Example

Consider the goal formula

$$AonB \wedge BonC$$

regressed with operator

$$\langle AonC \wedge Aclear \wedge Bclear, AonB \wedge \neg Bclear \wedge Cclear \rangle$$

giving new goal

$$AonC \wedge Aclear \wedge Bclear \wedge BonC.$$

It is intuitively clear that no state satisfying this formula is reachable by any plan from a legal blocks world state.

- Goal formulae and formulae obtained by regression from them often represent some states that are not reachable from the initial state.
- If none of the states is reachable from the initial state because there are no plans reaching the formula.
- We would like to have reachable states only, if possible.
- Same problem shows up in satisfiability planning: partial valuations considered by satisfiability algorithms may represent unreachable states, and this may result in unnecessary search.

Invariants

Goal: Restriction to states that are reachable.

Problem: Testing reachability is computationally as complex as testing whether a plan exists.

Solution: Use an approximate notion of reachability.

Implementation: Compute in polynomial time formulae that characterize a superset of the reachable states.

Invariants: definition

Definition

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

1. $I \models \phi$, and
2. for every $o \in O$ and state s such that $s \models \phi$ and s is reachable from I, also $app_o(s) \models \phi$.

Stated differently...

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

Example

If $l \in D_i^{max}$ for all $i \geq 1$ then l is an invariant.
Hence our algorithm for computing the sets D_i^{max} is a method for identifying a restricted class of invariants.

Invariants: the strongest invariant

Definition

An invariant ϕ is the strongest invariant of $\langle A, I, O, G \rangle$ if for any invariant ψ, $\phi \models \psi$.

The strongest invariant exactly characterizes the set of all states that are reachable from the initial state:
For all states s, $s \models \phi$ if and only if s is reachable.

Remark

There are infinitely many strongest invariants, but they are all logically equivalent. (If ϕ is a strongest invariant, then so is $\phi \vee \phi$...)

Invariants

Example: the strongest invariant for blocks world

The strongest invariant for the blocks world

Let X be the set of blocks, for example $X = \{A, B, C, D\}$. The conjunction of the following formulae is the strongest invariant for the set of all states for the blocks X.

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

Invariants: connection to plan existence

Theorem

Let ϕ be the strongest invariant for $\langle A, I, O, G\rangle$. Then $\langle A, I, O, G\rangle$ has a plan if and only if $G \wedge \phi$ is satisfiable.

Proof.

Very easy! $\square$

Theorem

Computing the strongest invariant ϕ is PSPACE-hard.

Proof.

By reduction from the plan existence problem.
Fact: Testing plan existence is PSPACE-hard for $\langle A, I, O, G\rangle$ even when $G = q$ for a state variable $q \in A$. (We'll show this in two weeks!)

Invariants: connection to plan existence

Proof continues..

Let $o = \langle q, a_1 \wedge \cdots \wedge a_n \rangle$ with $A = \{a_1, \ldots, a_n, q\}$.
For $\langle A, I, O, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \wedge a_1 \wedge \cdots \wedge a_n \rangle$ a plan exists.
Testing satisfiability of $\phi \wedge q \wedge a_1 \wedge \cdots \wedge a_n$ can be done in polynomial time: replace every state variable in the strongest invariant ϕ by $\top$ and simplify, getting $\top$ or $\bot$.
So, if we had a polynomial-time algorithm for computing the strongest invariant ϕ, we could test plan existence in polynomial time.
Hence plan existence is polynomial-time reducible to computing the strongest invariant.
Since the former is PSPACE-hard also the latter is PSPACE-hard. □

Invariants: connection to plan existence

Proof continues..

Let $o = \langle q, a_1 \wedge \cdots \wedge a_n \rangle$ with $A = \{a_1, \ldots, a_n, q\}$.
For $\langle A, I, O, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \wedge a_1 \wedge \cdots \wedge a_n \rangle$ a plan exists.
Testing satisfiability of $\phi \wedge q \wedge a_1 \wedge \cdots \wedge a_n$ can be done in polynomial time: replace every state variable in the strongest invariant ϕ by $\top$ and simplify, getting $\top$ or $\bot$.
So, if we had a polynomial-time algorithm for computing the strongest invariant ϕ, we could test plan existence in polynomial time.
Hence plan existence is polynomial-time reducible to computing the strongest invariant.
Since the former is PSPACE-hard also the latter is PSPACE-hard. □

Invariants: connection to plan existence

Proof continues..

