Principles of AI Planning
5. State-space search: progression and regression

Malte Helmert

Albert-Ludwigs-Universität Freiburg

October 31st, 2008
State-space search

- **state-space search**: one of the big success stories of AI
- many planning algorithms based on state-space search (we’ll see some other algorithms later, though)
- will be the focus of this and the following topics
- we assume prior knowledge of basic search algorithms
  - uninformed vs. informed
  - systematic vs. local
- background on search: Russell & Norvig, Artificial Intelligence – A Modern Approach, chapters 3 and 4
Satisficing or optimal planning?

Must carefully distinguish two different problems:

- **satisficing planning**: any solution is OK (although shorter solutions typically preferred)
- **optimal planning**: plans must have shortest possible length

Both are often solved by search, but:

- details are very different
- almost no overlap between good techniques for satisficing planning and good techniques for optimal planning
- many problems that are trivial for satisficing planners are impossibly hard for optimal planners
Planning by state-space search

How to apply search to planning? \(\rightsquigarrow\) many choices to make!

**Choice 1: Search direction**

- **progression**: forward from initial state to goal
- **regression**: backward from goal states to initial state
- **bidirectional search**
How to apply search to planning? \(\rightsquigarrow\) many choices to make!

Choice 2: Search space representation

- search nodes are associated with states
- search nodes are associated with sets of states
Planning by state-space search

How to apply search to planning? $\leadsto$ many choices to make!

### Choice 3: Search algorithm

- **uninformed search:** depth-first, breadth-first, iterative depth-first, ...
- **heuristic search (systematic):** greedy best-first, A*, Weighted A*, IDA*, ...
- **heuristic search (local):** hill-climbing, simulated annealing, beam search, ...
Planning by state-space search

How to apply search to planning? \( \leadsto \) many choices to make!

Choice 4: Search control

- **heuristics** for informed search algorithms
- **pruning techniques**: invariants, symmetry elimination, helpful actions pruning, ...
Search-based satisficing planners

FF (Hoffmann & Nebel, 2001)

- search direction: forward search
- search space representation: single states
- search algorithm: enforced hill-climbing (informed local)
- heuristic: FF heuristic (inadmissible)
- pruning technique: helpful actions (incomplete)

⇝ one of the best satisficing planners
Search-based optimal planners

Fast Downward + $h_{HHH}$ (Helmert, Haslum & Hoffmann, 2007)

- **search direction**: forward search
- **search space representation**: single states
- **search algorithm**: A* (informed systematic)
- **heuristic**: merge-and-shrink abstractions (admissible)
- **pruning technique**: none

⇝ one of the best optimal planners
Our plan for the next lectures

Choices to make:

1. search direction: progression/regression/both
   ⇝ this chapter

2. search space representation: states/sets of states
   ⇝ this chapter

3. search algorithm: uninformed/heuristic; systematic/local
   ⇝ next chapter

4. search control: heuristics, pruning techniques
   ⇝ following chapters
Planning by forward search: progression

**Progression**: Computing the successor state $app_o(s)$ of a state $s$ with respect to an operator $o$.

**Progression planners** find solutions by forward search:
- start from initial state
- iteratively pick a previously generated state and progress it through an operator, generating a new state
- solution found when a goal state generated

**pro**: very easy and efficient to implement
Two alternative search spaces for progression planners:

1. **search nodes correspond to states**
   - when the same state is generated along different paths, it is not considered again (duplicate detection)
   - **pro:** fast
   - **con:** memory intensive (must maintain closed list)

2. **search nodes correspond to operator sequences**
   - different operator sequences may lead to identical states (transpositions)
   - **pro:** can be very memory-efficient
   - **con:** much wasted work (often exponentially slower)

→ first alternative usually preferable
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Progression planning example (depth-first search)
Forward search vs. backward search

Going through a transition graph in forward and backward directions is **not symmetric**:

- forward search starts from a **single** initial state; backward search starts from a **set** of goal states
- when applying an operator \( o \) in a state \( s \) in forward direction, there is a **unique successor state** \( s' \);
  - if we applied operator \( o \) to end up in state \( s' \), there can be **several possible predecessor states** \( s \)

\[ \rightsquigarrow \]
most natural representation for backward search in planning associates **sets of states** with search nodes
Planning by backward search: regression

Regression: Computing the possible predecessor states $\text{regr}_o(S)$ of a set of states $S'$ with respect to the last operator $o$ that was applied.

Regression planners find solutions by backward search:
- start from set of goal states
- iteratively pick a previously generated state set and regress it through an operator, generating a new state set
- solution found when a generated state set includes the initial state

Pro: can handle many states simultaneously
Con: basic operations complicated and expensive
identify state sets with logical formulae:

