linearly)
A First Negative Result: Conditional Effects Cannot be Compiled into Boolean Preconditions

Consider domain $\mathcal{D}$ with only one (STRIPS$_{C,B}$) operator $o$:

$$\langle \top, (p_1 \triangleright -p_1) \land (-p_1 \triangleright p_1) \land \ldots \land (p_k \triangleright -p_k) \land (-p_k \triangleright p_k) \rangle,$$

which “inverts” a given state. For all $(I, G)$ with

$$G = \bigwedge\{ -v \mid v \in A, I \models v \} \land \bigwedge\{ v \mid v \in A, I \not\models v \},$$

there exists a STRIPS$_{C,B}$ one-step plan.

Assume there exists a compilation preserving plan size linearly leading to a STRIPS$_B$ domain structure $\mathcal{D}'$. There are exponentially many possible initial states, but only polynomially many different $c$-step plans for $\mathcal{D}'$. Some STRIPS$_B$ plan $\pi$ is used for different initial states $I_1, I_2$ (for large enough $k$). Let $v$ be a variable with $I_1(v) \neq I_2(v)$.

$\Rightarrow$ In one case, $v$ must be set by $\pi$, in the other case, it must be cleared.

$\Rightarrow$ This is not possible in an unconditional plan.

$\Rightarrow$ The transformation is **not solution preserving**!

$\Rightarrow$ **Conditional effects** cannot be compiled away (if plan size can grow only linearly)
Another Negative Result: $\text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N$

$k$-$\text{FISEX}$: Planning problem with fixed plan length $k$ and varying initial state. Does there exist an initial state leading to a successful $k$-step plan?

1-$\text{FISEX}$ is NP-complete for $\text{STRIPS}_{Bc}$ ($= \text{SAT}$).

$k$-$\text{FISEX}$ is polynomial for $\text{STRIPS}_N$ (regression analysis)

\[ \not\leq^c \]

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out – provided the polynomial hierarchy does not collapse. The proof method uses non-uniform complexity classes such as $P/poly$.

$\Rightarrow$ Bäckström’s conjecture holds in the compilation framework.
Another Negative Result: $\text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N$

$k$-FISEX: Planning problem with fixed plan length $k$ and varying initial state. Does there exist an initial state leading to a successful $k$-step plan?

1-FISEX is NP-complete for $\text{STRIPS}_{Bc}$ ($= \text{SAT}$).

$k$-FISEX is polynomial for $\text{STRIPS}_N$ (regression analysis)

$\leadsto$ $\text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N$ (if $P \neq \text{NP}$)

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out – provided the polynomial hierarchy does not collapse. The proof method uses non-uniform complexity classes such as $P/\text{poly}$.

$\leadsto$ Bäckström’s conjecture holds in the compilation framework.
Another Negative Result: \( \text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N \)

\textbf{\( k \)-FISEX}: Planning problem with fixed plan length \( k \) and varying initial state. Does there exist an initial state leading to a successful \( k \)-step plan?

1-FISEX is NP-complete for \( \text{STRIPS}_{Bc} (= \text{SAT}) \).

\( k \)-FISEX is polynomial for \( \text{STRIPS}_N \) (regression analysis)

\[ \sim \Rightarrow \text{STRIPS}_{Bc} \not\leq^c_p \text{STRIPS}_N \text{ (if } P \neq \text{NP)} \]

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out – provided the polynomial hierarchy does not collapse. The proof method uses \textit{non-uniform complexity classes} such as \( P/poly \).

\[ \sim \Rightarrow \text{Bäckström's conjecture holds} \text{ in the compilation framework.} \]
Another Negative Result: \( \text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N \)

**k-FISEX**: Planning problem with fixed plan length \( k \) and varying initial state. Does there exist an initial state leading to a successful \( k \)-step plan?

1-FISEX is NP-complete for \( \text{STRIPS}_{Bc} (= \text{SAT}) \).

\( k \)-FISEX is polynomial for \( \text{STRIPS}_N \) (regression analysis)

\[ \leadsto \text{STRIPS}_{Bc} \not\leq^c_p \text{STRIPS}_N \text{ (if P \neq NP)} \]

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out – provided the polynomial hierarchy does not collapse. The proof method uses non-uniform complexity classes such as \( P/\text{poly} \).

\[ \leadsto \text{B"ackstr"om’s conjecture holds} \text{ in the compilation framework.} \]
Another Negative Result: $\text{STRIPS}_{Bc} \not\leq_c \text{STRIPS}_N$

$k$-FISEX: Planning problem with fixed plan length $k$ and varying initial state. Does there exist an initial state leading to a successful $k$-step plan?