Let $o = \langle q, a_1 \wedge \cdots \wedge a_n \rangle$ with $A = \{a_1, \ldots, a_n, q\}$.
For $\langle A, I, O, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \wedge a_1 \wedge \cdots \wedge a_n \rangle$ a plan exists.
Testing satisfiability of $\phi \wedge q \wedge a_1 \wedge \cdots \wedge a_n$ can be done in polynomial time: replace every state variable in the strongest invariant ϕ by $\top$ and simplify, getting $\top$ or $\bot$.
So, if we had a polynomial-time algorithm for computing the strongest invariant ϕ, we could test plan existence in polynomial time.
Hence plan existence is polynomial-time reducible to computing the strongest invariant.
Since the former is PSPACE-hard also the latter is PSPACE-hard. □

Invariants: connection to plan existence

Proof continues..

Let $o = \langle q, a_1 \wedge \cdots \wedge a_n \rangle$ with $A = \{a_1, \ldots, a_n, q\}$.
For $\langle A, I, O, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \rangle$ a plan exists
iff for $\langle A, I, O \cup \{o\}, q \wedge a_1 \wedge \cdots \wedge a_n \rangle$ a plan exists.
Testing satisfiability of $\phi \wedge q \wedge a_1 \wedge \cdots \wedge a_n$ can be done in polynomial time: replace every state variable in the strongest invariant ϕ by $\top$ and simplify, getting $\top$ or $\bot$.
So, if we had a polynomial-time algorithm for computing the strongest invariant ϕ, we could test plan existence in polynomial time.
Hence plan existence is polynomial-time reducible to computing the strongest invariant.
Since the former is PSPACE-hard also the latter is PSPACE-hard. $\square$

Computation of invariants: informally

Similar to distance estimation with D_i^{max}: compute sets C_i of n-literal clauses characterizing (giving an upper bound!) the states that are reachable in i steps.

Example

$C_0 = \{a, \neg b, c\} \sim \{101\}$

$C_1 = \{a \vee b, \neg a \vee \neg b, c\} \sim \{101, 011\}$ — $a, \neg b$ falsified

$C_2 = \{\neg a \vee \neg b, c\} \sim \{001, 011, 101\}$ — $a \vee b$ falsified

$C_3 = \{\neg a \vee \neg b, c \vee a\} \sim \{001, 011, 100, 101\}$ — c falsified

$C_4 = \{\neg a \vee \neg b\} \sim \{000, 001, 010, 011, 100, 101\}$ — $c \vee a$ falsified

$C_5 = \{\neg a \vee \neg b\} \sim \{000, 001, 010, 011, 100, 101\}$

$C_i = C_5$ for all $i > 5$

$\neg a \vee \neg b$ is the only invariant found.

Computation of invariants: 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:
    1. One of the literals in the clause is necessarily true: retain.
    2. Otherwise, if the clause is too long: forget it.
    3. Otherwise, replace the clause by new clauses obtained by adding literals that are now true.
3. When all clauses remain, stop: they are invariants.

Example

Let $C_0 = \{\textit{Aclear}, \neg\textit{Bclear}, \textit{AonB}, \neg\textit{BonA}, \neg\textit{AonT}, \textit{BonT}\}$ and $o = \langle \textit{Aclear} \wedge \textit{AonB}, \textit{Bclear} \wedge \neg\textit{AonB} \wedge \textit{AonT}\rangle$.

1. $C_0 \cup \{\textit{Aclear} \wedge \textit{AonB}\}$ is satisfiable: o is applicable.
2. The 1-literal clauses $\neg\textit{Bclear}$, $\textit{AonB}$ and $\neg\textit{AonT}$ become false when o is applied.
3. They are not thrown away, like we did when computing D_i^{max}. They are replaced by weaker clauses.
4. Literals true after applying o in state s such that $s \models C$: $\textit{Aclear}, \textit{Bclear}, \neg\textit{AonB}, \neg\textit{BonA}, \textit{AonT}, \textit{BonT}$
5. 2-literal clauses that are weaker than $\textit{AonB}$ and now true are $\textit{AonB} \vee \textit{Aclear}, \textit{AonB} \vee \textit{Bclear}, \textit{AonB} \vee \neg\textit{AonB}, \textit{AonB} \vee \neg\textit{BonA}, \textit{AonB} \vee \textit{AonT}, \textit{AonB} \vee \textit{BonT}$.

Example

Let $C_0 = \{Aclear, \neg Bclear, AonB, \neg BonA, \neg AonT, BonT\}$ and $o = \langle Aclear \wedge AonB, Bclear \wedge \neg AonB \wedge AonT\rangle$.

