Invariant formulae

- Connection to reachability and the existence of plans
- An algorithm for computing invariants
- Application to planning by propositional satisfiability and regression.

No lectures on Monday May 30 and Wednesday June 2 (Pfingsten), Monday June 7 and Wednesday June 9.

Invariants: definition

A formula ϕ is an invariant of problem instance $\langle P, I, O, G \rangle$ if

1. $I \models \phi$, and
2. for every $o \in O$ and state s such that $s \models \phi$, also $\text{app}_o(s) \models \phi$.

$\Longrightarrow$ ϕ is true in every state that is reachable from I by some sequence of operators.

Invariants: the strongest invariant

A formula ϕ is *the strongest invariant* if for any invariant ψ, $\phi \models \psi$.

The strongest invariant *exactly characterizes* the set S of all states reachable from I with operators $o \in O$:
For all states s, $s \models \phi$ if and only if $s \in S$.

(Actually, there are several strongest invariants, but they are all logically equivalent.)

Invariants: an example (blocks world)

$\langle \text{ontable}(x) \wedge \text{clear}(x) \wedge \text{clear}(y), \text{on}(x,y) \wedge \neg\text{clear}(y) \wedge \neg\text{ontable}(x) \rangle$
$\langle \text{clear}(x) \wedge \text{on}(x,y), \text{ontable}(x) \wedge \text{clear}(y) \wedge \neg\text{on}(x,y) \rangle$
$\langle \text{clear}(x) \wedge \text{on}(x,y) \wedge \text{clear}(z), \text{on}(x,z) \wedge \text{clear}(y) \wedge \neg\text{clear}(z) \wedge \neg\text{on}(x,$

$\text{clear}(x) \leftrightarrow \forall y \in X \backslash \{x\}. \neg\text{on}(y,x)$
$\text{ontable}(x) \leftrightarrow \forall y \in X \backslash \{x\}. \neg\text{on}(x,y)$
$\neg\text{on}(x,y) \vee \neg\text{on}(x,z)$ when $y \neq z$
$\neg\text{on}(y,x) \vee \neg\text{on}(z,x)$ when $y \neq z$
$\neg(\text{on}(x_1,x_2) \wedge \text{on}(x_2,x_3) \wedge \cdots \wedge \text{on}(x_{n-1},x) \wedge \text{on}(x_n,x_1))$
for every $n \geq 1$ and $\{x_1, \ldots, x_n\} \subseteq X$

Invariants: connection to plan existence

Let ϕ be the strongest invariant for $\langle P, I, O, G\rangle$. Then $\langle P, I, O, G\rangle$ has a plan if and only if $G \wedge \phi$ is satisfiable (the set of goal states and the set of reachable states intersect.)

THEOREM Computing the strongest invariant ϕ is PSPACE-hard.

PROOF

Fact 1: Testing existence of plans with 1-literal goal A is PSPACE-hard. (TM simulation with one accepting state.)

proof continues...

Let $o = \langle A, p_1 \wedge \cdots \wedge p_n\rangle$ with $P = \{p_1, \ldots, p_n\}$.

For $\langle P, I, O, A\rangle$ a plan exists
iff for $\langle P, I, O \cup \{o\}, A\rangle$ a plan exists
iff for $\langle P, I, O \cup \{o\}, A \wedge p_1 \wedge \cdots \wedge p_n\rangle$ a plan exists.

Testing satisfiability of $\phi \wedge A \wedge p_1 \wedge \cdots \wedge p_n$ (*exactly one goal state!*) can be done in polynomial time: replace state variables in ϕ by $\top$ and simplify.

$\Longrightarrow$ Plan existence is polynomial-time reducible to computing the strongest invariant. $\Longrightarrow$ The latter is PSPACE-hard. Q.E.D.

