ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-568

Regression for Classical and Nondeterministic Planning

Jussi Rintanen
NICTA & the Australian National University
Canberra, Australia

Abstract. Many forms of reasoning about actions and planning can be reduced to *regression*, the computation of the weakest precondition a state has to satisfy to guarantee the satisfaction of another condition in the successor state. In this work we formalize a general syntactic regression operation for ground PDDL operators, show its correctness, and define a composition operation based on regression. As applications we present a very simple yet powerful algorithm for computing invariants, as well as a generalization of the h^n heuristic of Haslum and Geffner to PDDL.

1 Introduction

Although it is well known that the expressivity of PDDL [13] is required for efficient modeling of many planning problems [14], most planner implementations still restrict to the STRIPS language in which action preconditions are conjunctions of (positive) literals and all effects are unconditional. Anecdotal evidence tells that this is due to the difficulty to reason about actions more general than STRIPS.

PDDL can often be efficiently reduced to STRIPS, but certain classes of operators that have disjunctive preconditions or several conditional effects with logically independent antecedents lead to an exponential number of STRIPS operators. Furthermore, reduction to STRIPS is impossible for many generalizations of classical planning: in the presence of partial observability, splitting one operator to several is in general incorrect because in execution time it may not be possible to choose which operator to execute. This provides a strong motivation for the generalization of STRIPS-based algorithms and other planning techniques to more general languages such as PDDL.

Our work defines a regression operation for ground PDDL operators and demonstrates its applications to planning. Pednault [15] defines regression for his ADL class of operators but his definition skips over the concrete syntax of what is today known as ADL/PDDL, The key component of the regression operation we define for ground PDDL is Definition 3 that maps a PDDL operator and a state variable to formulae describing the conditions under which the variable becomes true and false. The basis of regression operations is the substitution of a variable by an expression that describes its new value. This was used in the assignment axioms of the Hoare calculus [9] and later by Dijkstra for computing weakest preconditions [4].

The structure of the paper is as follows. Section 2 defines the classical planning problem for ground PDDL, the regression operation and the composition operation, and discusses their formal properties. Section 3 gives applications to invariants and heuristics. Section 4 defines regression for nondeterministic operators, Section 5 discusses related work, and Section 6 concludes the paper.

2 Definitions

Definition 1 *Let A be a set of state variables. An operator is a pair $\langle p, e\rangle$ where p is a propositional formula over A describing the precondition, and e is an effect, defined recursively as follows.*

1. *a and $\neg a$ for state variables $a \in A$ are effects.*
2. *$e_1 \wedge \cdots \wedge e_n$ is an effect if $e_1, \ldots, e_n$ are effects.*
3. *$c \rhd e$ is an effect if c is a formula and e is an effect.*

The meaning of conditional effects $c \rhd e$ is that effects e take place if the condition c is true.

Definition 2 (Execution) *Let $\langle p, e\rangle$ be an operator over A. Let $s : A \to \{0, 1\}$ be a state. The operator is* executable in *s if $s \models p$ and the set $[\![e]\!]_s$ is consistent. This set is recursively defined as follows.*

1. *$[\![a]\!]_s = \{a\}$ and $[\![\neg a]\!]_s = \{\neg a\}$ for $a \in A$.*
2. *$[\![e_1 \wedge \cdots \wedge e_n]\!]_s = \bigcup_{i=1}^{n} [\![e_i]\!]_s$.*
3. *$[\![c \rhd e]\!]_s = [\![e]\!]_s$ if $s \models c$ and $[\![c \rhd e]\!]_s = \emptyset$ otherwise.*

An operator $\langle p, e\rangle$ induces a partial function $R\langle p, e\rangle$ on states: states s and s' are related by $R\langle p, e\rangle$ if $s \models p$ and s' is obtained from s by making the literals in $[\![e]\!]_s$ true and retaining the truth-values of state variables not occurring in $[\![e]\!]_s$. Define $exc_o(s) = s'$ by $sR(o)s'$ and $exc_{o_1;\ldots;o_n}(s) = exc_{o_n}(\ldots exc_{o_1}(s)\ldots)$.

The main application of regression is in backward-search in which the basic step, computing a formula that represents the predecessor states (the new subgoal), is regression. The key component of regression for PDDL-style operators is given next.

Definition 3 *We recursively define the condition $E_l(e)$ of literal l made true by an operator with the effect e as follows.*

$$\begin{aligned} E_l(l) &= \top \\ E_l(l') &= \bot \text{ when } l \neq l' \text{ (for literals } l') \\ E_l(e_1 \wedge \cdots \wedge e_n) &= E_l(e_1) \vee \cdots \vee E_l(e_n) \\ E_l(c \rhd e) &= c \wedge E_l(e) \end{aligned}$$

The symbols $\top$ and $\bot$ denote *true* and *false*, respectively. The case $E_l(e_1 \wedge \cdots \wedge e_n) = E_l(e_1) \vee \cdots \vee E_l(e_n)$ is defined as a disjunction because it is sufficient that at least one effect makes l true.