- search nodes correspond to state sets
- each state set is represented by a logical formula: 
  $\phi$ represents $\{ s \in S' \mid s \models \phi \}$
- many basic search operations like detecting duplicates are NP-hard or coNP-hard
Regression planning example (depth-first search)
Regression planning example (depth-first search)

\[
\begin{align*}
\phi_1 &= \text{regr}^{-}\rightarrow(G) \\
\phi_2 &= \text{regr}^{-}\rightarrow(\phi_1) \\
\phi_3 &= \text{regr}^{-}\rightarrow(\phi_2) \\
I \mid = \phi_3
\end{align*}
\]
Regression planning example (depth-first search)

\[ \phi_1 = \text{regr} \rightarrow (G) \]

\[ \phi_1 \rightarrow G \]
Regression planning example (depth-first search)

\[
\phi_1 = \text{regr} \rightarrow (G) \\
\phi_2 = \text{regr} \rightarrow (\phi_1)
\]
Regression planning example (depth-first search)

\[\phi_1 = \text{regr}(G)\]
\[\phi_2 = \text{regr}(\phi_1)\]
\[\phi_3 = \text{regr}(\phi_2), I \models \phi_3\]
Regression for STRIPS planning tasks

### Definition (STRIPS planning task)

A planning task is a **STRIPS planning task** if all operators are STRIPS operators and the goal is a conjunction of literals.

### Regression for STRIPS planning tasks

Regression for STRIPS planning tasks is very simple:

- **Goals** are conjunctions of literals $l_1 \land \cdots \land l_n$.
- **First step**: Choose an operator that makes some of $l_1, \ldots, l_n$ true and makes none of them false.
- **Second step**: Remove goal literals achieved by the operator and add its preconditions.
- **Outcome**: Outcome of regression is again conjunction of literals.
STRIPS regression

**Definition**

Let $\phi = \phi_1 \land \cdots \land \phi_k$, $\gamma = \gamma_1 \land \cdots \land \gamma_n$ and $\eta = \eta_1 \land \cdots \land \eta_m$ be non-contradictory conjunctions of literals.

The **STRIPS regression** of $\phi$ with respect to $o = \langle \gamma, \eta \rangle$ is

$$sregr_o(\phi) := \bigwedge (((\{\phi_1, \ldots, \phi_k\} \setminus \{\eta_1, \ldots, \eta_m\}) \cup \{\gamma_1, \ldots, \gamma_n\})$$

provided that this conjunction is non-contradictory and that $\neg \phi_i \not\equiv \eta_j$ for all $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, m\}$.

(Otherwise, $sregr_o(\phi)$ is undefined.)

(A conjunction of literals is contradictory iff it contains two complementary literals.)
STRIPS regression example

NOTE: Predecessor states are in general not unique. This picture is just for illustration purposes.

\[ o_1 = \langle \text{on}, \text{clr}, \neg \text{on}, \text{onT}, \text{clr} \rangle \]
\[ o_2 = \langle \text{on}, \text{clr}, \text{clr}, \neg \text{clr}, \neg \text{on}, \text{onT}, \text{clr} \rangle \]
\[ o_3 = \langle \text{onT}, \text{clr}, \text{clr}, \neg \text{clr}, \neg \text{onT}, \text{on} \rangle \]

\[ G = \text{on}, \text{on} \]
\[ \phi_1 = sregr_{o_3}(G) = \text{on}, \text{onT}, \text{clr}, \text{clr} \]
\[ \phi_2 = sregr_{o_2}(\phi_1) = \text{onT}, \text{clr}, \text{on}, \text{clr} \]
\[ \phi_3 = sregr_{o_1}(\phi_2) = \text{onT}, \text{on}, \text{clr}, \text{on} \]
Regression for general planning tasks

- With disjunctions and conditional effects, things become more tricky. How to regress $A \lor (B \land C)$ with respect to $\langle Q, D \triangleright B \rangle$?
- The story about goals and subgoals and fulfilling subgoals, as in the STRIPS case, is no longer useful.
- We present a general method for doing regression for any formula and any operator.
- Now we extensively use the idea of representing sets of states as formulae.
Effect preconditions

Definition (effect precondition)

The effect precondition $EPC_l(e)$ for literal $l$ and effect $e$ is defined as follows:

$$
EPC_l(l) = \top
$$
$$
EPC_l(l') = \bot \text{ if } l \neq l' \text{ (for literals } l')
$$
$$
EPC_l(e_1 \land \cdots \land e_n) = EPC_l(e_1) \lor \cdots \lor EPC_l(e_n)
$$
$$
EPC_l(c \triangleright e) = EPC_l(e) \land c
$$

Intuition: $EPC_l(e)$ describes the situations in which effect $e$ causes literal $l$ to become true.
Effect precondition examples

Example

\[ EPC_a(b \land c) = \bot \lor \bot \equiv \bot \]
\[ EPC_a(a \land (b \triangleright a)) = \top \lor (\top \land b) \equiv \top \]
\[ EPC_a((c \triangleright a) \land (b \triangleright a)) = (\top \land c) \lor (\top \land b) \equiv c \lor b \]
Effect preconditions: connection to change sets

Lemma (A)

Let $s$ be a state, $l$ a literal and $e$ an effect. Then $l \in [e]_s$ if and only if $s \models EPC_l(e)$.

Proof.

Induction on the structure of the effect $e$.

Base case 1, $e = l$: $l \in [l]_s = \{l\}$ by definition, and $s \models EPC_l(l) = \top$ by definition. Both sides of the equivalence are true.

Base case 2, $e = l'$ for some literal $l' \neq l$: $l \notin [l']_s = \{l'\}$ by definition, and $s \not\models EPC_l(l') = \bot$ by definition. Both sides are false.
Lemma (A)

Let $s$ be a state, $l$ a literal and $e$ an effect. Then $l \in [e]_s$ if and only if $s \models EPC_l(e)$.