1. $C_0 \cup \{Aclear \wedge AonB\}$ is satisfiable: o is applicable.
2. The 1-literal clauses $\neg Bclear$, $AonB$ and $\neg AonT$ become false when o is applied.
3. They are not thrown away, like we did when computing D_i^{max}. They are replaced by weaker clauses.
4. Literals true after applying o in state s such that $s \models C$: $Aclear, Bclear, \neg AonB, \neg BonA, AonT, BonT$
5. 2-literal clauses that are weaker than $AonB$ and now true are $AonB \vee Aclear, AonB \vee Bclear, AonB \vee \neg AonB, AonB \vee \neg BonA, AonB \vee AonT, AonB \vee BonT$.

Example

Let $C_0 = \{Aclear, \neg Bclear, AonB, \neg BonA, \neg AonT, BonT\}$ and $o = \langle Aclear \wedge AonB, Bclear \wedge \neg AonB \wedge AonT\rangle$.

1. $C_0 \cup \{Aclear \wedge AonB\}$ is satisfiable: o is applicable.
2. The 1-literal clauses $\neg Bclear$, $AonB$ and $\neg AonT$ become false when o is applied.
3. They are not thrown away, like we did when computing D_i^{max}. They are replaced by weaker clauses.
4. Literals true after applying o in state s such that $s \models C$: $Aclear, Bclear, \neg AonB, \neg BonA, AonT, BonT$
5. 2-literal clauses that are weaker than $AonB$ and now true are $AonB \vee Aclear, AonB \vee Bclear, AonB \vee \neg AonB, AonB \vee \neg BonA, AonB \vee AonT, AonB \vee BonT$.

Computation of invariants
Example

Example

Let $C_0 = \{\mathit{Aclear}, \neg \mathit{Bclear}, \mathit{AonB}, \neg \mathit{BonA}, \neg \mathit{AonT}, \mathit{BonT}\}$ and $o = \langle \mathit{Aclear} \wedge \mathit{AonB}, \mathit{Bclear} \wedge \neg \mathit{AonB} \wedge \mathit{AonT}\rangle$.

1. $C_0 \cup \{\mathit{Aclear} \wedge \mathit{AonB}\}$ is satisfiable: o is applicable.
2. The 1-literal clauses $\neg \mathit{Bclear}$, AonB and $\neg \mathit{AonT}$ become false when o is applied.
3. They are not thrown away, like we did when computing D_i^{max}. They are replaced by weaker clauses.
4. Literals true after applying o in state s such that $s \models C$: $\mathit{Aclear}, \mathit{Bclear}, \neg \mathit{AonB}, \neg \mathit{BonA}, \mathit{AonT}, \mathit{BonT}$
5. 2-literal clauses that are weaker than AonB and now true are $\mathit{AonB} \vee \mathit{Aclear}, \mathit{AonB} \vee \mathit{Bclear}, \mathit{AonB} \vee \neg \mathit{AonB}, \mathit{AonB} \vee \neg \mathit{BonA}, \mathit{AonB} \vee \mathit{AonT}, \mathit{AonB} \vee \mathit{BonT}$.

Example continues..

7. For $\neg Bclear$ and $\neg AonT$ we respectively get $\neg Bclear \vee Aclear, \neg Bclear \vee Bclear, \neg Bclear \vee \neg AonB, \neg Bclear \vee \neg BonA, \neg Bclear \vee AonT, \neg Bclear \vee BonT$ **and** $\neg AonT \vee Aclear, \neg AonT \vee Bclear, \neg AonT \vee \neg AonB, \neg AonT \vee \neg BonA, \neg AonT \vee AonT, \neg AonT \vee BonT$.
8. By eliminating logically equivalent ones, tautologies, and those that follow from those in C_0 not falsified we get $C_1 = \{Aclear, \neg BonA, BonT, AonB \vee Bclear, AonB \vee AonT, \neg Bclear \vee \neg AonB, \neg Bclear \vee AonT, \neg AonT \vee Bclear, \neg AonT \vee \neg AonB\}$ for distance 1 states.
9. The precondition of $\langle Bclear \wedge BonT \wedge Aclear, BonA \wedge \neg Aclear \wedge \neg BonT\rangle$ is satisfiable with C_1, and the set C_2 contains all invariants for 2 blocks.