1-FISEX is NP-complete for $\text{STRIPS}_{Bc}$ (\(\equiv\) SAT).

$k$-FISEX is polynomial for $\text{STRIPS}_N$ (regression analysis)

\[ \sim \implies \text{STRIPS}_{Bc} \not\leq_c \text{STRIPS}_N \text{ (if P} \not= \text{NP)} \]

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out – provided the polynomial hierarchy does not collapse. The proof method uses non-uniform complexity classes such as $P/poly$.

\[ \sim \implies \text{Bäckström’s conjecture holds} \text{ in the compilation framework.} \]
Another Negative Result: \( \text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N \)

**k-FISEX**: Planning problem with fixed plan length \( k \) and varying initial state. Does there exist an initial state leading to a successful \( k \)-step plan?

1-FISEX is NP-complete for \( \text{STRIPS}_{Bc} (= \text{SAT}) \).

\( k \)-FISEX is polynomial for \( \text{STRIPS}_N \) (regression analysis)

\[ \sim \text{STRIPS}_{Bc} \not\leq^c \text{STRIPS}_N \text{ (if P} \neq \text{NP)} \]

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out – provided the polynomial hierarchy does not collapse. The proof method uses non-uniform complexity classes such as \( P/poly \).

\[ \sim \text{Bäckström’s conjecture holds} \text{ in the compilation framework.} \]
Another Negative Result: $\text{STRIPS}_{Bc} \not\subseteq^c \text{STRIPS}_N$

$k$-FISEX: Planning problem with fixed plan length $k$ and varying initial state. Does there exist an initial state leading to a successful $k$-step plan?

1-FISEX is NP-complete for $\text{STRIPS}_{Bc}$ ($= \text{SAT}$).

$k$-FISEX is polynomial for $\text{STRIPS}_N$ (regression analysis)

$$\leadsto \text{STRIPS}_{Bc} \not\subseteq^p \text{STRIPS}_N \text{ (if } P \neq \text{NP})$$

Using a technique first used by Kautz & Selman, one can show that even arbitrary compilations can be ruled out — provided the polynomial hierarchy does not collapse. The proof method uses non-uniform complexity classes such as $P/poly$.

$$\leadsto \text{Bäckström’s conjecture holds}$$ in the compilation framework.
A Final Negative Result: Boolean Preconditions Cannot be Compiled Away Even in the Presence of Conditional Effects

- Boolean preconditions have the power of **families of Boolean circuits with logarithmic depth** (because Boolean formula have this power) ($= \text{NC}^1$).
- Conditional effects can simulate only **families of circuits with fixed depth** ($= \text{AC}^0$).
- The parity function can be expressed in the first framework ($\text{NC}^1$) while it cannot be expressed in the second ($\text{AC}^0$).

$\rightarrow$ The negative result follows **unconditionally!**
A Final Negative Result: Boolean Preconditions Cannot be Compiled Away Even in the Presence of Conditional Effects

- Boolean preconditions have the power of **families of Boolean circuits with logarithmic depth** (because Boolean formula have this power) \((= \text{NC}^1)\).

- Conditional effects can simulate only **families of circuits with fixed depth** \((= \text{AC}^0)\).

- The parity function can be expressed in the first framework \((\text{NC}^1)\) while it cannot be expressed in the second \((\text{AC}^0)\).

\(\Rightarrow\) The negative result follows **unconditionally**!
A Final Negative Result: Boolean Preconditions Cannot be Compiled Away Even in the Presence of Conditional Effects

- Boolean preconditions have the power of families of **Boolean circuits with logarithmic depth** (because Boolean formula have this power) \( (= \text{NC}^1) \)
- Conditional effects can simulate only families of circuits with fixed depth \( (= \text{AC}^0) \).
- The parity function can be expressed in the first framework \( (\text{NC}^1) \) while it cannot be expressed in the second \( (\text{AC}^0) \).

\[ \Rightarrow \] The negative result follows unconditionally!
A Final Negative Result: Boolean Preconditions Cannot be Compiled Away Even in the Presence of Conditional Effects

- Boolean preconditions have the power of **families of Boolean circuits with logarithmic depth** (because Boolean formula have this power) ($= \text{NC}^1$).

- Conditional effects can simulate only **families of circuits with fixed depth** ($= \text{AC}^0$).

- The parity function can be expressed in the first framework ($\text{NC}^1$) while it cannot be expressed in the second ($\text{AC}^0$).