Proof.

Induction on the structure of the effect $e$.

Base case 1, $e = l$: $l \in [l]_s = \{l\}$ by definition, and $s \models EPC_l(l) = \top$ by definition. Both sides of the equivalence are true.

Base case 2, $e = l'$ for some literal $l' \neq l$: $l \notin [l']_s = \{l'\}$ by definition, and $s \not\models EPC_l(l') = \bot$ by definition. Both sides are false.
Lemma (A)

Let $s$ be a state, $l$ a literal and $e$ an effect. Then $l \in [e]_s$ if and only if $s \models EPC_l(e)$.

Proof.

Induction on the structure of the effect $e$.

Base case 1, $e = l$: $l \in [l]_s = \{l\}$ by definition, and $s \models EPC_l(l) = \top$ by definition. Both sides of the equivalence are true.

Base case 2, $e = l'$ for some literal $l' \neq l$: $l \notin [l']_s = \{l'\}$ by definition, and $s \not\models EPC_l(l') = \bot$ by definition. Both sides are false.
Effect preconditions: connection to change sets

Proof (ctd.)

Inductive case 1, $e = e_1 \land \cdots \land e_n$:

$l \in [e]_s$ iff $l \in [e_1]_s \cup \cdots \cup [e_n]_s$  

(Def $[e_1 \land \cdots \land e_n]_s$)

iff $l \in [e']_s$ for some $e' \in \{e_1, \ldots, e_n\}$

iff $s \models EPC_l(e')$ for some $e' \in \{e_1, \ldots, e_n\}$ (IH)

iff $s \models EPC_l(e_1) \lor \cdots \lor EPC_l(e_n)$

iff $s \models EPC_l(e_1 \land \cdots \land e_n)$.  

(Def $EPC$)

Inductive case 2, $e = c \triangleright e'$:

$l \in [c \triangleright e']_s$ iff $l \in [e']_s$ and $s \models c$  

(Def $[c \triangleright e']_s$)

iff $s \models EPC_l(e')$ and $s \models c$ (IH)

iff $s \models EPC_l(e') \land c$

iff $s \models EPC_l(c \triangleright e')$.  

(Def $EPC'$)
Effect preconditions: connection to change sets

Proof (ctd.)

Inductive case 1, $e = e_1 \land \cdots \land e_n$:

1. $l \in [e]_s$ iff $l \in [e_1]_s \cup \cdots \cup [e_n]_s$ (Def $[e_1 \land \cdots \land e_n]_s$)
2. $l \in [e']_s$ for some $e' \in \{e_1, \ldots, e_n\}$
3. $s \models EPC_l(e')$ for some $e' \in \{e_1, \ldots, e_n\}$ (IH)
4. $s \models EPC_l(e_1) \lor \cdots \lor EPC_l(e_n)$
5. $s \models EPC_l(e_1 \land \cdots \land e_n)$ (Def $EPC$)

Inductive case 2, $e = c \triangleright e'$:

1. $l \in [c \triangleright e']_s$ iff $l \in [e']_s$ and $s \models c$ (Def $[c \triangleright e']_s$)
2. $s \models EPC_l(e')$ and $s \models c$ (IH)
3. $s \models EPC_l(e') \land c$
4. $s \models EPC_l(c \triangleright e')$. (Def $EPC$)
Remark

Notice that in terms of $EPC_a(e)$, any operator $\langle c, e \rangle$ can be expressed in normal form as

$$\left\langle c, \bigwedge_{a \in A} ((EPC_a(e) \triangleright a) \land (EPC_{\neg a}(e) \triangleright \neg a)) \right\rangle.$$
Regressing state variables

The formula $EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$ expresses the value of state variable $a \in A$ after applying $o$ in terms of values of state variables before applying $o$.

Either:

- $a$ became true, or
- $a$ was true before and it did not become false.
Regressing state variables: examples

Example

Let \( e = (b \triangleright a) \land (c \triangleright \neg a) \land b \land \neg d \).

<table>
<thead>
<tr>
<th>variable</th>
<th>( \text{EPC}<em>{\ldots}(e) \lor (\cdots \land \neg \text{EPC}</em>{\ldots}(e)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( b \lor (a \land \neg c) )</td>
</tr>
<tr>
<td>( b )</td>
<td>( \top \lor (b \land \neg \bot) \equiv \top )</td>
</tr>
<tr>
<td>( c )</td>
<td>( \bot \lor (c \land \neg \bot) \equiv c )</td>
</tr>
<tr>
<td>( d )</td>
<td>( \bot \lor (d \land \neg \top) \equiv \bot )</td>
</tr>
</tbody>
</table>
Regressing state variables: correctness

Lemma (B)

Let $a$ be a state variable, $o = \langle c, e \rangle$ an operator, $s$ a state, and $s' = \text{app}_o(s)$. Then $s \models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$ if and only if $s' \models a$.

Proof.

$(\Rightarrow)$: Assume $s \models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$. Do a case analysis on the two disjuncts.

