University of Basel

Certifying Unsolvability using CNF Formulas

Fabian Kruse <fabian.kruse@unibas.ch>

Department of Mathematics and Computer Science, University of Basel

23. Feburary 2023

STRIPS Planning Task

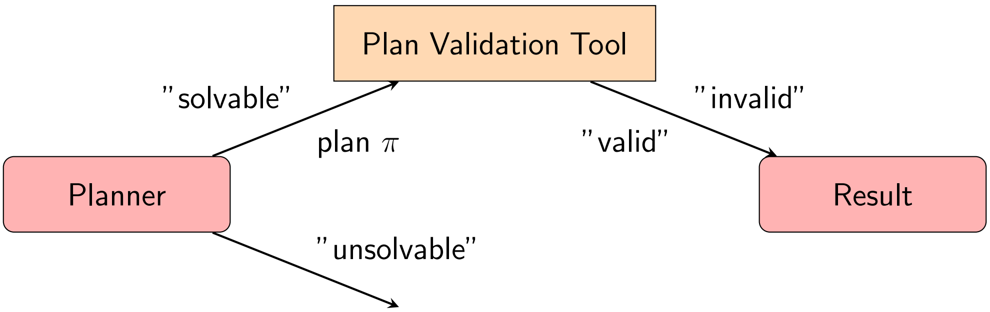
Definition

A **STRIPS planning task** Π is defined as $\Pi = \langle V^\Pi, A^\Pi, I^\Pi, G^\Pi \rangle$ where

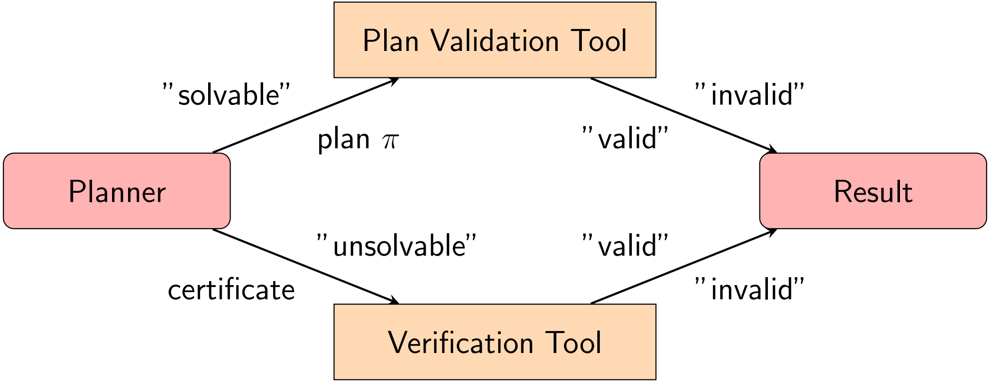
- V^Π is a finite set of propositional variables
- A^Π is a finite set of actions
- $I^\Pi \subseteq V^\Pi$ is the initial state
- $G^\Pi \subseteq V^\Pi$ is the goal

A subset $s \subseteq V^\Pi$ is called a **state** of Π. The set of all states of Π is denoted by S^Π.

Certifying Planning Systems

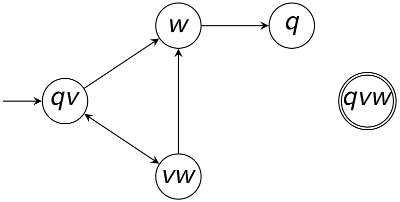


Certifying Planning Systems



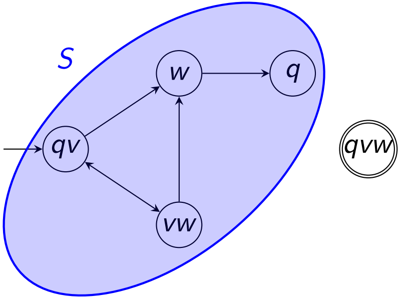
Inductive Certificates

Definition



Inductive Certificates

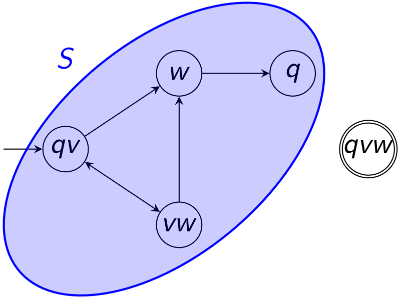
Definition



Inductive Certificates

Definition

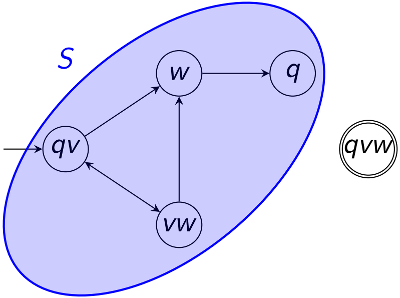
- $I^\Pi \in S$



Inductive Certificates

Definition

- $I^{\Pi} \in S$
- $S \cap S_G^{\Pi} = \emptyset$

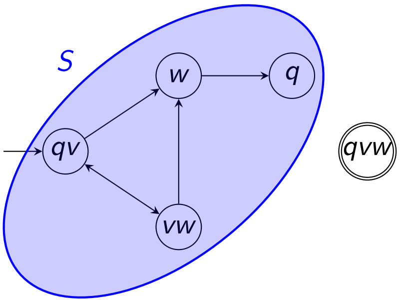


Inductive Certificates

"cannot be left"