Definition 4 *Let A be the set of state variables. We define the condition $E_l(o)$ of operator $o = \langle p, e\rangle$ being executable so that literal l is made true as $p \wedge E_l(e) \wedge \bigwedge_{a \in A} \neg(E_a(e) \wedge E_{\neg a}(e))$.*

The third conjunct in the formula requires that no state variable is made both true and false. The formula $E_l(e)$ indicates in which states the literal l is made true by e. It is closely related to $[\![e]\!]_s$.

Lemma 5 *Let A be the set of state variables, s a state on A, l a literal on A, and o an operator with effect e. Then*

1. $l \in [\![e]\!]_s$ *if and only if* $s \models E_l(e)$, *and*
2. $exc_o(s)$ *is defined and* $l \in [\![e]\!]_s$ *if and only if* $s \models E_l(o)$.

The formula $E_a(e) \vee (a \wedge \neg E_{\neg a}(e))$ expresses the truth of $a \in A$ after the execution of e in terms of truth-values of state variables before the execution: either a becomes true, or a is true before and does not become false.

Lemma 6 *Let $a \in A$ be a state variable, $o = \langle p, e\rangle \in O$ an operator, and s and $s' = exc_o(s)$ states. Then $s \models E_a(e) \vee (a \wedge \neg E_{\neg a}(e))$ if and only if $s' \models a$.*

Definition 7 (Regression) *Let ϕ be a propositional formula and $o = \langle p, e\rangle$ an operator. The* regression *of ϕ with respect to o is $rg_o(\phi) = \phi_r \wedge p \wedge \chi$ where $\chi = \bigwedge_{a\in A} \neg(E_a(e) \wedge E_{\neg a}(e))$ and ϕ_r is obtained from ϕ by replacing every $a \in A$ by $E_a(e) \vee (a \wedge \neg E_{\neg a}(e))$. Define $rg_e(\phi) = \phi_r \wedge \chi$ and $rg_{o_1;\ldots;o_n}(\phi) = rg_{o_1}(\cdots rg_{o_n}(\phi)\cdots)$.*

The formula χ corresponds to the requirement that $[\![e]\!]_s$ is consistent for an operator to be executable.

The reason why regression is useful is that it allows to compute the predecessor states by simple formula manipulation.

Next we formalize the important property of regression.

Theorem 8 *Let ϕ be a formula over A, o an operator over A, and S the set of all states i.e. valuations of A. Then $\{s \in S | s \models rg_o(\phi)\} = \{s \in S |\, exc_o(s) \models \phi\}$.*

Proof: We show that for any state s, $s \models \mathrm{rg}_o(\phi)$ if and only if $exc_o(s)$ is defined and $exc_o(s) \models \phi$. By definition $\mathrm{rg}_o(\phi) = \phi_r \wedge p \wedge \chi$ for $o = \langle p, e\rangle$ where ϕ_r is obtained from ϕ by replacing each $a \in A$ by $E_a(e) \vee (a \wedge \neg E_{\neg a}(e))$ and $\chi = \bigwedge_{a\in A} \neg(E_a(e) \wedge E_{\neg a}(e))$.

First we show that $s \models c \wedge \chi$ if and only if $exc_o(s)$ is defined.

$s \models c \wedge \chi$ iff $s \models c$ and $\{a, \neg a\} \not\subseteq [\![e]\!]_s$ for all $a \in A$
iff $exc_o(s)$ is defined

The two equivalences are respectively by Lemma 5 and Definition 2.

Then we show that $s \models \phi_r$ if and only if $exc_o(s) \models \phi$. This is by structural induction over subformulae ϕ' of ϕ and formulae ϕ'_r obtained from ϕ' by replacing $a \in A$ by $E_a(e) \vee (a \wedge \neg E_{\neg a}(e))$

Induction hypothesis: $s \models \phi'_r$ if and only if $exc_o(s) \models \phi'$.

Base case 1, $\phi' = \top$: Now $\phi'_r = \top$ and both are true in the respective states.

Base case 2, $\phi' = \bot$: Now $\phi'_r = \bot$ and both are false in the respective states.

Base case 3, $\phi' = a$ for some $a \in A$: Now $\phi'_r = E_a(e) \vee (a \wedge \neg E_{\neg a}(e))$. By Lemma 6 $s \models \phi'_r$ if and only if $exc_o(s) \models \phi'$.