1. Assume that $s \models EPC_a(e)$. By Lemma A, we have $a \in [e]_s$ and hence $s' \models a$.
2. Assume that $s \models a \land \neg EPC_{\neg a}(e)$. By Lemma, we have $A \neg a \not\in [e]_s$. Hence $a$ remains true in $s'$.
Regressing state variables: correctness

**Lemma (B)**

Let $a$ be a state variable, $o = \langle c, e \rangle$ an operator, $s$ a state, and $s' = \text{app}_o(s)$.

Then $s \models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$ if and only if $s' \models a$.

**Proof.**

$(\Rightarrow)$: Assume $s \models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$.

Do a case analysis on the two disjuncts.

1. Assume that $s \models EPC_a(e)$. By Lemma A, we have $a \in [e]_s$ and hence $s' \models a$.

2. Assume that $s \models a \land \neg EPC_{\neg a}(e)$. By Lemma, we have $A \neg a \not\in [e]_s$. Hence $a$ remains true in $s'$. 
Regressing state variables: correctness

Lemma (B)

Let $a$ be a state variable, $o = \langle c, e \rangle$ an operator, $s$ a state, and $s' = \text{app}_o(s)$.

Then $s \models EPC_a(e) \vee (a \land \neg EPC_{\neg a}(e))$ if and only if $s' \models a$.

Proof.

$(\Rightarrow)$: Assume $s \models EPC_a(e) \vee (a \land \neg EPC_{\neg a}(e))$.

Do a case analysis on the two disjuncts.

1. Assume that $s \models EPC_a(e)$. By Lemma A, we have $a \in [e]_s$ and hence $s' \models a$.

2. Assume that $s \models a \land \neg EPC_{\neg a}(e)$. By Lemma, we have $\neg a \not\in [e]_s$. Hence $a$ remains true in $s'$. 
Lemma (B)

Let \( a \) be a state variable, \( o = \langle c, e \rangle \) an operator, \( s \) a state, and \( s' = \text{app}_o(s) \).
Then \( s \models \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e)) \) if and only if \( s' \models a \).

Proof.

\((\Rightarrow)\): Assume \( s \models \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e)) \).
Do a case analysis on the two disjuncts.

1. Assume that \( s \models \text{EPC}_a(e) \). By Lemma A, we have \( a \in [e]_s \) and hence \( s' \models a \).
2. Assume that \( s \models a \land \neg \text{EPC}_{\neg a}(e) \). By Lemma, we have \( \neg a \notin [e]_s \). Hence \( a \) remains true in \( s' \).
Regressing state variables: correctness

Proof (ctd.)

(⇐): We showed that if the formula is true in \( s \), then \( a \) is true in \( s' \). For the second part, we show that if the formula is false in \( s \), then \( a \) is false in \( s' \).

- So assume \( s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e)) \).
- Then \( s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e)) \) (de Morgan).
- Analyze the two cases: \( a \) is true or it is false in \( s \).
  1. Assume that \( s \models a \). Now \( s \models EPC_{\neg a}(e) \) because \( s \models \neg a \lor EPC_{\neg a}(e) \).
     Hence by Lemma A \( \neg a \in [e]_s \) and we get \( s' \not\models a \).
  2. Assume that \( s \not\models a \). Because \( s \models \neg EPC_a(e) \), by Lemma A we get \( a \notin [e]_s \) and hence \( s' \not\models a \).

Therefore in both cases \( s' \not\models a \).
Regressing state variables: correctness

Proof (ctd.)

(\Rightarrow): We showed that if the formula is true in \( s \), then \( a \) is true in \( s' \). For the second part, we show that if the formula is false in \( s \), then \( a \) is false in \( s' \).

- So assume \( s \notmodels \ EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e)) \).
- Then \( s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e)) \) (de Morgan).
- Analyze the two cases: \( a \) is true or it is false in \( s \).
  1. Assume that \( s \models a \). Now \( s \models EPC_{\neg a}(e) \) because \( s \models \neg a \lor EPC_{\neg a}(e) \).
     Hence by Lemma A \( \neg a \in [e]_s \) and we get \( s' \notmodels a \).
  2. Assume that \( s \notmodels a \). Because \( s \models \neg EPC_a(e) \), by Lemma A we get \( a \notin [e]_s \) and hence \( s' \notmodels a \).

Therefore in both cases \( s' \notmodels a \).
Regressing state variables: correctness

Proof (ctd.)

\(\iff\): We showed that if the formula is \text{true} in \(s\), then \(a\) is \text{true} in \(s'\). For the second part, we show that if the formula is \text{false} in \(s\), then \(a\) is \text{false} in \(s'\).

- So assume \(s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))\).
- Then \(s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e))\) (de Morgan).

Analyze the two cases: \(a\) is true or it is false in \(s\).

1. Assume that \(s \models a\). Now \(s \models EPC_{\neg a}(e)\) because \(s \models \neg a \lor EPC_{\neg a}(e)\).
   Hence by Lemma A \(\neg a \in [e]_s\) and we get \(s' \not\models a\).
2. Assume that \(s \not\models a\). Because \(s \models \neg EPC_a(e)\), by Lemma A we get \(a \notin [e]_s\) and hence \(s' \not\models a\).

Therefore in both cases \(s' \not\models a\).
Regressing state variables: correctness

Proof (ctd.)