Invariant computation: informally

Similar to distance estimation: compute sets C_i characterizing (giving *an upper bound*!) states reachable by i steps:

$$\begin{aligned}
C_0 &= \{A, \neg B, C\} \sim \{101\} \\
C_1 &= \{A \vee B, \neg A \vee \neg B, C\} \sim \{101, 011\} \\
C_2 &= \{\neg A \vee \neg B, C\} \sim \{001, 011, 101\} \\
C_3 &= \{\neg A \vee \neg B, C \vee A\} \sim \{001, 011, 100, 101\} \\
C_4 &= \{\neg A \vee \neg B\} \sim \{000, 001, 010, 011, 100, 101\} \\
C_5 &= \{\neg A \vee \neg B\} \sim \{000, 001, 010, 011, 100, 101\} \\
C_i &= C_5 \text{ for all } i \geq 5
\end{aligned}$$

Invariant computation: informally

1. Start with all 1-literal clauses that are true in the initial state.
2. Repeatedly test every operator vs. every clause, whether the clause can be shown to be true after applying the operator:
   - One of the literals in the clause is necessarily true: retain.
   - Otherwise, if the clause is too long: forget it.
   - Otherwise, generate new clauses by adding literals that are now true.
3. When no change, stop $\Longrightarrow$ All clauses are invariants.

Invariant computation: function simplepreserved

PROCEDURE simplepreserved($l_1 \vee \cdots \vee l_n$,Δ,$\langle l'_1 \wedge \cdots \wedge l'_{n'}, l''_1 \wedge \cdots \wedge l''_{n''} \rangle$);
IF $\{\overline{l'''_1}, \cdots, \overline{l'''_m}\} \subseteq \{l'_1, \ldots, l'_{n'}\}$ for some $l'''_1 \vee \cdots \vee l'''_m \in \Delta$ *THEN RETURN* true;
FOR EACH $l \in \{l_1, \ldots, l_n\}$ *DO*
    IF $\bar{l} \notin \{l''_1, \ldots, l''_{n''}\}$ *THEN GOTO* OK;
    FOR EACH $l' \in \{l_1, \ldots, l_n\} \backslash \{l\}$ *DO*
        IF $l' \in \{l''_1, \ldots, l''_{n''}\}$ *THEN GOTO* OK;
        IF $l' \in \{l'_1, \ldots, l'_{n'}\}$ *OR* $\overline{l'''_1} \vee \cdots \vee \overline{l'''_m} \vee l' \in \Delta$ for some $\{l'''_1, \ldots, l'''_m\} \subseteq \{l'_1, \ldots, l'_{n'}\}$,
            AND $\overline{l'} \notin \{l''_1, \ldots, l''_{n''}\}$
        THEN GOTO OK;
    END DO
    RETURN false;
    OK:
END DO
RETURN true;

Invariant computation: function simplepreserved

Let $\Delta = \{C \vee B\}$.

simplepreserved($A \vee B$, Δ, $\langle \neg C, C \wedge D \rangle$) returns *true*

simplepreserved($A \vee B$, Δ, $\langle \neg C, \neg A \wedge B \rangle$) returns *true*

simplepreserved($A \vee B$, Δ, $\langle B, \neg A \rangle$) returns *true*

simplepreserved($A \vee B$, Δ, $\langle \neg C, \neg A \rangle$) returns *true*

Invariant computation: function simplepreserved

LEMMA

Let Δ be a set of clauses, $\phi = l_1 \vee \cdots \vee l_n$ a clause, and o an operator of the form $\langle l'_1 \wedge \cdots \wedge l'_{n'}, l''_1 \wedge \cdots \wedge l''_{n''} \rangle$ where l'_j and l''_k are literals.

If simplepreserved(ϕ,Δ,o) returns *true*, then *app*$_o(s) \models \phi$ for any state s such that $s \models \Delta \cup \{\phi\}$ and o is applicable in s. (It may under these conditions also return *false*).

Invariant computation: the main procedure

PROCEDURE invariants(P, I, O, n);
C := $\{p \in P | I \models p\} \cup \{\neg p | p \in P, I \not\models p\}$;
REPEAT
    C' := C;
    FOR EACH $l_1 \vee \cdots \vee l_m \in C$ *AND* $o \in O$ *DO*
        IF not preserved($l_1 \vee \cdots \vee l_m$,C',o) *THEN*
            BEGIN
                C := $C \backslash \{l_1 \vee \cdots \vee l_m\}$;
                IF $m < n$ *THEN*
                    FOR EACH $p \in P$ *DO*
                        C := $C \cup \{l_1 \vee \cdots \vee l_m \vee p, \;\; l_1 \vee \cdots \vee l_m \vee \neg p\}$;
            END
UNTIL $C = C'$;
RETURN C;

