Principles of AI Planning

5. State-space search: progression and regression

Malte Helmert

Albert-Ludwigs-Universität Freiburg

October 31st, 2008

Principles of AI Planning

October 31st, 2008 — 5. State-space search: progression and regression

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

Planning by state-space 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? $\rightsquigarrow$ 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? $\rightsquigarrow$ 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

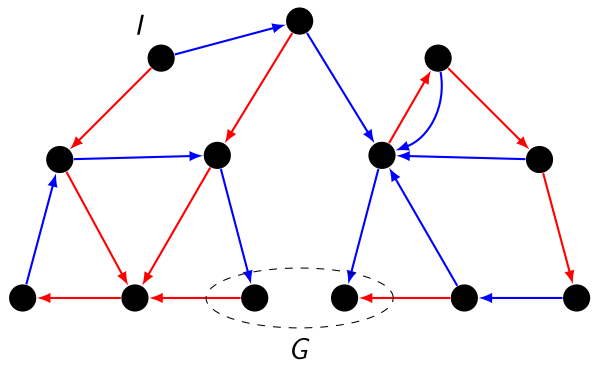
Search space representation in progression planners

Two alternative search spaces for progression planners:

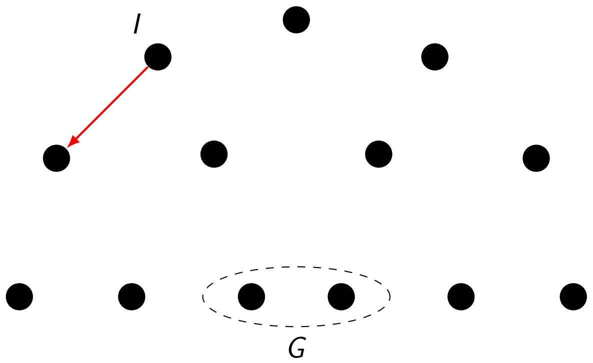
1. search nodes correspond to states
   - when the same state is generated along different paths, it is not considered again (duplicate detection)
   - pro: fast
   - con: memory intensive (must maintain closed list)
2. search nodes correspond to operator sequences
   - different operator sequences may lead to identical states (transpositions)
   - pro: can be very memory-efficient
   - con: much wasted work (often exponentially slower)

⇝ first alternative usually preferable

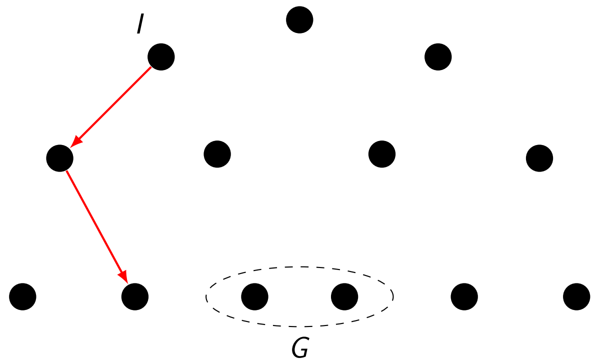
Progression planning example (depth-first search)



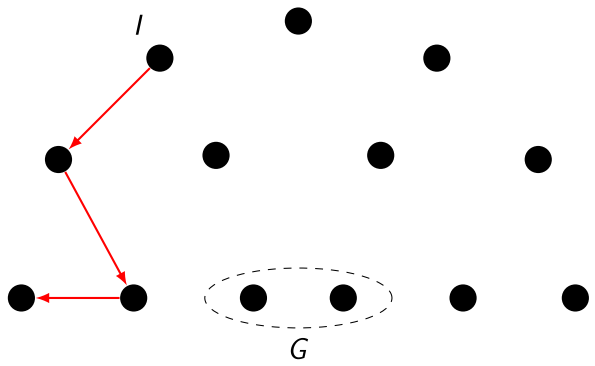
Progression planning example (depth-first search)



