Principles of AI Planning

18. Expressive power

Bernhard Nebel and Robert Mattmüller

Albert-Ludwigs-Universität Freiburg

February 14th, 2012

Principles of AI Planning

February 14th, 2012 — 18. Expressive power

1 Motivation

- Why?
- Examples

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*?

$\rightsquigarrow$ *Compiling away* or change planning algorithm?

$\rightarrow$ 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.

$\rightsquigarrow$ 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 $c_1 \vee \ldots \vee c_n$ into n operators with the same effects and c_i as preconditions

$\rightsquigarrow$ 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: Conditional effects

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

$\rightarrow$ Can this be avoided?

2 Propositional STRIPS and Variants

- Disjunctive Preconditions: Difficult or Easy?
- STRIPS Variants
- Partially Ordered STRIPS Variants
- Computational Complexity

Propositional STRIPS and Variants

- In the following we will only consider propositional STRIPS and some variants of it.
- Planning task:
$$\mathcal{T} = \langle A, I, O, G \rangle.$$
- Often we refer to domain structures $\mathcal{D} = \langle A, O \rangle$.

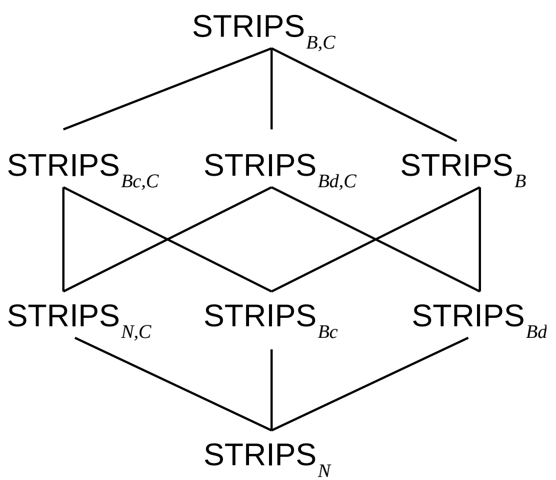
Disjunctive Preconditions: Trivial or Essential?

- 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" $\rightsquigarrow$ conditional effects imply disjunctive preconditions (?) (General Boolean preconditions)

More "Expressive Power"

STRIPS_N	:	plain strips with negative literals
STRIPS_{Bd}	:	precondition in disjunctive normal form
STRIPS_{Bc}	:	precondition in conjunctive normal form
STRIPS_B	:	Boolean expressions as preconditions
STRIPS_C	:	conditional effects
$\text{STRIPS}_{C,N}$	:	conditional effects & negative literals
...		

Ordering Planning Formalisms Partially



Computational Complexity ...

Theorem
PLANEX is PSPACE-complete for STRIPS_N, $\text{STRIPS}_{C,B}$, and for all formalisms "between" the two.

Proof.
Follows from theorems proved in the previous lecture. □

3 Expressive Power

- Measuring Expressive Power
- Compilation Schemes
- Compilability
- Positive Results
- Negative Results
- Using Circuit Complexity ...
- General Compilability Results

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

$\rightsquigarrow$ 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

$\rightsquigarrow$ *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

$\leadsto$ All formalisms are trivially equivalent (because planning is PSPACE-complete for all propositional STRIPS formalisms)

Method 4: Modular & Polysize Mappings

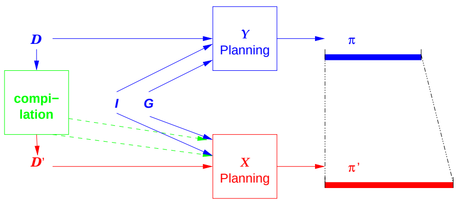
- preserving
  - solution existence
  - plan size linearly or polynomially etc.
  - the exact plan size
  - the plan "structure"
  - the solutions/plans themselves
- **limiting**
  - in the *size* of the result (poly. size)
  - in the *computational resources* (poly. time)
- **transforming**
  - entire planning instances
  - domain structure and states in isolation

$\leadsto$ When measuring the expressiveness of planning formalisms, domain structures should be considered independently from states

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 π' should not grow too much (additive constant, linear growth, polynomial growth)

$\leadsto$ Similar 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 $|\mathcal{D}|$)

$\mathcal{Y} \preceq^x_p \mathcal{X}$: polynomial-time compilability

Theorem

For all x, y, the relations $\preceq^x_y$ are transitive and reflexive.

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

$\rightsquigarrow$ So, similarly to the concentration on decision problems when determining complexity, we simplify things here

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

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

$$c = L_1 \vee \ldots \vee L_k$$

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

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

$\rightsquigarrow$ $\mathrm{STRIPS}_{Bd} \preceq^1_p \mathrm{STRIPS}_N$

Another Positive Result: $\mathrm{STRIPS}_{C,Bc} \preceq^c_p \mathrm{STRIPS}_{C,N}$

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

$\rightsquigarrow$ Operator sets grow only **polynomially**

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

$\rightsquigarrow$ Anderson et al's conjecture holds in a weak version

A First Negative Result: Conditional Effects Cannot be Compiled into Boolean Preconditions

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

$$\langle \top, (p_1 \rhd \neg p_1) \wedge (\neg p_1 \rhd p_1) \wedge \ldots \wedge (p_k \rhd \neg p_k) \wedge (\neg p_k \rhd p_k)\rangle,$$

which "inverts" a given state. For all (I, G) with

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

there exists a $\mathrm{STRIPS}_{C,B}$ 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 π is used for different initial states I_1, I_2 (for large enough k). Let v be a variable with $I_1(v) \neq I_2(v)$.