(⇐): We showed that if the formula is true in \( s \), then \( a \) is true in \( s' \). For the second part, we show that if the formula is false in \( s \), then \( a \) is false in \( s' \).

- So assume \( s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e)) \).
- Then \( s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e)) \) (de Morgan).
- Analyze the two cases: \( a \) is true or it is false in \( s \).

1. Assume that \( s \models a \). Now \( s \models EPC_{\neg a}(e) \) because \( s \models \neg a \lor EPC_{\neg a}(e) \).
   Hence by Lemma A \( \neg a \in [e]_s \) and we get \( s' \not\models a \).

2. Assume that \( s \not\models a \). Because \( s \models \neg EPC_a(e) \), by Lemma A we get \( a \notin [e]_s \) and hence \( s' \not\models a \).

Therefore in both cases \( s' \not\models a \).
Regressing state variables: correctness

Proof (ctd.)

$\iff$: We showed that if the formula is true in $s$, then $a$ is true in $s'$. For the second part, we show that if the formula is false in $s$, then $a$ is false in $s'$.

- So assume $s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$.
- Then $s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e))$ (de Morgan).
- Analyze the two cases: $a$ is true or it is false in $s$.

1. Assume that $s \models a$. Now $s \models EPC_{\neg a}(e)$ because $s \models \neg a \lor EPC_{\neg a}(e)$. Hence by Lemma A $\neg a \in [e]_s$ and we get $s' \not\models a$.

2. Assume that $s \not\models a$. Because $s \models \neg EPC_a(e)$, by Lemma A we get $a \not\in [e]_s$ and hence $s' \not\models a$.

Therefore in both cases $s' \not\models a$. 

Regression state variables: correctness

Proof (ctd.)

\(\iff\): We showed that if the formula is true in \(s\), then \(a\) is true in \(s'\). For the second part, we show that if the formula is false in \(s\), then \(a\) is false in \(s'\).

- So assume \(s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))\).
- Then \(s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e))\) (de Morgan).
- Analyze the two cases: \(a\) is true or it is false in \(s\).

1. Assume that \(s \models a\). Now \(s \models EPC_{\neg a}(e)\) because \(s \models \neg a \lor EPC_{\neg a}(e)\).
   Hence by Lemma A \(\neg a \in [e]_s\) and we get \(s' \not\models a\).

2. Assume that \(s \not\models a\). Because \(s \models \neg EPC_a(e)\), by Lemma A we get \(a \notin [e]_s\) and hence \(s' \not\models a\).

Therefore in both cases \(s' \not\models a\).
Regressing state variables: correctness

Proof (ctd.)

$(\Leftarrow)$: We showed that if the formula is true in $s$, then $a$ is true in $s'$. For the second part, we show that if the formula is false in $s$, then $a$ is false in $s'$.

- So assume $s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$.
- Then $s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e))$ (de Morgan).
- Analyze the two cases: $a$ is true or it is false in $s$.

1. Assume that $s \models a$. Now $s \models EPC_{\neg a}(e)$ because $s \models \neg a \lor EPC_{\neg a}(e)$.
   Hence by Lemma A $\neg a \in [e]_s$ and we get $s' \not\models a$.
2. Assume that $s \not\models a$. Because $s \models \neg EPC_a(e)$, by Lemma A we get $a \notin [e]_s$ and hence $s' \not\models a$.

Therefore in both cases $s' \not\models a$. 

Regression: general definition

We base the definition of regression on formulae \( EPC_l(e) \).

**Definition (general regression)**

Let \( \phi \) be a propositional formula and \( o = \langle c, e \rangle \) an operator. The *regression of \( \phi \) with respect to \( o \)* is

\[
\text{regr}_o(\phi) = c \land \phi_r \land f
\]

where

1. \( \phi_r \) is obtained from \( \phi \) by replacing each \( a \in A \) by \( EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e)) \), and
2. \( f = \bigwedge_{a \in A} \neg(EPC_a(e) \land EPC_{\neg a}(e)) \).

The formula \( f \) says that no state variable may become simultaneously true and false.
Regression examples

- \( \text{regr}_{\langle a, b \rangle}(b) \equiv a \land (\top \lor (b \land \neg \bot)) \land \top \equiv a \)

- \( \text{regr}_{\langle a, b \rangle}(b \land c \land d) \)
  \[ \equiv a \land \left( \top \lor (b \land \neg \bot) \right) \land \left( \bot \lor (c \land \neg \bot) \right) \land \left( \bot \lor (d \land \neg \bot) \right) \land \top \]
  \[ \equiv a \land c \land d \]

- \( \text{regr}_{\langle a, c \triangleright b \rangle}(b) \equiv a \land (c \lor (b \land \neg \bot)) \land \top \equiv a \land (c \lor b) \)

- \( \text{regr}_{\langle a, (c \triangleright b) \land (b \triangleright \neg b) \rangle}(b) \equiv a \land (c \lor (b \land \neg b)) \land \neg (c \land b) \)
  \[ \equiv a \land c \land \neg b \]

- \( \text{regr}_{\langle a, (c \triangleright b) \land (d \triangleright \neg b) \rangle}(b) \equiv a \land (c \lor (b \land \neg d)) \land \neg (c \land d) \)
  \[ \equiv a \land (c \lor b) \land (c \lor \neg d) \land (\neg c \lor \neg d) \]
Regression example: blocks world