Inductive case 1, $\phi' = \neg\theta$: By the induction hypothesis $s \models \theta_r$ iff $exc_o(s) \models \theta$. Hence $s \models \phi'_r$ iff $exc_o(s) \models \phi'$ by the truth-definition of $\neg$.

Inductive case 2, $\phi' = \theta \vee \theta'$: By the induction hypothesis $s \models \theta_r$ iff $exc_o(s) \models \theta$, and $s \models \theta'_r$ iff $exc_o(s) \models \theta'$. Hence $s \models \phi'_r$ iff $exc_o(s) \models \phi'$ by the truth-definition of $\vee$.

Inductive case 3 for $\phi' = \theta \wedge \theta'$ goes like the previous case. □

It may appear that for n consecutive regression steps the size of the formula grows exponentially, as each variable occurrence may be replaced by a bigger formula containing several variables. However, if the formula is represented in the circuit form instead of a tree-like formula, each variable occurs at most once. Hence a sequence of regression steps only leads to a worst-case polynomial increase in size. The circuits can often be simplified to keep them small, and in special cases, like STRIPS operators, there is a constant upper bound on the size of formulae/circuits.

In addition to being the basis of backward search, regression has many other applications in reasoning about sequences of actions. Central questions concern the relation between a given action and a given sequence of actions: whether they are executable in exactly the same states and whether they have the same effects. This is the basis of computing macro-actions [10] and the elimination of redundant actions [8]. Answering this question requires the composition of a sequence of two or more operators. The composition $o_1 \circ o_2$ of $o_1 = \langle p_1, e_1\rangle$ and $o_2 = \langle p_2, e_2\rangle$ is an operator that behaves like applying o_1 followed by o_2. For a to be true after o_2 we can regress a with respect to o_2, obtaining $E_a(e_2) \vee (a \wedge \neg E_{\neg a}(e_2))$. Condition for this formula to be true after o_1 is obtained by regressing with e_1, leading to

$$\begin{aligned}&\mathrm{rg}_{e_1}(E_a(e_2) \vee (a \wedge \neg E_{\neg a}(e_2)))\\ &= \mathrm{rg}_{e_1}(E_a(e_2)) \vee (\mathrm{rg}_{e_1}(a) \wedge \neg \mathrm{rg}_{e_1}(E_{\neg a}(e_2)))\\ &= \mathrm{rg}_{e_1}(E_a(e_2)) \vee ((E_a(e_1) \vee (a \wedge \neg E_{\neg a}(e_2))) \wedge \neg \mathrm{rg}_{e_1}(E_{\neg a}(e_2))).\end{aligned}$$

Since we want to define an effect $\phi \rhd a$ of $o_1 \circ o_2$ so that a becomes true whenever o_1 followed by o_2 would make it true, the formula ϕ does not have to represent the case in which a is true already before the execution of $o_1 \circ o_2$. Hence we can simplify the above formula to

$$\mathrm{rg}_{e_1}(E_a(e_2)) \vee (E_a(e_1) \wedge \neg \mathrm{rg}_{e_1}(E_{\neg a}(e_2))).$$

An analogous formula is needed for making $\neg a$ false. This leads to the following definition.

Definition 9 (Composition) *Let $o_1 = \langle p_1, e_1\rangle$ and $o_2 = \langle p_2, e_2\rangle$ be two operators on A. Then their* composition *$o_1 \circ o_2$ is defined as*

$$\left\langle p, \bigwedge_{a\in A} \begin{array}{l}((rg_{e_1}(E_a(e_2)) \vee (E_a(e_1) \wedge \neg rg_{e_1}(E_{\neg a}(e_2)))) \rhd a) \wedge\\ ((rg_{e_1}(E_{\neg a}(e_2)) \vee (E_{\neg a}(e_1) \wedge \neg rg_{e_1}(E_a(e_2)))) \rhd \neg a)\end{array} \right\rangle$$

where $p = rg_{o_1}(p_2) \wedge \bigwedge_{a\in A} \neg\,(E_a(e_1) \wedge E_{\neg a}(e_1))$.

Example 10 *Consider $o = \langle \top, (\neg b_0 \rhd b_0) \wedge (\neg b_1 \wedge b_0 \rhd (b_1 \wedge \neg b_0)) \wedge (\neg b_2 \wedge b_1 \wedge b_0 \rhd (b_2 \wedge \neg b_1 \wedge \neg b_0)\rangle$ which increments a 3-bit binary number by 1.*[1] *The composition of o with itself, representing increment by 2, is (after applying the De Morgan laws)*