$\Rightarrow$ The negative result follows **unconditionally!**
We know what **Boolean circuits** are (directed, acyclic graphs with different types of nodes: *and*, *or*, *not*, *input*, *output*)

- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate

When we want to recognize formal languages with circuits, we need a sequence of circuits with an increasing number of input gates $\leadsto$ **family of circuits**

Families with polynomial size and poly-log ($\log^k n$) depth complexity classes $\text{NC}^k$ (Nick’s class)

$\text{NC} = \bigcup_k \text{NC}^k \subseteq P$, the class of problems that can be solved efficiently in parallel

The class of languages that can be characterized by polynomially sized Boolean formulae is identical to $\text{NC}^1$
**Boolean Circuits**

- We know what **Boolean circuits** are (directed, acyclic graphs with different types of nodes: *and, or, not, input, output*)
- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate
- When we want to **recognize formal languages** with circuits, we need a **sequence of circuits** with an increasing number of input gates \(\leadsto\) **family of circuits**
- Families with polynomial size and poly-log (\(\log^k n\)) depth
- Complexity classes \(\text{NC}^k\) (Nick’s class)
- \(\text{NC} = \bigcup_k \text{NC}^k \subseteq P\), the class of problems that can be solved efficiently in parallel
- The class of languages that can be characterized by polynomially sized Boolean formulae is identical to \(\text{NC}^1\)
We know what Boolean circuits are (directed, acyclic graphs with different types of nodes: *and, or, not, input, output*).

- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate

When we want to recognize formal languages with circuits, we need a sequence of circuits with an increasing number of input gates $\Rightarrow$ family of circuits.

Families with polynomial size and poly-log ($\log^k n$) depth complexity classes $NC^k$ (Nick’s class).

$NC = \bigcup_k NC^k \subseteq P$, the class of problems that can be solved efficiently in parallel.

The class of languages that can be characterized by polynomially sized Boolean formulae is identical to $NC^1$. 
Boolean Circuits

- We know what **Boolean circuits** are (directed, acyclic graphs with different types of nodes: *and, or, not, input, output*)
- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate
- When we want to **recognize formal languages** with circuits, we need a **sequence of circuits** with an increasing number of input gates $\leadsto$ **family of circuits**
  - Families with polynomial size and poly-log ($\log^k n$) depth
  - complexity classes $\text{NC}^k$ (Nick’s class)
  - $\text{NC} = \bigcup_k \text{NC}^k \subseteq P$, the class of problems that can be solved efficiently in parallel
  - The class of languages that can be characterized by polynomially sized Boolean formulae is identical to $\text{NC}^1$
Boolean Circuits

- We know what **Boolean circuits** are (directed, acyclic graphs with different types of nodes: *and*, *or*, *not*, *input*, *output*)
- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate
- When we want to **recognize formal languages** with circuits, we need a **sequence of circuits** with an increasing number of input gates $\leadsto$ **family of circuits**
- Families with polynomial size and poly-log $(\log^k n)$ depth
  - complexity classes **$\text{NC}^k$** (Nick’s class)
  - **$\text{NC} = \bigcup_k \text{NC}^k \subseteq P$**, the class of problems that can be solved efficiently in parallel
- The class of languages that can be characterized by polynomially sized Boolean formulae is identical to **$\text{NC}^1$**
Boolean Circuits

- We know what **Boolean circuits** are (directed, acyclic graphs with different types of nodes: *and*, *or*, *not*, *input*, *output*)
- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate
- When we want to *recognize formal languages* with circuits, we need a *sequence of circuits* with an increasing number of input gates $\leadsto$ *family of circuits*
- Families with polynomial size and poly-log ($\log^k n$) depth
- complexity classes $\text{NC}^k$ (Nick’s class)
- $\text{NC} = \bigcup_k \text{NC}^k \subseteq P$, the class of problems that can be solved efficiently in parallel
- The class of languages that can be characterized by polynomially sized Boolean formulae is identical to $\text{NC}^1$
Boolean Circuits

- We know what **Boolean circuits** are (directed, acyclic graphs with different types of nodes: *and*, *or*, *not*, *input*, *output*)
- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate
- When we want to recognize formal languages with circuits, we need a *sequence of circuits* with an increasing number of input gates ↦ *family of circuits*
- Families with polynomial size and poly-log \((\log^k n)\) depth complexity classes \(NC^k\) (Nick’s class)
- \(NC = \bigcup_k NC^k \subseteq P\), the class of problems that can be solved efficiently in parallel
  - The class of languages that can be characterized by polynomially sized Boolean formulae is identical to \(NC^1\)
Boolean Circuits