Example

Let $C_i = \{\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC},$
$\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}\},$
$o = \langle \mathit{AinRome}, \mathit{AinParis} \wedge \neg \mathit{AinRome} \rangle.$

1. Does o preserve truth of $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$?
2. Because o makes $\neg \mathit{AinParis}$ false, we must show that $\neg \mathit{AinNYC}$ is true after applying o.
3. But $\neg \mathit{AinNYC}$ is not even mentioned in o!
4. However, since $\mathit{AinRome}$ is the precondition of o and $\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}$ was true before applying o, we can infer that $\neg \mathit{AinNYC}$ was true before applying o.
5. Since o does not make $\neg \mathit{AinNYC}$ false, it is true also after applying o, and then so is $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$.

Computation of invariants
Example

Example

Let $C_i = \{\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}\}$,
$o = \langle \mathit{AinRome}, \mathit{AinParis} \wedge \neg \mathit{AinRome} \rangle$.

1. Does o preserve truth of $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$?
2. Because o makes $\neg \mathit{AinParis}$ false, we must show that $\neg \mathit{AinNYC}$ is true after applying o.
3. But $\neg \mathit{AinNYC}$ is not even mentioned in o!
4. However, since $\mathit{AinRome}$ is the precondition of o and $\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}$ was true before applying o, we can infer that $\neg \mathit{AinNYC}$ was true before applying o.
5. Since o does not make $\neg \mathit{AinNYC}$ false, it is true also after applying o, and then so is $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$.

Example

Let $C_i = \{\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}\}$,
$o = \langle \mathit{AinRome}, \mathit{AinParis} \wedge \neg \mathit{AinRome} \rangle$.

1. Does o preserve truth of $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$?
2. Because o makes $\neg \mathit{AinParis}$ false, we must show that $\neg \mathit{AinNYC}$ is true after applying o.
3. But $\neg \mathit{AinNYC}$ is not even mentioned in o!
4. However, since *AinRome* is the precondition of o and $\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}$ was true before applying o, we can infer that $\neg \mathit{AinNYC}$ was true before applying o.
5. Since o does not make $\neg \mathit{AinNYC}$ false, it is true also after applying o, and then so is $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$.

Computation of invariants
Example

Example

Let $C_i = \{\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}\}$, $o = \langle \mathit{AinRome}, \mathit{AinParis} \wedge \neg \mathit{AinRome} \rangle$.

1. Does o preserve truth of $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$?
2. Because o makes $\neg \mathit{AinParis}$ false, we must show that $\neg \mathit{AinNYC}$ is true after applying o.
3. But $\neg \mathit{AinNYC}$ is not even mentioned in o!
4. However, since $\mathit{AinRome}$ is the precondition of o and $\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}$ was true before applying o, we can infer that $\neg \mathit{AinNYC}$ was true before applying o.
5. Since o does not make $\neg \mathit{AinNYC}$ false, it is true also after applying o, and then so is $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$.

Computation of invariants
Example

Example

Let $C_i = \{\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}\}$,
$o = \langle \mathit{AinRome}, \mathit{AinParis} \wedge \neg \mathit{AinRome} \rangle$.

1. Does o preserve truth of $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$?
2. Because o makes $\neg \mathit{AinParis}$ false, we must show that $\neg \mathit{AinNYC}$ is true after applying o.
3. But $\neg \mathit{AinNYC}$ is not even mentioned in o!
4. However, since *AinRome* is the precondition of o and $\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}$ was true before applying o, we can infer that $\neg \mathit{AinNYC}$ was true before applying o.
5. Since o does not make $\neg \mathit{AinNYC}$ false, it is true also after applying o, and then so is $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$.

Example

Let $C_i = \{\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}, \neg \mathit{AinParis} \vee \neg \mathit{AinNYC}\}$,
$o = \langle \mathit{AinRome}, \mathit{AinParis} \wedge \neg \mathit{AinRome} \rangle$.

1. Does o preserve truth of $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$?
2. Because o makes $\neg \mathit{AinParis}$ false, we must show that $\neg \mathit{AinNYC}$ is true after applying o.
3. But $\neg \mathit{AinNYC}$ is not even mentioned in o!
4. However, since *AinRome* is the precondition of o and $\neg \mathit{AinRome} \vee \neg \mathit{AinNYC}$ was true before applying o, we can infer that $\neg \mathit{AinNYC}$ was true before applying o.
5. Since o does not make $\neg \mathit{AinNYC}$ false, it is true also after applying o, and then so is $\neg \mathit{AinParis} \vee \neg \mathit{AinNYC}$.