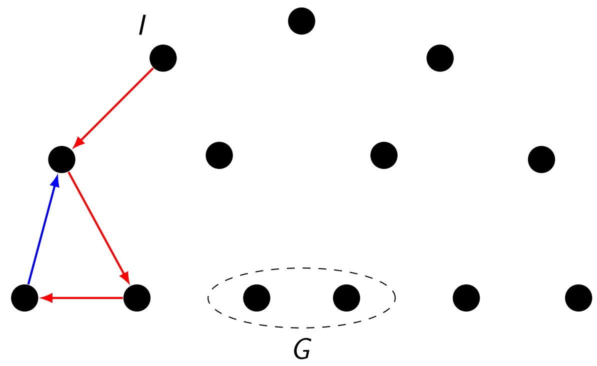
Progression planning example (depth-first search)



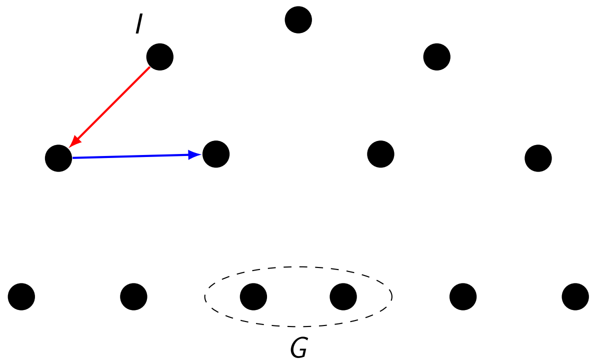
Progression planning example (depth-first search)



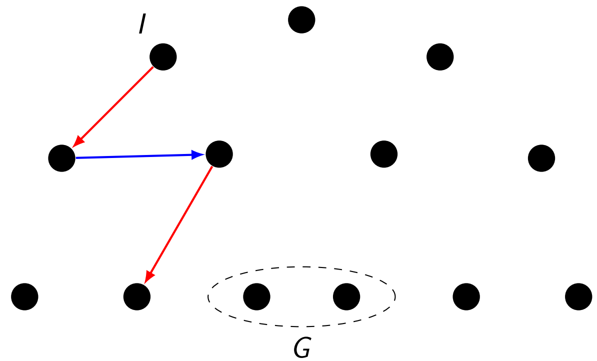
Progression planning example (depth-first search)



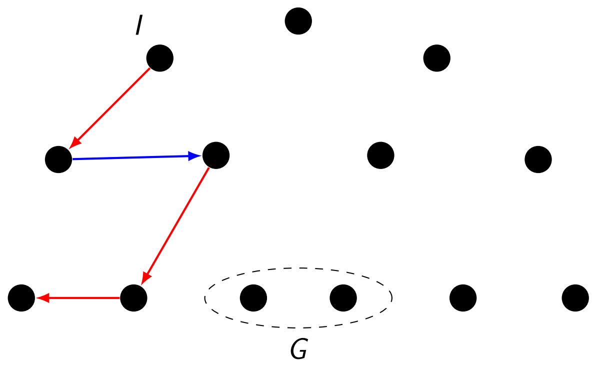
Progression planning example (depth-first search)



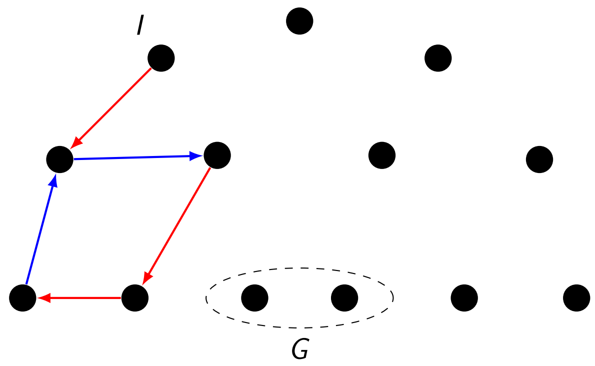
Progression planning example (depth-first search)



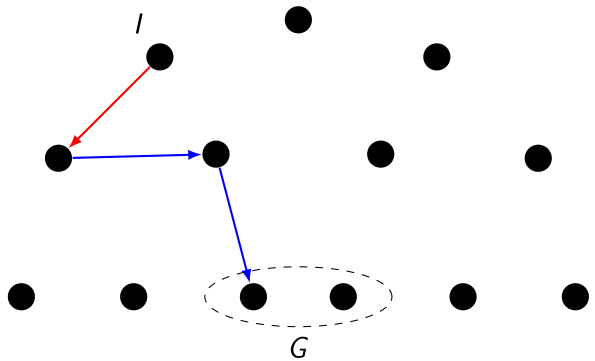
Progression planning example (depth-first search)