Definition

- $I^\Pi \in S$
- $S \cap S_G^\Pi = \emptyset$
- S is inductive in Π



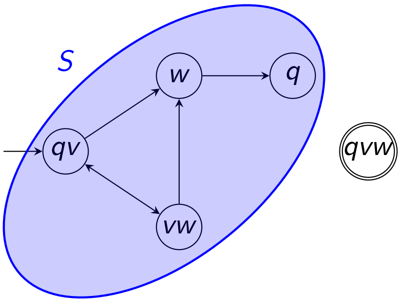
Inductive Certificates

"cannot be left"

Definition

An **inductive certificate** for planning task Π is given by a set $S \subseteq S^\Pi$ of states, such that

- $I^\Pi \in S$
- $S \cap S_G^\Pi = \emptyset$
- S is inductive in Π



Inductive Certificates

"cannot be left"

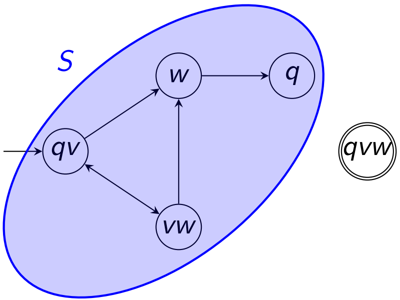
Definition

An **inductive certificate** for planning task Π is given by a set $S \subseteq S^{\Pi}$ of states, such that

- $I^{\Pi} \in S$
- $S \cap S_G^{\Pi} = \emptyset$
- S is inductive in Π

Theorem

Planning task Π is unsolvable iff there exists an inductive certificate for Π



Conjunctive Normal Form (CNF)

A finite conjunction of clauses is a formula in **conjunctive normal form** (CNF).

$$\varphi = \bigwedge \bigvee lit$$

- Widely studied and commonly used in Computer Science
- Testing a CNF formula for satisfiability is **NP**-complete

Conjunctive Normal Form (CNF)

A finite conjunction of clauses is a formula in **conjunctive normal form** (CNF).

$$\varphi = \bigwedge \bigvee lit$$

- Widely studied and commonly used in Computer Science
- Testing a CNF formula for satisfiability is **NP**-complete

Conjunctive Normal Form (CNF)

A finite conjunction of clauses is a formula in **conjunctive normal form** (CNF).

$$\varphi = \bigwedge \bigvee lit$$

- Widely studied and commonly used in Computer Science
- Testing a CNF formula for satisfiability is **NP**-complete

Why CNF?

- STRIPS problem descriptions are very close to propositional logic
  - e.g. state $s = \{v, w\}$ over variables $V^\Pi = \{q, v, w\}$ described by $\varphi_s = v \land w \land \neg q$
- SAT-solver allow certified verification

But: SAT-solving is **NP**-complete

- However: SAT-solvers are often much more efficient for "real" problems

Thesis investigates feasibility of CNF formalism

Why CNF?

- STRIPS problem descriptions are very close to propositional logic
  - e.g. state $s = \{v, w\}$ over variables $V^\Pi = \{q, v, w\}$ described by $\varphi_s = v \land w \land \neg q$
- SAT-solver allow certified verification

But: SAT-solving is **NP**-complete

- However: SAT-solvers are often much more efficient for "real" problems

Thesis investigates feasibility of CNF formalism

Why CNF?

- STRIPS problem descriptions are very close to propositional logic
  - e.g. state $s = \{v, w\}$ over variables $V^{\Pi} = \{q, v, w\}$ described by $\varphi_s = v \wedge w \wedge \neg q$
- SAT-solver allow certified verification

But: SAT-solving is **NP**-complete

- However: SAT-solvers are often much more efficient for "real" problems

Thesis investigates feasibility of CNF formalism

Why CNF?

- STRIPS problem descriptions are very close to propositional logic
  - e.g. state $s = \{v, w\}$ over variables $V^{\Pi} = \{q, v, w\}$ described by $\varphi_s = v \wedge w \wedge \neg q$
- SAT-solver allow certified verification

But: SAT-solving is **NP**-complete

- However: SAT-solvers are often much more efficient for "real" problems

Thesis investigates feasibility of CNF formalism

Why CNF?