Consider blocks world operators to move blocks $A$ and $B$ onto the table from the other block if they are clear:

\[
o_1 = \langle \top, (A-on-B \land A-clear) \triangleright (A-on-T \land B-clear \land \neg A-on-B) \rangle
\]
\[
o_2 = \langle \top, (B-on-A \land B-clear) \triangleright (B-on-T \land A-clear \land \neg B-on-A) \rangle
\]

Proof by regression that $o_2, o_1$ puts both blocks onto the table from any blocks world state:

\[
G = A-on-T \land B-on-T
\]
\[
\phi_1 = \text{regr}_{o_1}(G) \equiv ((A-on-B \land A-clear) \lor A-on-T) \land B-on-T
\]
\[
\phi_2 = \text{regr}_{o_2}(\phi_1)
\]
\[
\equiv ((A-on-B \land ((B-on-A \land B-clear) \lor A-clear)) \lor A-on-T)
\]
\[
\land ((B-on-A \land B-clear) \lor B-on-T)
\]

All three legal 2-block states satisfy $\phi_2$.
Similar plans exist for any number of blocks.
Regression example: binary counter

\[\neg b_0 \triangleright b_0 \land \\
((\neg b_1 \land b_0) \triangleright (b_1 \land \neg b_0)) \land \\
((\neg b_2 \land b_1 \land b_0) \triangleright (b_2 \land \neg b_1 \land \neg b_0))\]

\[EPC_{b_2}(e) = \neg b_2 \land b_1 \land b_0\]
\[EPC_{b_1}(e) = \neg b_1 \land b_0\]
\[EPC_{b_0}(e) = \neg b_0\]
\[EPC_{\neg b_2}(e) = \perp\]
\[EPC_{\neg b_1}(e) = \neg b_2 \land b_1 \land b_0\]
\[EPC_{\neg b_0}(e) = (\neg b_1 \land b_0) \lor (\neg b_2 \land b_1 \land b_0) \equiv (\neg b_1 \lor \neg b_2) \land b_0\]

Regression replaces state variables as follows:

- \(b_2\) by \((\neg b_2 \land b_1 \land b_0) \lor (b_2 \land \neg \perp) \equiv (b_1 \land b_0) \lor b_2\)
- \(b_1\) by \((\neg b_1 \land b_0) \lor (b_1 \land \neg (\neg b_2 \land b_1 \land b_0))\)
\[\equiv (\neg b_1 \land b_0) \lor (b_1 \land (b_2 \lor \neg b_0))\]
- \(b_0\) by \(\neg b_0 \lor (b_0 \land \neg ((\neg b_1 \lor \neg b_2) \land b_0)) \equiv \neg b_0 \lor (b_1 \land b_2)\)
General regression: correctness

**Theorem (correctness of \(\text{regr}_o(\phi)\))**

Let \(\phi\) be a formula, \(o\) an operator, \(s\) any state and \(s' = \text{app}_o(s)\). Then \(s \models \text{regr}_o(\phi)\) if and only if \(s' \models \phi\).

**Proof.**

Let \(e\) be the effect of \(o\). We show by structural induction over subformulae \(\phi'\) of \(\phi\) that \(s \models \phi'_r\) iff \(s' \models \phi'\), where \(\phi'_r\) is \(\phi'\) with every \(a \in A\) replaced by \(EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))\).

The rest of \(\text{regr}_o(\phi)\) just states that \(o\) is applicable in \(s\).

**Induction hypothesis** \(s \models \phi'_r\) if and only if \(s' \models \phi'\).

**Base cases 1 & 2** \(\phi' = \top\) or \(\phi' = \bot\): trivial, as \(\phi'_r = \phi'\).

**Base case 3** \(\phi' = a\) for some \(a \in A\):

Then \(\phi'_r = EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))\).

By Lemma B, \(s \models \phi'_r\) iff \(s' \models \phi'\).
Theorem (correctness of $\text{regr}_o(\phi)$)

Let $\phi$ be a formula, $o$ an operator, $s$ any state and $s' = \text{app}_o(s)$. Then $s \models \text{regr}_o(\phi)$ if and only if $s' \models \phi$.

Proof.

Let $e$ be the effect of $o$. We show by structural induction over subformulae $\phi'$ of $\phi$ that $s \models \phi'_r$ iff $s' \models \phi'$, where $\phi'_r$ is $\phi'$ with every $a \in A$ replaced by $\text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e))$.

The rest of $\text{regr}_o(\phi)$ just states that $o$ is applicable in $s$.

Induction hypothesis $s \models \phi'_r$ if and only if $s' \models \phi'$.

Base cases 1 & 2 $\phi' = \top$ or $\phi' = \bot$: trivial, as $\phi'_r = \phi'$.

Base case 3 $\phi' = a$ for some $a \in A$:

Then $\phi'_r = \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e))$.

By Lemma B, $s \models \phi'_r$ iff $s' \models \phi'$.
General regression: correctness

Theorem (correctness of $\text{regr}_o(\phi)$)

Let $\phi$ be a formula, $o$ an operator, $s$ any state and $s' = \text{app}_o(s)$. Then $s \models \text{regr}_o(\phi)$ if and only if $s' \models \phi$.