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 $regr_o(S)$ 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

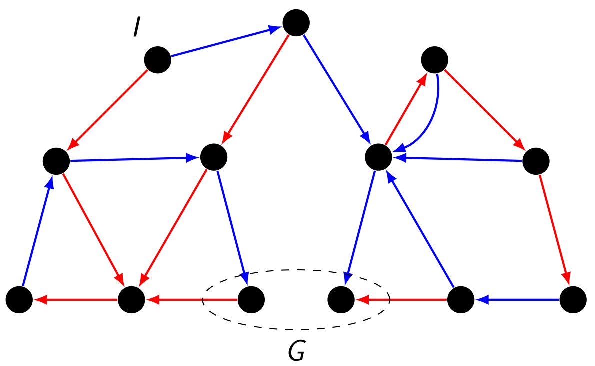
Pro: can handle many states simultaneously
Con: basic operations complicated and expensive

Search space representation in regression planners

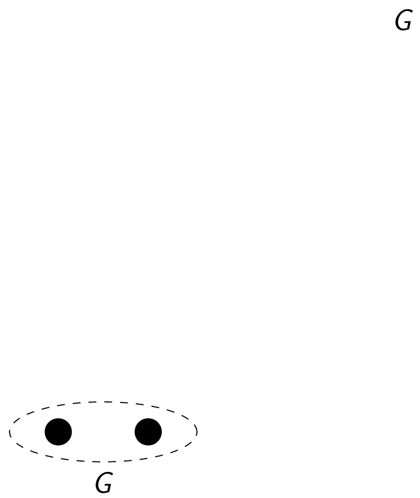
identify state sets with logical formulae:

- search nodes correspond to state sets
- each state set is represented by a logical formula: ϕ 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)



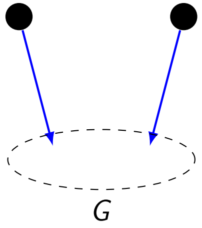
Regression planning example (depth-first search)



Regression planning example (depth-first search)

$\phi_1 = regr_{\longrightarrow}(G)$

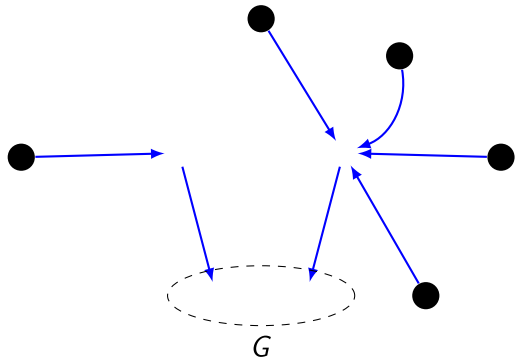
$\phi_1 \longrightarrow G$



Regression planning example (depth-first search)

$\phi_1 = regr_{\longrightarrow}(G)$
$\phi_2 = regr_{\longrightarrow}(\phi_1)$

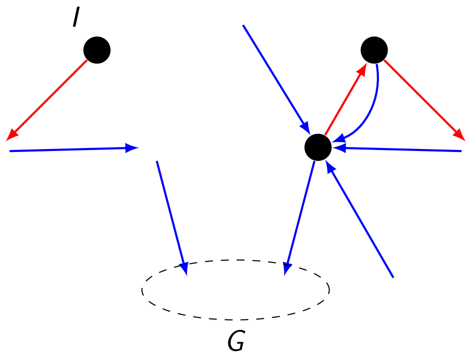
$\phi_2 \longrightarrow \phi_1 \longrightarrow G$



Regression planning example (depth-first search)

$\phi_1 = regr_{\longrightarrow}(G)$
$\phi_2 = regr_{\longrightarrow}(\phi_1)$
$\phi_3 = regr_{\longrightarrow}(\phi_2), I \models \phi_3$

$\phi_3 \longrightarrow \phi_2 \longrightarrow \phi_1 \longrightarrow G$



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 is very simple:

- Goals are conjunctions of literals $l_1 \wedge \cdots \wedge l_n$.
- 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.
- $\rightsquigarrow$ Outcome of regression is again conjunction of literals.