$$\begin{array}{l}\langle \top, ((\neg b_2 \vee \neg b_1) \wedge b_0) \vee (\neg b_0 \wedge b_2 \wedge b_1) \rhd b_0) \wedge\\ \quad ((\neg b_0 \wedge b_1 \wedge \neg b_2) \vee\\ \quad (((b_2 \wedge b_1) \vee \neg b_0) \wedge ((\neg b_1 \vee (b_0 \wedge \neg b_2)) \wedge (\neg b_0 \vee b_1))) \rhd \neg b_0) \wedge\\ \quad ((((b_2 \wedge b_1) \vee \neg b_0) \wedge ((\neg b_1 \vee (b_0 \wedge \neg b_2)) \wedge (\neg b_0 \vee b_1)))\\ \quad \vee (b_0 \wedge \neg b_1) \rhd b_1) \wedge\\ \quad ((\neg b_0 \wedge b_1 \wedge \neg b_2) \vee (b_0 \wedge \neg b_2 \wedge b_1) \rhd \neg b_1) \wedge\\ \quad ((\neg b_0 \wedge b_1 \wedge \neg b_2) \vee (b_0 \wedge \neg b_2 \wedge b_1) \rhd b_2)\rangle.\end{array}$$

Further logical simplification and elimination of redundant conditional effects and simplifying unnecessary conditions yields

$$\begin{array}{l}\langle \top, (\neg b_0 \wedge b_2 \wedge b_1 \rhd b_0) \wedge\\ \quad (\neg b_1 \rhd b_1) \wedge\\ \quad (\neg b_2 \wedge b_1 \rhd \neg b_1) \wedge\\ \quad (\neg b_2 \wedge b_1 \rhd b_2)\rangle.\end{array}$$

■

Theorem 11 *Let o_1 and o_2 be operators and s a state. Then $exc_{o_1\circ o_2}(s)$ is defined if and only if $exc_{o_1;o_2}(s)$ is defined, and $exc_{o_1\circ o_2}(s) = exc_{o_1;o_2}(s)$.*

[1] Notice that 111 is not incremented further.

3 Applications

3.1 Invariants

Very interestingly, the regression operation can be used as the main component of a powerful and intuitive algorithm for computing *invariants*. An invariant property of a planning problem is satisfied by every state that is reachable from the initial state(s). An equivalent inductive definition states that a property is invariant if the initial states satisfy it and every action preserves it.

Main applications of invariants are planning by SAT and CSPs [11] in which invariants help to prune the search space, the validation of domain models in which invariants give information about dependencies between state variables, inexpensive incomplete tests for unreachability, and the computation of heuristics.

We generalize the inductive algorithm [16] to general operators. The novelty is the extremely simple structure of the algorithm given the generality of the operator definition. The algorithm invariants(A,I,O,n) in Figure 1 computes invariants with at most n literals for operators O and an initial state I over state variables A. The runtimes increase quickly as n is increased and in practice one can use $n = 2$ or $n = 3$. We define $\text{lits}(l_1 \vee \cdots \vee l_n) = \{l_1, \ldots, l_n\}$. The loop on line 5 is repeated until there are no $o \in O$ and clauses $c \in C$ such that $C \cup \{\text{rg}_o(\neg c)\}$ is satisfiable.

Lemma 12 *Let C be a set of clauses, ϕ a formula, and o an operator. If $C \cup \{rg_o(\neg\phi)\}$ is unsatisfiable, then $exc_o(s) \models \phi$ for all states s such that $s \models C$ and o is executable in s.*

Proof: Easy corollary of Theorem 8. □

```
1:  procedure invariants(A, I, O, n);
2:  C := {a ∈ A | I ⊨ a} ∪ {¬a | a ∈ A, I ⊭ a};
3:  repeat
4:      C' := C;
5:      for each o ∈ O and c ∈ C s.t. C' ∪ {rg_o(¬c)} ∈ SAT do
6:          C := C \ {c};
7:          if |lits(c)| < n then
8:              begin                    (* Add weaker clauses. *)
9:                  C := C ∪ {c ∨ a | a ∈ A} ∪ {c ∨ ¬a | a ∈ A};
10:             end
11:     end do
12: until C = C';
13: return C;
```

Figure 1. Algorithm for computing a set of invariant clauses

On lines 7 and 9, when a clause c is not guaranteed to hold, weaker clauses $c \vee l$ may be, so replace c by all clauses that are weaker by having one more literal. If these clauses don't hold either, they will be similarly removed and replaced by weaker ones.

Theorem 13 *Let A be a set of state variables, I a state, O a set of operators, and $n \geq 1$ an integer. Then the procedure invariants(A, I, O, n) returns a set C of clauses with at most n literals so that $exc_{o_1;\ldots;o_m}(I) \models C$ for any sequence $o_1; \ldots; o_m$ of operators from O.*

Proof: Let C_0 be the value assigned to the variable C on line 2 in the procedure and $C_1, C_2, \ldots$ be the values of the variable in the end of each iteration of the outermost *repeat* loop.