Proof.

Let $e$ be the effect of $o$. We show by structural induction over subformulae $\phi'$ of $\phi$ that $s \models \phi'_r$ iff $s' \models \phi'$, where $\phi'_r$ is $\phi'$ with every $a \in A$ replaced by $\text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e))$.

The rest of $\text{regr}_o(\phi)$ just states that $o$ is applicable in $s$.

Induction hypothesis $s \models \phi'_r$ if and only if $s' \models \phi'$.

Base cases 1 & 2 $\phi' = \top$ or $\phi' = \bot$: trivial, as $\phi'_r = \phi'$.

Base case 3 $\phi' = a$ for some $a \in A$:

Then $\phi'_r = \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e))$.

By Lemma B, $s \models \phi'_r$ iff $s' \models \phi'$. 
General regression: correctness

**Theorem (correctness of \( \text{regr}_o(\phi) \))**

Let \( \phi \) be a formula, \( o \) an operator, \( s \) any state and \( s' = \text{app}_o(s) \). Then \( s \models \text{regr}_o(\phi) \) if and only if \( s' \models \phi \).

**Proof.**

Let \( e \) be the effect of \( o \). We show by structural induction over subformulæ \( \phi' \) of \( \phi \) that \( s \models \phi'_r \) iff \( s' \models \phi' \), where \( \phi'_r \) is \( \phi' \) with every \( a \in A \) replaced by \( \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e)) \).

The rest of \( \text{regr}_o(\phi) \) just states that \( o \) is applicable in \( s \).

**Induction hypothesis** \( s \models \phi'_r \) if and only if \( s' \models \phi' \).

**Base cases 1 & 2** \( \phi' = \top \) or \( \phi' = \bot \): trivial, as \( \phi'_r = \phi' \).

**Base case 3** \( \phi' = a \) for some \( a \in A \):

Then \( \phi'_r = \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e)) \).

By Lemma B, \( s \models \phi'_r \) iff \( s' \models \phi' \).
General regression: correctness

**Theorem (correctness of** $\text{regr}_o(\phi)$$)$

Let $\phi$ be a formula, $o$ an operator, $s$ any state and $s' = \text{app}_o(s)$. Then $s \models \text{regr}_o(\phi)$ if and only if $s' \models \phi$.

**Proof.**

Let $e$ be the effect of $o$. We show by structural induction over subformulae $\phi'$ of $\phi$ that $s \models \phi'_r$ iff $s' \models \phi'$, where $\phi'_r$ is $\phi'$ with every $a \in A$ replaced by $\text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e))$.

The rest of $\text{regr}_o(\phi)$ just states that $o$ is applicable in $s$.

**Induction hypothesis** $s \models \phi'_r$ if and only if $s' \models \phi'$.

**Base cases 1 & 2** $\phi' = \top$ or $\phi' = \bot$: trivial, as $\phi'_r = \phi'$.

**Base case 3** $\phi' = a$ for some $a \in A$:

Then $\phi'_r = \text{EPC}_a(e) \lor (a \land \neg \text{EPC}_{\neg a}(e))$.

By Lemma B, $s \models \phi'_r$ iff $s' \models \phi'$. 
General regression: correctness

Proof (ctd.)

Inductive case 1 \( \phi' = \neg \psi \): By the induction hypothesis \( s \models \psi_r \)
iff \( s' \models \psi \). Hence \( s \models \phi'_r \) iff \( s' \models \phi' \) by the logical semantics of \( \neg \).

Inductive case 2 \( \phi' = \psi \lor \psi' \): By the induction hypothesis
\( s \models \psi_r \) iff \( s' \models \psi \), and \( s \models \psi'_r \) iff \( s' \models \psi' \).

Hence \( s \models \phi'_r \) iff \( s' \models \phi' \) by the logical semantics of \( \lor \).

Inductive case 3 \( \phi' = \psi \land \psi' \): By the induction hypothesis
\( s \models \psi_r \) iff \( s' \models \psi \), and \( s \models \psi'_r \) iff \( s' \models \psi' \).

Hence \( s \models \phi'_r \) iff \( s' \models \phi' \) by the logical semantics of \( \land \).
General regression: correctness

Proof (ctd.)

Inductive case 1  \( \phi' = \neg \psi \): By the induction hypothesis \( s \models \psi_r \)
iff \( s' \models \psi \). Hence \( s \models \phi'_r \) iff \( s' \models \phi' \) by the
logical semantics of \( \neg \).

Inductive case 2  \( \phi' = \psi \lor \psi' \): By the induction hypothesis
\( s \models \psi_r \) iff \( s' \models \psi \), and \( s \models \psi'_r \) iff \( s' \models \psi' \). Hence \( s \models \phi'_r \) iff \( s' \models \phi' \) by the logical
semantics of \( \lor \).

Inductive case 3  \( \phi' = \psi \land \psi' \): By the induction hypothesis
\( s \models \psi_r \) iff \( s' \models \psi \), and \( s \models \psi'_r \) iff \( s' \models \psi' \). Hence \( s \models \phi'_r \) iff \( s' \models \phi' \) by the logical
semantics of \( \land \).
General regression: correctness

Proof (ctd.)