STRIPS regression

Definition

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

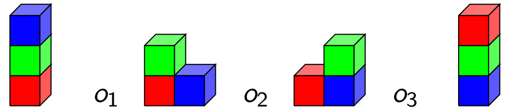
The STRIPS regression of ϕ with respect to $o = \langle \gamma, \eta \rangle$ is

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

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

$$o_1 = \langle {\color{blue}\blacksquare} on {\color{green}\blacksquare} \land {\color{blue}\blacksquare} clr, \neg {\color{blue}\blacksquare} on {\color{green}\blacksquare} \land {\color{blue}\blacksquare} onT \land {\color{green}\blacksquare} clr \rangle$$
$$o_2 = \langle {\color{green}\blacksquare} on {\color{red}\blacksquare} \land {\color{green}\blacksquare} clr \land {\color{blue}\blacksquare} clr, \neg {\color{blue}\blacksquare} clr \land \neg {\color{green}\blacksquare} on {\color{red}\blacksquare} \land {\color{green}\blacksquare} on {\color{blue}\blacksquare} \land {\color{red}\blacksquare} clr \rangle$$
$$o_3 = \langle {\color{red}\blacksquare} onT \land {\color{red}\blacksquare} clr \land {\color{green}\blacksquare} clr, \neg {\color{green}\blacksquare} clr \land \neg {\color{red}\blacksquare} onT \land {\color{red}\blacksquare} on {\color{green}\blacksquare} \rangle$$

$$G = {\color{red}\blacksquare} on {\color{green}\blacksquare} \land {\color{green}\blacksquare} on {\color{blue}\blacksquare}$$
$$\phi_1 = sregr_{o_3}(G) = {\color{green}\blacksquare} on {\color{blue}\blacksquare} \land {\color{red}\blacksquare} onT \land {\color{red}\blacksquare} clr \land {\color{green}\blacksquare} clr$$
$$\phi_2 = sregr_{o_2}(\phi_1) = {\color{red}\blacksquare} onT \land {\color{green}\blacksquare} clr \land {\color{green}\blacksquare} on {\color{red}\blacksquare} \land {\color{blue}\blacksquare} clr$$
$$\phi_3 = sregr_{o_1}(\phi_2) = {\color{red}\blacksquare} onT \land {\color{green}\blacksquare} on {\color{red}\blacksquare} \land {\color{blue}\blacksquare} clr \land {\color{blue}\blacksquare} on {\color{green}\blacksquare}$$

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 $\langle Q, D \rhd B \rangle$?
- 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:

$$\begin{aligned} EPC_l(l) &= \top \\ EPC_l(l') &= \bot \text{ if } l \neq l' \quad \text{(for literals } l'\text{)} \\ EPC_l(e_1 \land \dots \land e_n) &= EPC_l(e_1) \lor \dots \lor EPC_l(e_n) \\ EPC_l(c \rhd e) &= EPC_l(e) \land c \end{aligned}$$

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

Effect precondition examples

Example

$$\begin{aligned} EPC_a(b \wedge c) &= \bot \vee \bot \equiv \bot \\ EPC_a(a \wedge (b \rhd a)) &= \top \vee (\top \wedge b) \equiv \top \\ EPC_a((c \rhd a) \wedge (b \rhd a)) &= (\top \wedge c) \vee (\top \wedge b) \equiv c \vee b \end{aligned}$$

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.

Effect preconditions: connection to change sets

Proof (ctd.)