Induction hypothesis: for every $\{o_1, \ldots, o_i\} \subseteq O$ and $c \in C_i$, $exc_{o_1;\ldots;o_i}(I) \models c$.

Base case $i = 0$: $exc_\epsilon(I)$ for the empty sequence is by definition I itself, and by construction C_0 consists of only formulae that are true in the initial state.

Inductive case $i \geq 1$: Take any $\{o_1, \ldots, o_i\} \subseteq O$ and $c \in C_i$. Analyze two cases.

1. If $c \in C_{i-1}$, then by the induction hypothesis $exc_{o_1;\ldots;o_{i-1}}(I) \models c$. Since $c \in C_i$ it must be that $C_{i-1} \cup \{\text{rg}_{o_i}(\neg c)\}$ is unsatisfiable. Hence by Lemma 12 $exc_{o_1;\ldots;o_i}(I) \models c$.
2. If $c \notin C_{i-1}$, it must be because $C_{i-1} \cup \{\text{rg}_{o'}(\neg c')\}$ is satisfiable for some $o' \in O$ and $c' \in C_{i-1}$ such that c is obtained from c' by conjoining some literals to it and $c' \models c$. Since $c' \in C_{i-1}$, by the induction hypothesis $exc_{o_1;\ldots;o_{i-1}}(I) \models c'$. Since $c' \models c$ also $exc_{o_1;\ldots;o_{i-1}}(I) \models c$. Since $C_{i-1} \cup \{\text{rg}_{o_i}(\neg c)\}$ is unsatisfiable, $exc_{o_1;\ldots;o_i}(I) \models c$ by Lemma 12.

This finishes the induction proof. The iteration of the procedure stops when $C_i = C_{i-1}$, meaning that the claim of the theorem holds for arbitrarily long sequences $o_1; \ldots; o_m$. □

To make the algorithm run in polynomial time the satisfiability and logical consequence tests should be performed by algorithms that approximate these tests in polynomial time. If restricted to STRIPS operators, the inductive invariant computation [16] is obtained by implementing the satisfiability test $C \cup \{\text{rg}_o(\neg c)\}$ as an incomplete test by unit resolution. More generally, it may be useful to have a stronger tractable satisfiability test. The proof of Theorem 13 remains valid as long as the incomplete satisfiability test does not falsely indicate unsatisfiability for a satisfiable set.

Inference of facts that hold at given time points was first considered in the GraphPlan algorithm of Blum and Furst in the form of mutexes [1]. This planning graph construction, similarly to early algorithms for computing invariants [5, 16] restricts to STRIPS operators. Later works have considered more general classes of operators [6, 12] adopting the inductive definition definition of invariants first used in [1, 16]. Gerevini and Schubert [6] consider conditional effects but no disjunctions. Lin [12] tries to find invariants for a class of problems by looking at problem instances with a small state space and eliminating candidate invariants if they are falsified by the chosen problem instances.

3.2 Haslum and Geffner's h^n

Our invariant algorithm computes a generalization of Haslum & Geffner's h^n heuristic [7] which is defined for STRIPS only. An estimate for the distance of any formula ϕ (precondition or goal) is k if ϕ is satisfiable with C_k but not with C_{k-1} (an incomplete satisfiability test can be used without sacrificing the admissibility of the heuristic.) Haslum and Geffner's estimate $G^n(V)$ for the distance of a set V of variables from the initial state can be expressed in terms of our sets C_i when our parameter n equals m: for $V = \{a_1, \ldots, a_m\}$, $G^n(V) = k$ iff there is $\neg b_1 \vee \cdots \vee \neg b_j \in C_{k-1}$ such that $\{b_1, \ldots, b_j\} \subseteq V$ and there is no such clause in C_k, and $G^n(V) = 0$ if $\neg a \notin C_0$ for all $a \in V$.

Haslum and Geffner define states as subsets of the set A of all state variables. We will call this kinds of states *h-states* to distinguish them from our definition of states. Haslum and Geffner define $R(V)$ as the set of pairs (B, o) such that the operator o reaches a h-state V from a h-state B. This is essentially a simple regression operation for STRIPS. We ignore the operator o (because we don't need it for costs

unlike Haslum and Geffner who consider non-unitary costs) and define $R(V)$ simply as all the minimal sets of variables that have to be true for variables V to be true after executing one of the operators. Now $R(V)$ has the following property.