Invariant computation: example

$I(A) = 1, I(B) = 0, I(C) = 0$

operators $o_1 = \langle A, \neg A \wedge B \rangle$, $o_2 = \langle B, \neg B \wedge C \rangle$, $o_3 = \langle C, \neg C \wedge A \rangle$

Compute invariants with at most 2 literals:

$$\begin{aligned} C_0 &= \{A, \neg B, \neg C\} \\ C_1 &= \{\neg \mathbf{C}, \mathbf{A} \vee \mathbf{B}, \neg \mathbf{B} \vee \neg \mathbf{A}\} \\ C_2 &= \{\neg B \vee \neg A, \neg \mathbf{C} \vee \neg \mathbf{A}, \neg \mathbf{C} \vee \neg \mathbf{B}\} \\ C_3 &= \{\neg B \vee \neg A, \neg C \vee \neg A, \neg C \vee \neg B\} \\ C_3 &= C_1 \end{aligned}$$

Invariant computation: general algorithm

PROCEDURE preserved($l_1 \vee \cdots \vee l_n$,Δ,$\langle c, e\rangle$);
IF $\Delta \models \neg c$ *THEN RETURN* true;
FOR EACH $l \in \{l_1, \ldots, l_n\}$ *DO*
  IF $\Delta \wedge \{EPC_{\bar{l}}(e)\} \models \bot$ *THEN GOTO* OK;
  FOR EACH $l' \in \{l_1, \ldots, l_n\} \backslash \{l\}$ *DO*
    IF $\Delta \cup \{EPC_{\bar{l}}(e), c\} \models EPC_{l'}(e)$ *THEN GOTO* OK;
    IF $\Delta \cup \{EPC_{\bar{l}}(e), c\} \models l'$ *AND* $\Delta \cup \{EPC_{\bar{l}}(e), c\} \models \neg EPC_{\bar{l'}}(e)$
    THEN GOTO OK;
  END DO
  RETURN false;
  OK:
END DO
RETURN true;

Invariant computation: general algorithm

The procedure *preserved* runs in polynomial time in the size of the clause, Δ and the operator, *except that the logical consequence tests need exponential time in the worst case.*

In the lecture notes we present an algorithm that runs in polynomial time and approximates logical consequence testing: these tests may fail in one direction without making invariant computation incorrect. (Computation of all invariants is not guaranteed anyway.)

Invariants: application in planning in the propositional logic

For every invariant $l_1 \vee \cdots \vee l_n$ add the clauses

$$l_1^t \vee \cdots \vee l_n^t$$

for all time points $t \geq 0$.

This may speed up planning a lot.

Invariants: application in backward planning

In backward search, the set of goal states and states obtained by regression often contain undesireable states:

Regression of	*in(A,Freiburg)*
by	⟨in(A,Strassburg), ¬in(A,Strassburg)∧in(A,Paris)⟩
gives	*in(A,Freiburg)∧in(A,Strassburg)*

The formula *in(A,Freiburg)∧in(A,Strassburg)* represents also states that are intuitively incorrect.

Invariants: application to backward planning

Problem: regression produces sets of states S such that

1. some states in S are not reachable from I,
2. none of the states in S are reachable from I.

The first problem would require the strongest invariant.

Partial solution to the second problem:

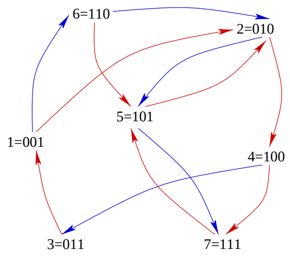
1. Compute invariant ϕ.
2. Do only regression steps such that $\mathrm{regr}_o(\psi) \wedge \phi$ is satisfiable.

Invariants: application to distance estimation

A formula ϕ has distance $> i$ if $C_i \cup \{\phi\}$ is not satisfiable.

This estimate can be much better than the one given by the sets of literals produced the first algorithm we gave for distance estimation.

Invariants: application to distances, example



distance	clauses true	
0	$\neg B_2$ $\neg B_1$	B_0
1	$\neg B_2 \vee B_1$	$\neg B_2 \vee \neg B_0$
	$\neg B_1 \vee \neg B_0$	$B_0 \vee B_1$
2	$\neg B_1 \vee \neg B_0$	
3		