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

$$\begin{aligned} & l \in [e]_s \text{ iff } l \in [e_1]_s \cup \cdots \cup [e_n]_s && (\text{Def } [e_1 \wedge \cdots \wedge e_n]_s) \\ & \text{iff } l \in [e']_s \text{ for some } e' \in \{e_1, \ldots, e_n\} \\ & \text{iff } s \models EPC_l(e') \text{ for some } e' \in \{e_1, \ldots, e_n\} && (\text{IH}) \\ & \text{iff } s \models EPC_l(e_1) \vee \cdots \vee EPC_l(e_n) \\ & \text{iff } s \models EPC_l(e_1 \wedge \cdots \wedge e_n). && (\text{Def } EPC) \end{aligned}$$

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

$$\begin{aligned} & l \in [c \rhd e']_s \text{ iff } l \in [e']_s \text{ and } s \models c && (\text{Def } [c \rhd e']_s) \\ & \text{iff } s \models EPC_l(e') \text{ and } s \models c && (\text{IH}) \\ & \text{iff } s \models EPC_l(e') \wedge c \\ & \text{iff } s \models EPC_l(c \rhd e'). && (\text{Def } EPC) \end{aligned}$$

□

Effect preconditions: connection to normal form

Remark

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

$$\left\langle c, \bigwedge_{a \in A} ((EPC_a(e) \rhd a) \wedge (EPC_{\neg a}(e) \rhd \neg a)) \right\rangle.$$

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 $e = (b \rhd a) \land (c \rhd \neg a) \land b \land \neg d$.

variable	$EPC_{\dots}(e) \lor (\dots \land \neg EPC_{\neg \dots}(e))$
a	$b \lor (a \land \neg c)$
b	$\top \lor (b \land \neg \bot) \equiv \top$
c	$\bot \lor (c \land \neg \bot) \equiv c$
d	$\bot \lor (d \land \neg \top) \equiv \bot$

Regressing state variables: correctness

Lemma (B)

Let a be a state variable, $o = \langle c, e \rangle$ an operator, s a state, and $s' = app_o(s)$.
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 $\neg a \notin [e]_s$. Hence a remains true in s'.

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 $s' \not\models a$.

$\square$

Regression: general definition

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

Definition (general regression)

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

$$regr_o(\phi) = c \wedge \phi_r \wedge f$$

where

1. ϕ_r is obtained from ϕ by replacing each $a \in A$ by $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$, and
2. $f = \bigwedge_{a\in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$.

The formula f says that no state variable may become simultaneously true and false.

Regression examples

- $regr_{\langle a,b\rangle}(b) \equiv a \wedge (\top \vee (b \wedge \neg\bot)) \wedge \top \equiv a$
- $regr_{\langle a,b\rangle}(b \wedge c \wedge d)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot)) \wedge (\bot \vee (c \wedge \neg\bot)) \wedge (\bot \vee (d \wedge \neg\bot)) \wedge \top$
  $\equiv a \wedge c \wedge d$
- $regr_{\langle a,c\rhd b\rangle}(b) \equiv a \wedge (c \vee (b \wedge \neg\bot)) \wedge \top \equiv a \wedge (c \vee b)$
- $regr_{\langle a,(c\rhd b)\wedge(b\rhd\neg b)\rangle}(b) \equiv a \wedge (c \vee (b \wedge \neg b)) \wedge \neg(c \wedge b)$
  $\equiv a \wedge c \wedge \neg b$
- $regr_{\langle a,(c\rhd b)\wedge(d\rhd\neg b)\rangle}(b) \equiv a \wedge (c \vee (b \wedge \neg d)) \wedge \neg(c \wedge d)$
  $\equiv a \wedge (c \vee b) \wedge (c \vee \neg d) \wedge (\neg c \vee \neg d)$

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:

$$o_1 = \langle\top, (A\text{-}on\text{-}B \wedge A\text{-}clear) \rhd (A\text{-}on\text{-}T \wedge B\text{-}clear \wedge \neg A\text{-}on\text{-}B)\rangle$$
$$o_2 = \langle\top, (B\text{-}on\text{-}A \wedge B\text{-}clear) \rhd (B\text{-}on\text{-}T \wedge A\text{-}clear \wedge \neg B\text{-}on\text{-}A)\rangle$$

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

$$\begin{aligned}
G &= A\text{-}on\text{-}T \wedge B\text{-}on\text{-}T \\
\phi_1 &= regr_{o_1}(G) \equiv ((A\text{-}on\text{-}B \wedge A\text{-}clear) \vee A\text{-}on\text{-}T) \wedge B\text{-}on\text{-}T \\
\phi_2 &= regr_{o_2}(\phi_1) \\
&\equiv ((A\text{-}on\text{-}B \wedge ((B\text{-}on\text{-}A \wedge B\text{-}clear) \vee A\text{-}clear)) \vee A\text{-}on\text{-}T) \\
&\quad \wedge ((B\text{-}on\text{-}A \wedge B\text{-}clear) \vee B\text{-}on\text{-}T)
\end{aligned}$$

All three legal 2-block states satisfy ϕ_2.
Similar plans exist for any number of blocks.