Lemma 14 *For all $B \in R(V)$, $\bigwedge_{a\in B} a \models rg_o(\bigwedge_{a\in V} a)$.*

The definition of the heuristic is as follows. For $V \subseteq A$ let

$$\begin{array}{ll} G^n(V) = 0 & \text{if } V \subseteq I \\ G^n(V) = \min_{B\in R(V)}(1 + G^n(B)) & \text{if } |V| \leq n \text{ and } V \not\subseteq I \\ G^n(V) = \max_{B\subset V, |B|=n} G^n(B) & \text{if } |V| > n. \end{array}$$

Theorem 15 *For a STRIPS problem, let C_i be the sets computed by the algorithm in Figure 1 as explained in the proof of Theorem 13. Let $V \subseteq A$ be a set of variables. If $G^n(V) = k$ for any $k \geq 1$, then $C_{k-1} \cup V$ is unsatisfiable and $C_k \cup V$ is satisfiable, and $G^n(V) = 0$ iff $C_0 \cup V$ is satisfiable.*

Proof: We give a proof sketch.

Induction hypothesis: for every $i \geq 0$, for any $V \subseteq A$,

1. if $G^n(V) = i$ then $C_i \cup V$ is satisfiable,
2. if $G^n(V) = i$ then $C_j \cup V$ is unsatisfiable for $j \in \{0, \ldots, i-1\}$.

Base case $i = 0$: Let $V \subseteq A$ be any set of variables.

1. If $G^n(V) = 0$ then $V \subseteq C_0$. Since C_0 is satisfiable, also $C_0 \cup V$ is satisfiable.
2. Holds trivially because $\{0, \ldots, i-1\} = \emptyset$.

Inductive case $i \geq 1$: Remark A. If $C_i \models \neg a_1 \vee \cdots \vee \neg a_k$, then $\neg b_1 \vee \cdots \vee \neg b_m \in C_i$ for some $\{b_1, \ldots, b_m\} \subseteq \{a_1, \ldots, a_k\}$.

1. Assume $G^n(V) = i$. Then there is an operator o that reaches the h-state V from a h-state B such that $G^n(B) = i - 1$. Since $G^n(B) = i - 1$, by the induction hypothesis $C_{i-1} \cup B$ is satisfiable. By Lemma 14 $\bigwedge_{a\in B} a \models \text{rg}_o(\bigwedge_{a\in V} a)$. Hence also $C_{i-1} \cup \{\text{rg}_o(\bigwedge_{a\in V} a)\}$ is satisfiable. Hence when constructing C_i the algorithm removes all clauses $\neg b_1 \vee \cdots \vee \neg b_j$ such that $\{b_1, \ldots, b_j\} \subseteq V$. Hence by Remark A $C_i \cup V$ is satisfiable.
2. Assume $G^n(V) = i \geq 1$. Then $G^n(B) \geq i - 1$ for all h-states B and operators that reach V from B.
   If $i > 1$, then by the induction hypothesis $C_{i-2} \cup B$ is unsatisfiable for any such B, and there is a clause $\neg b_1 \vee \cdots \vee \neg b_j \in C_{i-2}$ such that $\{b_1, \ldots, b_j\} \subseteq B$. Hence $C_{i-2} \cup \{\text{rg}_o(\bigwedge_{a\in V} a)\}$ is unsatisfiable for every $o \in O$. Therefore the clauses in C_{i-1} that contradict V are not removed, and hence $C_{i-1} \cup V$ is unsatisfiable.
   If $i = 1$, then $G^n(V) > 0$ because $V \not\subseteq I$, and hence $C_0 \cup V$ is unsatisfiable.
   Hence $C_{i-1} \cup V$ is in both cases unsatisfiable. □

4 Regression for Non-Deterministic Operators

Based on the regression operation for deterministic operators in Definition 7 regression for a class of nondeterministic operators can be defined. The operators' effects have nondeterministic choice $e_1|\cdots|e_n$ between two or more deterministic effects $e_1, \ldots, e_n$.

Definition 16 *Let ϕ be a formula and $o = \langle p, e_1|\cdots|e_n\rangle$ an operator where $e_1, \ldots, e_n$ are deterministic. Define*

$$rg_o^{nd}(\phi) = rg_{\langle p,e_1\rangle}(\phi) \wedge \cdots \wedge rg_{\langle p,e_n\rangle}(\phi).$$

Theorem 17 *Let ϕ be a formula over A, o an operator over A, and S the set of all states over A. Then for all $s \in S$, $s \models rg_o^{nd}(\phi)$ if and only if all possible successor states s' of s satisfy ϕ.*

Proof: This follows from the fact that each $\langle p, e_i\rangle$ represents one possible outcome the nondeterministic action may have, $\text{rg}_{\langle p,e_i\rangle}(\phi)$ represents all the states from which ϕ is reached by $\langle p, e_i\rangle$, and the intersection of these sets is exactly the set of states from which ϕ is reached no matter which outcome is the actual one. □

Example 18 *Let $o = \langle d, b|\neg c\rangle$. Then*

$$\begin{aligned} rg_o^{nd}(b \leftrightarrow c) &= rg_{\langle d,b\rangle}(b \leftrightarrow c) \wedge rg_{\langle d,\neg c\rangle}(b \leftrightarrow c) \\ &= (d \wedge (\top \leftrightarrow c)) \wedge (d \wedge (b \leftrightarrow \bot)) \\ &\equiv d \wedge c \wedge \neg b. \end{aligned}$$

■

Applications of the nondeterministic regression operation are similar to the deterministic one. Most notably, backward-search algorithms for planning with partial observability can be based on it.