Computation of invariants: procedure *preserved*

Test whether a clause remains true when operator is applied

PROCEDURE preserved($l_1 \vee \cdots \vee l_n$,C,o);
$\langle c, e \rangle$:= o;
FOR EACH $l \in \{l_1, \ldots, l_n\}$ *DO*
    IF $C \cup \{\mathit{EPC}_{\bar{l}}(o)\}$ is unsatisfiable *THEN GOTO* OK;
    FOR EACH $l' \in \{l_1, \ldots, l_n\} \backslash \{l\}$ *DO*
        IF $C \cup \{\mathit{EPC}_{\bar{l}}(o)\} \models \mathit{EPC}_{l'}(e)$ *THEN GOTO* OK;
        IF $C \cup \{\mathit{EPC}_{\bar{l}}(o)\} \models l' \wedge \neg \mathit{EPC}_{\overline{l'}}(e)$ *THEN GOTO* OK;
    END DO
    RETURN false;
    OK:
END DO
RETURN true;

Computation of invariants: function *preserved*

Let $C = \{c \vee b\}$.

1. preserved($a \vee b$, C, $\langle \neg c, c \wedge d \rangle$) returns *true*
2. preserved($a \vee b$, C, $\langle \neg c, \neg a \wedge b \rangle$) returns *true*
3. preserved($a \vee b$, C, $\langle b, \neg a \rangle$) returns *true*
4. preserved($a \vee b$, C, $\langle \neg c, \neg a \rangle$) returns *true*
5. preserved($a \vee b$, C, $\langle c, \neg a \rangle$) returns *false*

Computation of invariants: function *preserved*

Let $C = \{c \vee b\}$.

1. preserved($a \vee b$, C, $\langle \neg c, c \wedge d \rangle$) returns *true*
2. preserved($a \vee b$, C, $\langle \neg c, \neg a \wedge b \rangle$) returns *true*
3. preserved($a \vee b$, C, $\langle b, \neg a \rangle$) returns *true*
4. preserved($a \vee b$, C, $\langle \neg c, \neg a \rangle$) returns *true*
5. preserved($a \vee b$, C, $\langle c, \neg a \rangle$) returns *false*

Computation of invariants: function *preserved*

Let $C = \{c \vee b\}$.

1. preserved($a \vee b$, C, $\langle \neg c, c \wedge d \rangle$) returns *true*
2. preserved($a \vee b$, C, $\langle \neg c, \neg a \wedge b \rangle$) returns *true*
3. preserved($a \vee b$, C, $\langle b, \neg a \rangle$) returns *true*
4. preserved($a \vee b$, C, $\langle \neg c, \neg a \rangle$) returns *true*
5. preserved($a \vee b$, C, $\langle c, \neg a \rangle$) returns *false*

Computation of invariants: function *preserved*

Let $C = \{c \vee b\}$.

1. preserved($a \vee b$, C, $\langle \neg c, c \wedge d \rangle$) returns *true*
2. preserved($a \vee b$, C, $\langle \neg c, \neg a \wedge b \rangle$) returns *true*
3. preserved($a \vee b$, C, $\langle b, \neg a \rangle$) returns *true*
4. preserved($a \vee b$, C, $\langle \neg c, \neg a \rangle$) returns *true*
5. preserved($a \vee b$, C, $\langle c, \neg a \rangle$) returns *false*

Computation of invariants: function *preserved*

Let $C = \{c \vee b\}$.

1. preserved($a \vee b$, C, $\langle \neg c, c \wedge d \rangle$) returns *true*
2. preserved($a \vee b$, C, $\langle \neg c, \neg a \wedge b \rangle$) returns *true*
3. preserved($a \vee b$, C, $\langle b, \neg a \rangle$) returns *true*
4. preserved($a \vee b$, C, $\langle \neg c, \neg a \rangle$) returns *true*
5. preserved($a \vee b$, C, $\langle c, \neg a \rangle$) returns *false*

Computation of invariants: function *preserved*

Correctness

Lemma

Let C be a set of clauses, $\phi = l_1 \vee \cdots \vee l_n$ a clause, and o an operator.
If preserved(ϕ,C,o) returns true, *then app$_o(s) \models \phi$ for every state s such that $s \models C$ and app$_o(s)$ is defined.*

Computation of invariants: function *preserved*

Why is *preserved* incomplete?

Example

Let $o = \langle a, \neg b \wedge (c \rhd d) \wedge (\neg c \rhd e)\rangle$.
preserved($b \vee d \vee e$,$\emptyset$,o) returns false because it cannot prove for any literal in $b \vee d \vee e$ that it is true after application of o.
However, $d \vee e$ is true after applying o, and hence $b \vee d \vee e$ will be true as well.