Regression example: binary counter

$$\begin{gathered}
(\neg b_0 \rhd b_0) \wedge \\
((\neg b_1 \wedge b_0) \rhd (b_1 \wedge \neg b_0)) \wedge \\
((\neg b_2 \wedge b_1 \wedge b_0) \rhd (b_2 \wedge \neg b_1 \wedge \neg b_0))
\end{gathered}$$

$$\begin{aligned}
EPC_{b_2}(e) &= \neg b_2 \wedge b_1 \wedge b_0 \\
EPC_{b_1}(e) &= \neg b_1 \wedge b_0 \\
EPC_{b_0}(e) &= \neg b_0 \\
EPC_{\neg b_2}(e) &= \bot \\
EPC_{\neg b_1}(e) &= \neg b_2 \wedge b_1 \wedge b_0 \\
EPC_{\neg b_0}(e) &= (\neg b_1 \wedge b_0) \vee (\neg b_2 \wedge b_1 \wedge b_0) \equiv (\neg b_1 \vee \neg b_2) \wedge b_0
\end{aligned}$$

Regression replaces state variables as follows:

$$\begin{aligned}
b_2 &\text{ by } (\neg b_2 \wedge b_1 \wedge b_0) \vee (b_2 \wedge \neg\bot) \equiv (b_1 \wedge b_0) \vee b_2 \\
b_1 &\text{ by } (\neg b_1 \wedge b_0) \vee (b_1 \wedge \neg(\neg b_2 \wedge b_1 \wedge b_0)) \\
&\quad \equiv (\neg b_1 \wedge b_0) \vee (b_1 \wedge (b_2 \vee \neg b_0)) \\
b_0 &\text{ by } \neg b_0 \vee (b_0 \wedge \neg((\neg b_1 \vee \neg b_2) \wedge b_0)) \equiv \neg b_0 \vee (b_1 \wedge b_2)
\end{aligned}$$

General regression: correctness

Theorem (correctness of $regr_o(\phi)$)

Let ϕ be a formula, o an operator, s any state and $s' = app_o(s)$. Then $s \models regr_o(\phi)$ if and only if $s' \models \phi$.

Proof.

Let e be the effect of o. We show by structural induction over subformulae ϕ' of ϕ that $s \models \phi'_r$ iff $s' \models \phi'$, where ϕ'_r is ϕ' with every $a \in A$ replaced by $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$.
The rest of $regr_o(\phi)$ 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 $\phi'_r = EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$.
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 $\phi' = \psi \vee \psi'$: 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 $\vee$.

Inductive case 3 $\phi' = \psi \wedge \psi'$: 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 $\wedge$.

□

Emptiness and subsumption testing

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

- Testing that a formula $regr_o(\phi)$ does not represent the empty set (= search is in a dead end).
  For example, $regr_{\langle a, \neg p \rangle}(p) \equiv a \wedge \bot \equiv \bot$.
- Testing that a regression step does not make the set of states smaller (= more difficult to reach).
  For example, $regr_{\langle b, c \rangle}(a) \equiv a \wedge b$.

Both of these problems are NP-hard.