- We know what Boolean circuits are (directed, acyclic graphs with different types of nodes: and, or, not, input, output)
- **Size of circuit** = number of gates
- **Depth of circuit** = length of longest path from input gate to output gate
- When we want to recognize formal languages with circuits, we need a sequence of circuits with an increasing number of input gates \( \rightsquigarrow \) family of circuits
- Families with polynomial size and poly-log \( (\log^k n) \) depth complexity classes \( \text{NC}^k \) (Nick’s class)
- \( \text{NC} = \bigcup_k \text{NC}^k \subseteq P \), the class of problems that can be solved efficiently in parallel
- The class of languages that can be characterized by polynomially sized Boolean formulae is identical to \( \text{NC}^1 \)
The classes $\text{AC}^k$

- The classes $\text{NC}^k$ are defined with a fixed fan-in.
- If we have *unbounded fan-in*, we get the classes $\text{AC}^k$.
  - gate types: NOT, $n$-ary AND, $n$-ary OR for all $n \geq 2$
- Obviously: $\text{NC}^k \subseteq \text{AC}^k$
- Possible to show: $\text{AC}^{k-1} \subseteq \text{NC}^k$
- The *parity language* is in $\text{NC}^1$, but not in $\text{AC}^0$!
The classes $\text{AC}^k$

- The classes $\text{NC}^k$ are defined with a fixed fan-in.
- If we have *unbounded fan-in*, we get the classes $\text{AC}^k$.
  - gate types: NOT, $n$-ary AND, $n$-ary OR for all $n \geq 2$
- Obviously: $\text{NC}^k \subseteq \text{AC}^k$
- Possible to show: $\text{AC}^{k-1} \subseteq \text{NC}^k$
- The *parity language* is in $\text{NC}^1$, but not in $\text{AC}^0$!
The classes $\mathbf{AC}^k$

- The classes $\mathbf{NC}^k$ are defined with a fixed fan-in.
- If we have *unbounded fan-in*, we get the classes $\mathbf{AC}^k$.
  - Gate types: NOT, $n$-ary AND, $n$-ary OR for all $n \geq 2$.
- Obviously: $\mathbf{NC}^k \subseteq \mathbf{AC}^k$.
- Possible to show: $\mathbf{AC}^{k-1} \subseteq \mathbf{NC}^k$.
- The *parity language* is in $\mathbf{NC}^1$, but not in $\mathbf{AC}^0$!
The classes $\text{AC}^k$

- The classes $\text{NC}^k$ are defined with a fixed fan-in
- If we have *unbounded fan-in*, we get the classes $\text{AC}^k$
  - gate types: NOT, $n$-ary AND, $n$-ary OR for all $n \geq 2$
- Obviously: $\text{NC}^k \subseteq \text{AC}^k$
- Possible to show: $\text{AC}^{k-1} \subseteq \text{NC}^k$
- The *parity language* is in $\text{NC}^1$, but not in $\text{AC}^0$!
The classes $\text{AC}^k$

- The classes $\text{NC}^k$ are defined with a fixed fan-in.
- If we have \textit{unbounded fan-in}, we get the classes $\text{AC}^k$:
  - gate types: NOT, \textit{n}-ary AND, \textit{n}-ary OR for all $n \geq 2$
- Obviously: $\text{NC}^k \subseteq \text{AC}^k$
- Possible to show: $\text{AC}^{k-1} \subseteq \text{NC}^k$
- The \textit{parity language} is in $\text{NC}^1$, but not in $\text{AC}^0$!
The classes $\text{AC}^k$

- The classes $\text{NC}^k$ are defined with a fixed fan-in
- If we have *unbounded fan-in*, we get the classes $\text{AC}^k$
  - gate types: NOT, $n$-ary AND, $n$-ary OR for all $n \geq 2$
- Obviously: $\text{NC}^k \subseteq \text{AC}^k$
- Possible to show: $\text{AC}^{k-1} \subseteq \text{NC}^k$
- The *parity language* is in $\text{NC}^1$, but not in $\text{AC}^0$!
Accepting languages with families of domain structures with fixed goals

- We will view *families of domain structures* with fixed goals and fixed size plans as “machines” that accept languages.
- Consider families of poly-sized domain structures in $\text{STRIPS}_B$ and use one-step plans for acceptance.
- Obviously, this is the same as using Boolean formulae.
- All languages in $\text{NC}^1$ can be accepted in this way.
We will view \textit{families of domain structures} with fixed goals and fixed size plans as “machines” that accept languages.