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

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

$\rightsquigarrow$ The transformation is $\boxed{\textbf{not solution preserving}}$!

$\rightsquigarrow$ **Conditional effects** cannot be compiled away (if plan size can grow only linearly)

Another Negative Result: $\text{STRIPS}_{Bc} \not\preceq^c \text{STRIPS}_N$

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 STRIPS_{Bc} (= SAT).
k-FISEX is polynomial for STRIPS_N (regression analysis)

$$\boxed{\leadsto \text{STRIPS}_{Bc} \not\preceq^c_p \text{STRIPS}_N \text{ (if P}\neq\text{NP)}}$$

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 non-uniform complexity classes such as *P/poly*.

$\leadsto$ *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) (= NC^1)
- Conditional effects can simulate only families of circuits with fixed depth (= AC^0).
- The parity function can be expressed in the first framework (NC^1) while it cannot be expressed in the second (AC^0).

$\leadsto$ The negative result follows unconditionally!

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 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 NC^1

The classes AC^k

- The classes NC^k are defined with a fixed fan-in
- If we have unbounded fan-in, we get the classes 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 NC^1, but not in AC^0!

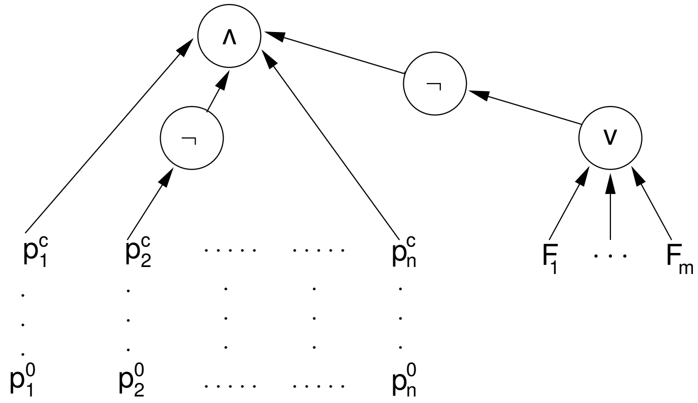
Accepting languages with families of domain structures with fixed goals

- 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 STRIPS_B and use one-step plans for acceptance.
- Obviously, this is the same as using Boolean formulae

$\rightsquigarrow$ All languages in NC^1 can be accepted in this way

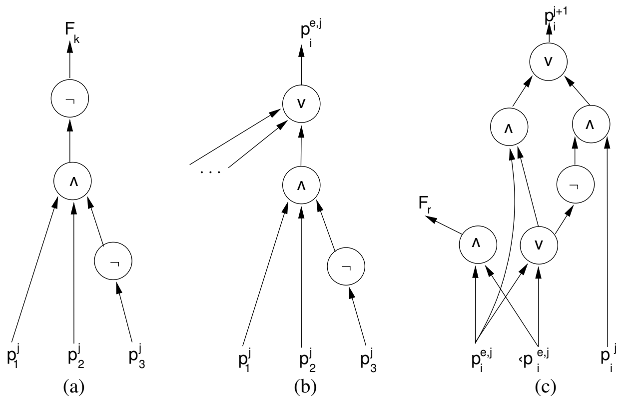
Simulating $\mathrm{STRIPS}_{C,N}$ c-step Plans with AC^0 circuits (1)

- Represent each operator and then chain the actions together ($O(|O|^c)$ different plans):



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

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



$\mathrm{STRIPS}_B \not\preceq^c \mathrm{STRIPS}_{C,N}$

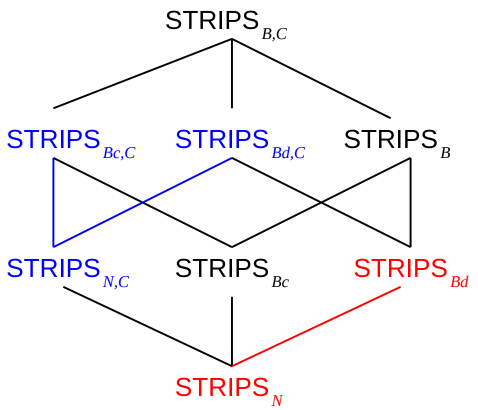
Theorem

$STRIPS_B \not\preceq^c STRIPS_{C,N}$.

Proof.

Assuming $\mathrm{STRIPS}_B \preceq^c \mathrm{STRIPS}_{C,N}$ has the consequence that the underlying compilation scheme could be used to compile a NC^1 circuit family into an 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.

4 Summary

Summary

- 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.