Formula growth

The formula $regr_{o_1}(regr_{o_2}(\ldots regr_{o_{n-1}}(regr_{o_n}(\phi))))$ may have size $O(|\phi||o_1||o_2|\ldots|o_{n-1}||o_n|)$, i. e., the product of the sizes of ϕ and the operators.
$\rightsquigarrow$ worst-case exponential size $O(m^n)$

Logical simplifications

- $\bot \wedge \phi \equiv \bot$, $\top \wedge \phi \equiv \phi$, $\bot \vee \phi \equiv \phi$, $\top \vee \phi \equiv \top$
- $a \vee \phi \equiv a \vee \phi[\bot/a]$, $\neg a \vee \phi \equiv \neg a \vee \phi[\top/a]$, $a \wedge \phi \equiv a \wedge \phi[\top/a]$, $\neg a \wedge \phi \equiv \neg a \wedge \phi[\bot/a]$
- 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 $l_1 \wedge \cdots \wedge l_n$ where l_i are literals and n is at most the number of state variables.

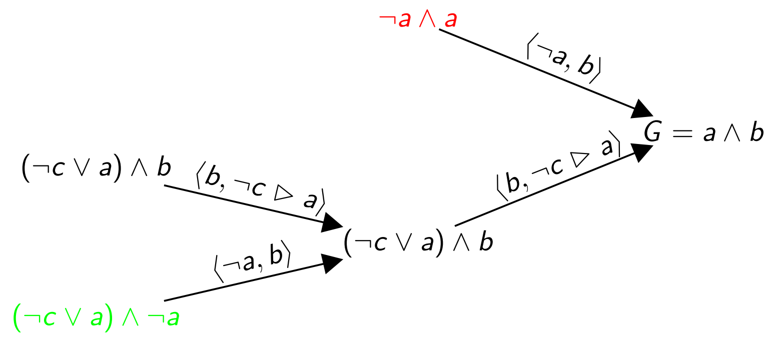
Idea handle disjunctivity when generating search trees

Alternatives:

1. Do nothing. (May lead to very big formulae!)
2. Always eliminate all disjunctivity.
3. Reduce disjunctivity if formula becomes too big.

Unrestricted regression: search tree example

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



Full splitting

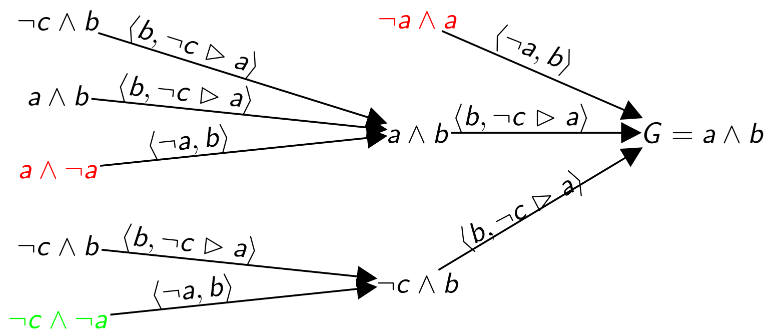
- Planners for STRIPS operators only need to use formulae $l_1 \wedge \cdots \wedge l_n$ where l_i are literals.
- Some general planners also restrict to this class of formulae. This is done as follows:
  1. Transform $regr_o(\phi)$ to disjunctive normal form (DNF): $(l_1^1 \wedge \cdots \wedge l_{n_1}^1)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

$\rightsquigarrow$ branching is both on the choice of operator and on the choice of the disjunct of the DNF formula

$\rightsquigarrow$ increased branching factor and bigger search trees, but avoids big formulae

Full splitting: search tree example

Reach goal $a \wedge b$ from state $I = \{a \mapsto 0, b \mapsto 0, c \mapsto 0\}$.
$(\neg c \vee a) \wedge b$ in DNF: $(\neg c \wedge b) \vee (a \wedge b)$
$\rightsquigarrow$ split into $\neg c \wedge b$ and $a \wedge b$



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 ϕ to $\phi_1, \ldots, \phi_n$ such that $\phi \equiv \phi_1 \vee \cdots \vee \phi_n$. For example:
  - Transform ϕ to $\phi_1 \vee \cdots \vee \phi_n$ by equivalences like distributivity: $(\phi \vee \phi') \wedge \psi \equiv (\phi \wedge \psi) \vee (\phi' \wedge \psi)$.
  - Choose state variable a, set $\phi_1 = a \wedge \phi$ and $\phi_2 = \neg a \wedge \phi$, and simplify with equivalences like $a \wedge \psi \equiv a \wedge \psi[\top/a]$.