Computation of invariants: the main procedure
Outline

1. C = the set of 1-literal clauses that are true in the initial state.
2. For each operator o and clause $l_1 \vee \cdots \vee l_m \in C$ test if $l_1 \vee \cdots \vee l_m$ remains true when o is applied.
3. If not, remove $l_1 \vee \cdots \vee l_m$, and if $m < n$ add clauses $l_1 \vee \cdots \vee l_m \vee a$ and $l_1 \vee \cdots \vee l_m \vee \neg a$ for every $a \in A$.
4. Repeat from step 2 if C has changed.
5. Otherwise every clause in C is an invariant.

The number of iterations is $\mathcal{O}(m^n)$ which is polynomial in the number of state variables $m = |A|$ for any fixed n.

Computation of invariants: the main procedure

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

Computation of invariants: the main procedure

Correctness

Theorem

The procedure invariants(A, I, O, n) returns a set C of clauses with at most n literals so that for any sequence $o_1, \ldots, o_m$ of operators in O app$_{o_1;\ldots;o_m}(I) \models C$.

Proof.

Let C_0 be the value first assigned to the variable C and $C_1, C_2, \ldots$ the values of C in the end of each iteration.

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

Base case $i = 0$: *app*$_\epsilon(I)$ 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.

Proof continues..

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

A Consider the case $\phi \in C_{i-1}$. By induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$. Since $\phi \in C_i$ preserved(ϕ,C_{i-1},o) returns *true*. Hence by the Lemma *app*$_{o_1;\ldots;o_i}(I) \models \phi$.

B Consider the case $\phi \notin C_{i-1}$.

1. As $\phi \notin C_{i-1}$ there is $\phi' \in C_{i-1}$ with $\phi = \phi' \vee l'_1 \vee \cdots \vee l'_m$ for some $l'_1, \cdots, l'_m$ and preserved(ϕ',C_{i-1},o') returns *false* for some $o' \in O$. Hence $\phi' \models \phi$.
2. As $\phi' \in C_{i-1}$ by induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi'$.
3. Since $\phi' \models \phi$ also *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$.
4. Since preserved(ϕ,C_i,o) returns *true* *app*$_{o_1;\ldots;o_i}(I) \models \phi$ by the Lemma.

Proof continues..

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

A. Consider the case $\phi \in C_{i-1}$. By induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$. Since $\phi \in C_i$ preserved(ϕ,C_{i-1},o) returns *true*. Hence by the Lemma *app*$_{o_1;\ldots;o_i}(I) \models \phi$.

B. Consider the case $\phi \notin C_{i-1}$.

1. As $\phi \notin C_{i-1}$ there is $\phi' \in C_{i-1}$ with $\phi = \phi' \vee l'_1 \vee \cdots \vee l'_m$ for some $l'_1, \cdots, l'_m$ and preserved(ϕ',C_{i-1},o') returns *false* for some $o' \in O$. Hence $\phi' \models \phi$.
2. As $\phi' \in C_{i-1}$ by induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi'$.
3. Since $\phi' \models \phi$ also *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$.
4. Since preserved(ϕ,C_i,o) returns *true* *app*$_{o_1;\ldots;o_i}(I) \models \phi$ by the Lemma.

□

Proof continues..

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

A. Consider the case $\phi \in C_{i-1}$. By induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$. Since $\phi \in C_i$ preserved(ϕ,C_{i-1},o) returns *true*. Hence by the Lemma *app*$_{o_1;\ldots;o_i}(I) \models \phi$.

B. Consider the case $\phi \notin C_{i-1}$.

1. As $\phi \notin C_{i-1}$ there is $\phi' \in C_{i-1}$ with $\phi = \phi' \vee l'_1 \vee \cdots \vee l'_m$ for some $l'_1, \cdots, l'_m$ and preserved(ϕ',C_{i-1},o') returns *false* for some $o' \in O$. Hence $\phi' \models \phi$.
2. As $\phi' \in C_{i-1}$ by induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi'$.
3. Since $\phi' \models \phi$ also *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$.
4. Since preserved(ϕ,C_i,o) returns *true* *app*$_{o_1;\ldots;o_i}(I) \models \phi$ by the Lemma.

Computation of invariants: the main procedure
Correctness

Proof continues..

Inductive case $i \geq 1$: Take any $\{o_1, \dots, o_i\} \subseteq O$ and $\phi \in C_i$.

