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.

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: Conditional effects

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

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 cannot easily be translated to basic STRIPS (CNF-preconditions)
- Anderson et al [AIPS 98]: “[D]isjunctive preconditions ...are... essential prerequisites for handling conditional effects" \( \rightarrow \) 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
- STRIPS\(_{C,N}\): conditional effects & negative literals

... 

Computational Complexity ...

**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.

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)

- 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)

- 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)

- 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

The Right Method: Compilation Schemes (Simplified)

- Transform domain structure $D = (A, O)$ (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)
  - Similar to knowledge compilation, with operators as the fixed part and initial states & goals as the varying part

Compilability

$Y \preceq X$ ($Y$ is compilable to $X$) iff there exists a compilation scheme from $Y$ to $X$.

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

Theorem

For all $x, y$, the relations $\preceq^x$ 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
**A (Trivial) Positive Result: STRIPS_{Bd} \preceq_p^1 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,
- \( \mathcal{D}' \) is only polynomially larger than \( \mathcal{D} \),
- compilation can be computed in polynomial time,
- resulting plans do not grow at all.
\[\rightsquigarrow \text{STRIPS}_{Bd} \preceq_p^1 \text{STRIPS}_N\]

---

**Another Positive Result: STRIPS_{C,Bc} \preceq_p^c 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
- Alternately between executing original operators (clauses replaced by new atoms) and evaluation operators
\[\rightsquigarrow \text{Operator sets grow only } \text{polynomially}\]
\[\rightsquigarrow \text{Plans are double as long as the original plans}\]
\[\rightsquigarrow \text{Anderson et al’s conjecture holds in a weak version}\]

---

**Negative Result: Conditional Effects Cannot be Compiled into Boolean Preconditions**

Consider domain \( \mathcal{D} \) with only one (\( \text{STRIPS}_{C,B} \)) operator \( o \):
\[
\langle T, (p_1 \lor \neg p_1) \land \neg (p_1 \lor p_1) \land \ldots \land (p_k \lor \neg p_k) \land \neg (\neg p_k \lor p_k) \rangle,
\]
which “inverts” a given state. For all \( (I, G) \) with
\[
G = \bigwedge \{ V : v \in A, I \models v \} \land \bigwedge \{ V : v \in A, I \not\models v \},
\]
there exists a STRIPS_{C,B} one-step plan.
Assume there exists a compilation preserving plan size linearly leading to a STRIPS_{Bd} domain structure \( \mathcal{D}' \). There are exponentially many possible initial states, but only polynomially many different \( c \)-step plans for \( \mathcal{D}' \). Some STRIPS_{Bd} plan \( \pi \) is used for different initial states \( l_1, l_2 \) (for large enough \( k \)). Let \( v \) be a variable with \( l_1(v) \neq l_2(v) \).
- In one case, \( v \) must be set by \( \pi \), in the other case, it must be cleared.
- This is not possible in an unconditional plan.
- The transformation is not solution preserving
\[\rightsquigarrow \text{Conditional effects cannot be compiled away (if plan size can grow only linearly)}\]

---

**Another Negative Result: STRIPS_{Bc} \not\preceq_p^c 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_{Bd} (= SAT).
\( k \)-FISEX is polynomial for STRIPS_N (regression analysis)
\[\rightsquigarrow \text{STRIPS}_{Bc} \not\preceq_p^c \text{STRIPS}_N \text{ (if } P \neq \text{NP)}\]

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 \( \text{P/poly}. \)
\[\rightsquigarrow \text{Bäckström’s conjecture holds in the compilation framework.}\]
Boolean Preconditions Cannot be Compiled Away Even When Using Conditional Effects

- Boolean preconditions have the power of families of Boolean circuits with logarithmic depth (because Boolean formula have this power) (= NC¹)
- Conditional effects can simulate only families of circuits with fixed depth (= AC⁰).
- The parity function can be expressed in the first framework (NC¹) while it cannot be expressed in the second (AC⁰).
  \[ \sim \] 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 \[ \sim \] family of circuits
- Families with polynomial size and poly-log (log⁰k 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 NC¹

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
  \[ \sim \] gate types: NOT, n-ary AND, n-ary OR for all n ≥ 2
- Obviously: NC^k \subseteq AC^k
- Possible to show: AC^{k-1} \subseteq NC^k
- The parity language is in NC¹, but not in AC⁰!

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
  \[ \sim \] All languages in NC¹ can be accepted in this way.
Simulating 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):

\[
\begin{align*}
\text{for } & F_1, \ldots, F_m; \quad p_1, \ldots, p_n; \quad v_1, \ldots, v_k; \\
\text{represent } & \text{each operator and chain together, } \forall j \in \{1, 2, \ldots, m\}.
\end{align*}
\]

February 12th, 2014 B. Nebel, R. Mattmüller – AI Planning

---

Simulating 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))

February 12th, 2014 B. Nebel, R. Mattmüller – AI Planning

---

STRIPS\(_B\) \(\preceq^c\) STRIPS\(_{C,N}\)

Theorem

\(\text{STRIPS}_B \preceq^c \text{STRIPS}_{C,N}\).

Proof.

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

February 12th, 2014 B. Nebel, R. Mattmüller – AI Planning

---

General Results for Compilability

Preserving Plan Size Linearly

All other potential positive results have been ruled out by our 3 negative results and transitivity.

February 12th, 2014 B. Nebel, R. Mattmüller – AI Planning
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.