5 Related Work

Regression is closely related to other forms of manipulation of formulae for computing the images or preimages of sets of states. We discuss some of the most closely related and some of the very recent related work and contrast them to regression.

5.1 Symbolic Pre-Images

General forms of reasoning about actions by the computation of *images* and *preimages*, leading to logic-based algorithms for computing sets of reachable states, has many applications and was originally introduced in the context of computer-aided verification as a technique for model-checking [3, 2]. Preimage computation is essentially regression whereas images are successors of sets of states.

Let $A = \{a_1, \ldots, a_n\}$, $A' = \{a'_1, \ldots, a'_n\}$ and $A'' = \{a''_1, \ldots, a''_n\}$. The variables in A refer to the values of state variables in a state and the variables in A' to the values in a successor state. Formulae ϕ over $A \cup A'$ can represent arbitrary binary relations on the set of all states.

The translation of a deterministic operator $\langle p, e\rangle$ into a formula is $\tau_A(o) = p \wedge \bigwedge_{a\in A} \neg(E_a(e) \wedge E_{\neg a}(e)) \wedge \bigwedge_{a\in A}(a' \leftrightarrow (E_a(e) \vee (a \wedge \neg E_{\neg a}(e))))$. The first two conjuncts express the conditions for the executability of the operator (truth of the precondition and the consistency of the effects) and the third conjunct expresses the new value of each state variable in terms of the old values of state variables.

With respect to an operator o the successor or predecessor states of a set of states, represented as a formula ϕ, can be computed by syntactic manipulation of ϕ and $\tau_A(o)$. The basic logical step in this computation is that of *existential abstraction* which eliminates the occurrences of one variable in a formula. It is defined by $\exists x.\phi = \phi[\top/x] \vee \phi[\bot/x]$ where $\phi[\theta/x]$ means replacing all occurrences of x in ϕ by θ.

Definition 19 *Let o be an operator and ϕ a formula. Define*

$$\begin{aligned} img_o(\phi) &= (\exists A.(\phi \wedge \tau_A(o)))[A/A'] \\ preimg_o(\phi) &= \exists A'.(\tau_A(o) \wedge \phi[A'/A]) \end{aligned}$$

Above $\phi[A'/A]$ denotes substitution of each $a \in A$ in ϕ by the corresponding variable $a' \in A'$.

Not surprisingly, there is a close connection between preimages and regression.

Theorem 20 *$rg_o(\phi) \equiv preimg_o(\phi)$.*

Example 21 *Let $A = \{a, b, c\}$. Let $o = \langle c, a \wedge (a \rhd b)\rangle$. Then*

$$rg_o(a \wedge b) = c \wedge (\top \wedge (b \vee a)) \equiv c \wedge (b \vee a).$$

The formula corresponding to o is

$$\tau_A(o) = c \wedge a' \wedge ((b \vee a) \leftrightarrow b') \wedge (c \leftrightarrow c').$$

The preimage of $a \wedge b$ with respect to o is represented by

$$\begin{aligned}\exists a'b'c'.(\tau_A(o) \wedge (a' \wedge b')) &\equiv \exists a'b'c'.(c \wedge a' \wedge ((b \vee a) \leftrightarrow b') \\ &\quad \wedge (c \leftrightarrow c') \wedge a' \wedge b') \\ &\equiv \exists a'b'c'.(a' \wedge b' \wedge c \wedge (b \vee a) \wedge c') \\ &\equiv \exists b'c'.(b' \wedge c \wedge (b \vee a) \wedge c') \\ &\equiv \exists c'.(c \wedge (b \vee a) \wedge c') \\ &\equiv c \wedge (b \vee a)\end{aligned}$$

■

This connection between preimages and regression is best understood based on the equivalence $a' \leftrightarrow (E_a(e) \vee (a \wedge \neg E_{\neg a}(e)))$ in the definition of $\tau_A(o)$: it corresponds to the substitution in the definition of regression. The advantage of regression is that no existential abstraction is needed, and the disadvantage is that it is restricted to operators/relations that can be represented as a conjunction of equivalences $\bigwedge_{a \in A} a' \leftrightarrow \phi_a$.