A Consider the case $\phi \in C_{i-1}$. By induction hypothesis *app*$_{o_1;\dots;o_{i-1}}(I) \models \phi$. Since $\phi \in C_i$ preserved(ϕ,C_{i-1},o) returns *true*. Hence by the Lemma *app*$_{o_1;\dots;o_i}(I) \models \phi$.

B Consider the case $\phi \notin C_{i-1}$.

1. As $\phi \notin C_{i-1}$ there is $\phi' \in C_{i-1}$ with $\phi = \phi' \vee l'_1 \vee \cdots \vee l'_m$ for some $l'_1, \cdots, l'_m$ and preserved(ϕ',C_{i-1},o') returns *false* for some $o' \in O$. Hence $\phi' \models \phi$.
2. As $\phi' \in C_{i-1}$ by induction hypothesis *app*$_{o_1;\dots;o_{i-1}}(I) \models \phi'$.
3. Since $\phi' \models \phi$ also *app*$_{o_1;\dots;o_{i-1}}(I) \models \phi$.
4. Since preserved(ϕ,C_i,o) returns *true* *app*$_{o_1;\dots;o_i}(I) \models \phi$ by the Lemma.

Proof continues..

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

A Consider the case $\phi \in C_{i-1}$. By induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$. Since $\phi \in C_i$ preserved(ϕ,C_{i-1},o) returns *true*. Hence by the Lemma *app*$_{o_1;\ldots;o_i}(I) \models \phi$.

B Consider the case $\phi \notin C_{i-1}$.

1. As $\phi \notin C_{i-1}$ there is $\phi' \in C_{i-1}$ with $\phi = \phi' \vee l'_1 \vee \cdots \vee l'_m$ for some $l'_1, \cdots, l'_m$ and preserved(ϕ',C_{i-1},o') returns *false* for some $o' \in O$. Hence $\phi' \models \phi$.
2. As $\phi' \in C_{i-1}$ by induction hypothesis *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi'$.
3. Since $\phi' \models \phi$ also *app*$_{o_1;\ldots;o_{i-1}}(I) \models \phi$.
4. Since preserved(ϕ,C_i,o) returns *true* *app*$_{o_1;\ldots;o_i}(I) \models \phi$ by the Lemma.

$\square$

Why is the strongest invariant not always found?

1. Practical implementations of the algorithm use polynomial time approximations of the tests for satisfiability and $\models$.
2. The function *preserved* is incomplete for operators in general (but complete for STRIPS operators.) Making it complete makes it NP-hard.
3. The strongest invariant may require arbitrarily long clauses, so the restriction to clauses of any fixed length makes it impossible to represent it.

Example

The acyclicity of the on relation in the blocks world needs clauses of length n when there are n blocks.

Why is the strongest invariant not always found?

1. Practical implementations of the algorithm use polynomial time approximations of the tests for satisfiability and $\models$.
2. The function *preserved* is incomplete for operators in general (but complete for STRIPS operators.) Making it complete makes it NP-hard.
3. The strongest invariant may require arbitrarily long clauses, so the restriction to clauses of any fixed length makes it impossible to represent it.

Example

The acyclicity of the **on** relation in the blocks world needs clauses of length n when there are n blocks.

Why is the strongest invariant not always found?

1. Practical implementations of the algorithm use polynomial time approximations of the tests for satisfiability and $\models$.
2. The function *preserved* is incomplete for operators in general (but complete for STRIPS operators.) Making it complete makes it NP-hard.
3. The strongest invariant may require arbitrarily long clauses, so the restriction to clauses of any fixed length makes it impossible to represent it.

Example

The acyclicity of the **on** relation in the blocks world needs clauses of length n when there are n blocks.

Why is the strongest invariant not always found?

1. Practical implementations of the algorithm use polynomial time approximations of the tests for satisfiability and $\models$.
2. The function *preserved* is incomplete for operators in general (but complete for STRIPS operators.) Making it complete makes it NP-hard.
3. The strongest invariant may require arbitrarily long clauses, so the restriction to clauses of any fixed length makes it impossible to represent it.

Example

The acyclicity of the **on** relation in the blocks world needs clauses of length n when there are n blocks.