Consider families of poly-sized domain structures in STRIPS$_B$ and use one-step plans for acceptance.

Obviously, this is the same as using Boolean formulae.

$\longrightarrow$ All languages in $\mathsf{NC}^1$ can be accepted in this way.
We will view families of domain structures with fixed goals and fixed size plans as “machines” that accept languages.

Consider families of poly-sized domain structures in STRIPS\(_B\) and use one-step plans for acceptance.

Obviously, this is the same as using Boolean formulae. All languages in NC\(^1\) can be accepted in this way.
Accepting languages with families of domain structures with fixed goals

- We will view *families of domain structures* with fixed goals and fixed size plans as “machines” that accept languages.
- Consider families of poly-sized domain structures in STRIPS$_B$ and use one-step plans for acceptance.
- Obviously, this is the same as using Boolean formulae.

$\implies$ All languages in $\text{NC}^1$ can be accepted in this way.
Represent each operator and then chain the actions together ($O(|O|^c)$ different plans):

Simulating $\text{STRIPS}_{C,N}$ $c$-Step Plans with $AC^0$ circuits (1)
Simulating $\text{STRIPS}_{C,N}$ $c$-Step Plans with $\text{AC}^0$ circuits (2)

- For each single action (precondition testing (a), conditional effects (b), and the computation of effects (c)
Theorem

\text{STRIPS}_B \not\leq^c \text{STRIPS}_{C,N}.

Proof.

Assuming \text{STRIPS}_B \leq^c \text{STRIPS}_{C,N} has the consequence that the underlying compilation scheme could be used to compile a \( \text{NC}^1 \) circuit family into an \( \text{AC}^0 \) circuit family, which is impossible in the general case.
Theorem

\[ \text{STRIPS}_B \not \leq^c \text{STRIPS}_{C,N} \].

Proof.

Assuming \( \text{STRIPS}_B \leq^c \text{STRIPS}_{C,N} \) has the consequence that the underlying compilation scheme could be used to compile a \( \text{NC}^1 \) circuit family into an \( \text{AC}^0 \) circuit family, which is impossible in the general case.
General Results for Compilability
Preserving Plan Size Linearly

All other potential positive results have been ruled out by our 3 negative results and transitivity.
Compilation schemes seem to be the right method to measure the relative expressive power of planning formalisms. Either we get a positive result preserving plan size linearly with a polynomial-time compilation or we get an impossibility result.

Results are relevant for building planning systems. CNF preconditions do not add much when we have already conditional effects.

Note: In all cases we can get a positive result if we allow for a polynomial blow-up of the plans.
Compilation schemes seem to be the right method to measure the relative expressive power of planning formalisms.

Either we get a positive result preserving plan size linearly with a polynomial-time compilation or we get an impossibility result.

Results are relevant for building planning systems.

CNF preconditions do not add much when we have already conditional effects.

Note: In all cases we can get a positive result if we allow for a polynomial blow-up of the plans.
Compilation schemes seem to be the right method to measure the relative expressive power of planning formalisms.

Either we get a positive result preserving plan size linearly with a polynomial-time compilation or we get an impossibility result.

Results are relevant for building planning systems.

CNF preconditions do not add much when we have already conditional effects.

Note: In all cases we can get a positive result if we allow for a polynomial blow-up of the plans.
Compilation schemes seem to be the right method to measure the *relative expressive power* of planning formalisms.

Either we get a positive result preserving plan size linearly with a *polynomial-time compilation* or we get an *impossibility result*.

→ **Results are relevant for building planning systems**

~~> **CNF preconditions** do not add much when we have already conditional effects.

Note: In all cases we can get a positive result if we allow for a polynomial blow-up of the plans.
Compilation schemes seem to be the right method to measure the *relative expressive power* of planning formalisms.

Either we get a positive result preserving plan size **linearly** with a **polynomial-time compilation**

or we get an **impossibility result**

→ **Results are relevant for building planning systems**

⇈ **CNF preconditions** do not add much when we have already conditional effects

Note: In all cases we can get a positive result if we allow for a polynomial blow-up of the plans.
Compilation schemes seem to be the right method to measure the relative expressive power of planning formalisms.

Either we get a positive result preserving plan size linearly with a polynomial-time compilation.

or we get an impossibility result.

→ Results are relevant for building planning systems.

¬¬ CNF preconditions do not add much when we have already conditional effects.

Note: In all cases we can get a positive result if we allow for a polynomial blow-up of the plans.