**Inductive case 1** $\phi' = \neg \psi$: By the induction hypothesis $s \models \psi_r$ iff $s' \models \psi$. Hence $s \models \phi'_r$ iff $s' \models \phi'$ by the logical semantics of $\neg$.

**Inductive case 2** $\phi' = \psi \lor \psi'$: By the induction hypothesis $s \models \psi_r$ iff $s' \models \psi$, and $s \models \psi'_r$ iff $s' \models \psi'$. Hence $s \models \phi'_r$ iff $s' \models \phi'$ by the logical semantics of $\lor$.

**Inductive case 3** $\phi' = \psi \land \psi'$: By the induction hypothesis $s \models \psi_r$ iff $s' \models \psi$, and $s \models \psi'_r$ iff $s' \models \psi'$. Hence $s \models \phi'_r$ iff $s' \models \phi'$ by the logical semantics of $\land$. 
Emptiness and subsumption testing

The following two tests are useful when performing regression searches, to avoid exploring unpromising branches:

- Testing that a formula $\text{regr}_o(\phi)$ does not represent the empty set (= search is in a dead end).
  For example, $\text{regr}_{\langle a, \neg p \rangle}(p) \equiv a \land \bot \equiv \bot$.

- Testing that a regression step does not make the set of states smaller (= more difficult to reach).
  For example, $\text{regr}_{\langle b, c \rangle}(a) \equiv a \land b$.

Both of these problems are NP-hard.
The formula $\text{regr}_{o_1}(\text{regr}_{o_2}(\ldots \text{regr}_{o_{n-1}}(\text{regr}_{o_n}(\phi)))\ldots)$ may have size $O(|\phi||o_1||o_2|\ldots|o_{n-1}||o_n|)$, i.e., the product of the sizes of $\phi$ and the operators.

$\Rightarrow$ worst-case exponential size $O(m^n)$

**Logical simplifications**

- $\bot \land \phi \equiv \bot$, $\top \land \phi \equiv \phi$, $\bot \lor \phi \equiv \phi$, $\top \lor \phi \equiv \top$
- $a \lor \phi \equiv a \lor \phi[\bot/a]$, $\neg a \lor \phi \equiv \neg a \lor \phi[\top/a]$
- $a \land \phi \equiv a \land \phi[\top/a]$, $\neg a \land \phi \equiv \neg a \land \phi[\bot/a]$
- idempotency, absorption, commutativity, associativity, ...
Restricting formula growth in search trees

Problem  very big formulae obtained by regression

Cause  disjunctivity in the formulae: formulae without disjunctions easily convertible to small formulae \( l_1 \land \cdots \land l_n \) where \( l_i \) are literals and \( n \) is at most the number of state variables.

Idea  handle disjunctivity when generating search trees

Alternatives:

1. Do nothing. (May lead to very big formulae!)
2. Always eliminate all disjunctivity.
3. Reduce disjunctivity if formula becomes too big.
Reach goal $a \land b$ from state $I = \{a \leftrightarrow 0, b \leftrightarrow 0, c \leftrightarrow 0\}$. 

$G = a \land b$
Full splitting

- Planners for STRIPS operators only need to use formulae $l_1 \land \cdots \land l_n$ where $l_i$ are literals.
- Some general planners also restrict to this class of formulae. This is done as follows:
  1. Transform $\text{regr}_o(\phi)$ to disjunctive normal form (DNF): $(l_1^1 \land \cdots \land l_{n_1}^1) \lor \cdots \lor (l_m^m \land \cdots \land l_{n_m}^m)$.
  2. Generate one subtree of the search tree for each disjunct $l_1^i \land \cdots \land l_{n_i}^i$.
- The DNF formulae need not exist in its entirety explicitly: can generate one disjunct at a time.

→ branching is both on the choice of operator and on the choice of the disjunct of the DNF formula

→ increased branching factor and bigger search trees, but avoids big formulae
Full splitting: search tree example

Reach goal $a \land b$ from state $I = \{a \leftrightarrow 0, b \leftrightarrow 0, c \leftrightarrow 0\}$.

$(\neg c \lor a) \land b$ in DNF: $(\neg c \land b) \lor (a \land b)$

$\Rightarrow$ split into $\neg c \land b$ and $a \land b$
General splitting strategies

- With full splitting search tree can be exponentially bigger than without splitting. (But it is not necessary to construct the DNF formulae explicitly!)

- Without splitting the formulae may have size that is exponential in the number of state variables.

- A compromise is to split formulae only when necessary: combine benefits of the two extremes.

- There are several ways to split a formula $\phi$ to $\phi_1, \ldots, \phi_n$ such that $\phi \equiv \phi_1 \lor \cdots \lor \phi_n$. For example:
  
  - Transform $\phi$ to $\phi_1 \lor \cdots \lor \phi_n$ by equivalences like distributivity: $(\phi \lor \phi') \land \psi \equiv (\phi \land \psi) \lor (\phi' \land \psi)$.
  
  - Choose state variable $a$, set $\phi_1 = a \land \phi$ and $\phi_2 = \neg a \land \phi$, and simplify with equivalences like $a \land \psi \equiv a \land \psi[\top/a]$.