5.2 C-Filter of Shahaf and Amir

Shahaf and Amir [17] present *C-Filtering* for computing (an implicit representation of) the image of a set of states with respect to a sequence of actions. Shahaf and Amir hint at a connection between C-Filtering and regression but do not clarify it. The C-Filter is simply the use of regression to test facts about a belief state B reached from an initial belief state I by a sequence of actions $o_1, \ldots, o_n$. Instead of explicitly constructing B by image computation, facts relating to B are queried by regressing them to queries about the initial state. For example, to test whether $B \cap B' \neq \emptyset$ for some belief state B' expressed as a formula ϕ, test the non-emptiness of the intersection by a satisfiability test of $I \wedge \mathrm{rg}_{o_1;\ldots;o_n}(\phi)$.

Shahaf and Amir claim as the novelty of C-Filtering the incremental construction of the substitutions $\mathrm{rg}_{o_1;\ldots;o_n}(a)/a$ as the action sequence $o_1, \ldots, o_n, \ldots$ progresses as well as the representation of the required formulae as Boolean circuits.

6 Conclusions

We have defined regression and composition operations for PDDL operators and a regression operation for nondeterministic actions. We have also discussed applications of general regression operations in connection with macro-actions, elimination of irredundant operators, invariants and heuristics. In particular, we gave an algorithm for computing invariants for a general definition of actions that includes disjunctive preconditions and conditional effects. The algorithm is powerful yet conceptually extremely simple, and its power can be traded to efficiency by controlling the accuracy and asymptotic runtime of approximate satisfiability tests. The algorithm also yields a generalization of the h^n heuristic [7].

Acknowledgements

The research was funded by Australian Government's Department of Broadband, Communications and the Digital Economy and the Australian Research Council through NICTA.

REFERENCES

[1] Avrim L. Blum and Merrick L. Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, **90**(1-2), 281–300, (1997).

[2] J. R. Burch, E. M. Clarke, D. E. Long, K. L. MacMillan, and D. L. Dill, 'Symbolic model checking for sequential circuit verification', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **13**(4), 401–424, (1994).

[3] Olivier Coudert, Christian Berthet, and Jean Christophe Madre, 'Verification of synchronous sequential machines based on symbolic execution', in *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, ed., Joseph Sifakis, volume 407 of *Lecture Notes in Computer Science*, pp. 365–373. Springer-Verlag, (1990).

[4] Edsger W. Dijkstra, 'Guarded commands, nondeterminacy and formal derivation of programs', *Communications of the ACM*, **18**(8), 453–457, (1975).

[5] Alfonso Gerevini and Lenhart Schubert, 'Inferring state constraints for domain-independent planning', in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pp. 905–912. AAAI Press, (1998).

[6] Alfonso Gerevini and Lenhart K. Schubert, 'Discovering state constraints in DISCOPLAN: Some new results', in *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000) and the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-2000)*, pp. 761–767. AAAI Press, (2000).

[7] Patrik Haslum and Héctor Geffner, 'Admissible heuristics for optimal planning', in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, eds., Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, pp. 140–149. AAAI Press, (2000).

[8] Patrik Haslum and Peter Jonsson, 'Planning with reduced operator sets', in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, eds., Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, pp. 150–158. AAAI Press, (2000).

[9] C. A. R. Hoare, 'An axiomatic basis for computer programming', *Communications of the ACM*, **12**(10), 576–580, (1969).

[10] Glenn A. Iba, 'A heuristic approach to the discovery of macro-operators', *Journal of Machine Learning*, **3**(4), 285–317, (1989).

[11] Henry Kautz and Bart Selman, 'Planning as satisfiability', in *Proceedings of the 10th European Conference on Artificial Intelligence*, ed., Bernd Neumann, pp. 359–363. John Wiley & Sons, (1992).

[12] Fangzhen Lin, 'Discovering state invariants', in *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004)*, eds., Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, pp. 536–544. AAAI Press, (2004).

[13] Drew McDermott, 'The Planning Domain Definition Language', Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University, (October 1998).

[14] Bernhard Nebel, 'On the compilability and expressive power of propositional planning formalisms', *Journal of Artificial Intelligence Research*, **12**, 271–315, (2000).

[15] Edwin P. D. Pednault, 'ADL and the state-transition model of action', *Journal of Logic and Computation*, **4**(5), 467–512, (1994).

[16] Jussi Rintanen, 'A planning algorithm not based on directional search', in *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, eds., A. G. Cohn, L. K. Schubert, and S. C. Shapiro, pp. 617–624. Morgan Kaufmann Publishers, (June 1998).

[17] Dafna Shahaf and Eyal Amir, 'Logical circuit filtering', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, ed., Manuela Veloso, pp. 2611–2618. AAAI Press, (2007).