Computation of invariants
Example

Initial state: $I \models a \wedge \neg b \wedge \neg c$

Operators:
$$o_1 = \langle a, \neg a \wedge b \rangle,$$
$$o_2 = \langle b, \neg b \wedge c \rangle,$$
$$o_3 = \langle c, \neg c \wedge a \rangle$$

Computation: Find invariants with at most 2 literals:

$$C_0 = \{a, \neg b, \neg c\}$$
$$C_1 = \{\mathbf{\neg c}, \mathbf{a} \vee \mathbf{b}, \mathbf{\neg b} \vee \mathbf{\neg a}\}$$
$$C_2 = \{\neg b \vee \neg a, \mathbf{\neg c} \vee \mathbf{\neg a}, \mathbf{\neg c} \vee \mathbf{\neg b}\}$$
$$C_3 = \{\neg b \vee \neg a, \neg c \vee \neg a, \neg c \vee \neg b\}$$
$$C_j = C_2 \text{ for all } j \geq 2$$

Invariants in satisfiability planning

Invariants in satisfiability planning

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.

Notice that the above formulae logical consequences of Φ_i^{seq} and Φ_i^{par}, so the invariants do not change the set of valuations of these formulae.

Invariants are critical for the efficiency of satisfiability planning on many types of problems.

Invariants in backward search

Motivating example

Example

Regression of in(A,Freiburg) by
$\langle$in(A,Strassburg), $\neg$in(A,Strassburg)$\wedge$in(A,Paris)$\rangle$
gives in(A,Freiburg)$\wedge$in(A,Strassburg)
No state satisfying in(A,Freiburg)$\wedge$in(A,Strassburg) makes sense if A denotes some usual physical object.

Invariants in backward search
Motivating example

Problem: Regression produces sets T of states such that

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

The first is not always a serious problem (but may worsen the quality of distance estimates, for example.)

Solution: Use invariants to avoid formulae that do not represent any reachable states.

1. Compute invariant ϕ.
2. Do only regression steps such that $regr_o(\psi) \wedge \phi$ is satisfiable.

Invariants in backward search

Motivating example

Problem: Regression produces sets T of states such that

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

The first is not always a serious problem (but may worsen the quality of distance estimates, for example.)

Solution: Use invariants to avoid formulae that do not represent any reachable states.

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

Summary

- Invariants are needed for making backward search and satisfiability planning more efficient.
- We gave an algorithm for computing a class of invariants.
    1. Start with 1-literal clauses true in the initial state.
    2. Repeatedly weaken clauses that could not be shown to be invariants.
    3. Stop when all clauses are guaranteed to be invariants.
- The algorithm runs in polynomial time if the satisfiability and logical consequence tests are approximated by a polynomial time algorithm.

Summary

- Invariants are needed for making backward search and satisfiability planning more efficient.
- We gave an algorithm for computing a class of invariants.
  1. Start with 1-literal clauses true in the initial state.
  2. Repeatedly weaken clauses that could not be shown to be invariants.
  3. Stop when all clauses are guaranteed to be invariants.
- The algorithm runs in polynomial time if the satisfiability and logical consequence tests are approximated by a polynomial time algorithm.

Summary

- Invariants are needed for making backward search and satisfiability planning more efficient.
- We gave an algorithm for computing a class of invariants.
    1. Start with 1-literal clauses true in the initial state.
    2. Repeatedly weaken clauses that could not be shown to be invariants.
    3. Stop when all clauses are guaranteed to be invariants.
- The algorithm runs in polynomial time if the satisfiability and logical consequence tests are approximated by a polynomial time algorithm.

Summary

- Invariants are needed for making backward search and satisfiability planning more efficient.
- We gave an algorithm for computing a class of invariants.
  1. Start with 1-literal clauses true in the initial state.
  2. Repeatedly weaken clauses that could not be shown to be invariants.
  3. Stop when all clauses are guaranteed to be invariants.
- The algorithm runs in polynomial time if the satisfiability and logical consequence tests are approximated by a polynomial time algorithm.

Summary

- Invariants are needed for making backward search and satisfiability planning more efficient.
- We gave an algorithm for computing a class of invariants.
  1. Start with 1-literal clauses true in the initial state.
  2. Repeatedly weaken clauses that could not be shown to be invariants.
  3. Stop when all clauses are guaranteed to be invariants.
- The algorithm runs in polynomial time if the satisfiability and logical consequence tests are approximated by a polynomial time algorithm.

Summary

- Invariants are needed for making backward search and satisfiability planning more efficient.
- We gave an algorithm for computing a class of invariants.
  1. Start with 1-literal clauses true in the initial state.
  2. Repeatedly weaken clauses that could not be shown to be invariants.
  3. Stop when all clauses are guaranteed to be invariants.
- The algorithm runs in polynomial time if the satisfiability and logical consequence tests are approximated by a polynomial time algorithm.