- STRIPS problem descriptions are very close to propositional logic
  - e.g. state $s = \{v, w\}$ over variables $V^{\Pi} = \{q, v, w\}$ described by $\varphi_s = v \land w \land \neg q$
- SAT-solver allow certified verification

But: SAT-solving is **NP**-complete

- However: SAT-solvers are often much more efficient for "real" problems

Thesis investigates feasibility of CNF formalism

Why CNF?

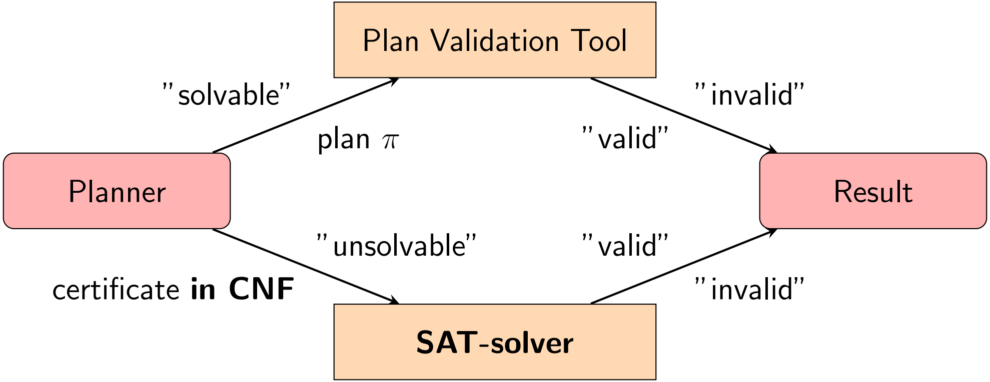
- STRIPS problem descriptions are very close to propositional logic
  - e.g. state $s = \{v, w\}$ over variables $V^{\Pi} = \{q, v, w\}$ described by $\varphi_s = v \land w \land \neg q$
- SAT-solver allow certified verification

But: SAT-solving is **NP**-complete

- However: SAT-solvers are often much more efficient for "real" problems

Thesis investigates feasibility of CNF formalism

Certifying Unsolvability using CNF Formulas



Generate Inductive Certificate Formulas

Formula φ_S should represent the set of reachable states S

In blind search: all reachable states are expanded

- Start with $\varphi_S := \bot$
- During search: append each expanded state s
  $$\varphi_S = \varphi_S \vee \underbrace{(\bigwedge_{v \in s} v \wedge \bigwedge_{v \notin s} \neg v)}_{\varphi_s}$$

Generate Inductive Certificate Formulas

Formula φ_S should represent the set of reachable states S

In blind search: all reachable states are expanded

- Start with $\varphi_S := \perp$
- During search: append each expanded state s

  $$\varphi_S = \varphi_S \vee \underbrace{\Big(\bigwedge_{v \in s} v \wedge \bigwedge_{v \notin s} \neg v\Big)}_{\varphi_s}$$

Generate Inductive Certificate Formulas

Formula φ_S should represent the set of reachable states S

In blind search: all reachable states are expanded

- Start with $\varphi_S := \bot$
- During search: append each expanded state s
  $\varphi_S = \varphi_S \vee \underbrace{(\bigwedge_{v \in s} v \wedge \bigwedge_{v \notin s} \neg v)}_{\varphi_s}$

Generate Inductive Certificate Formulas

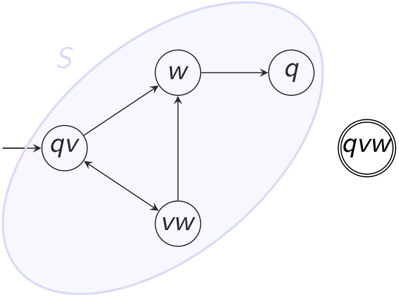
Formula φ_S should represent the set of reachable states S

In blind search: all reachable states are expanded

- Start with $\varphi_S := \bot$
- During search: append each expanded state s
  $$\varphi_S = \varphi_S \lor \underbrace{(\bigwedge_{v \in s} v \land \bigwedge_{v \notin s} \neg v)}_{\varphi_s}$$

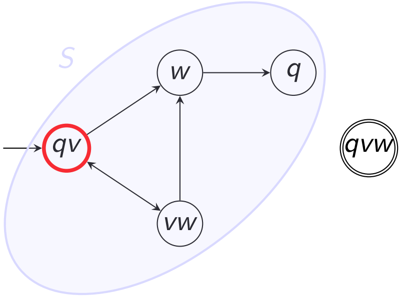
Blind Search

$$\varphi_S = \bot$$



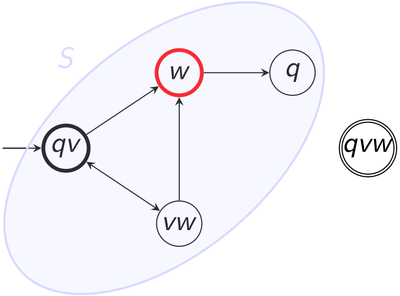
Blind Search

$$\varphi_S = (q \wedge v \wedge \neg w)$$



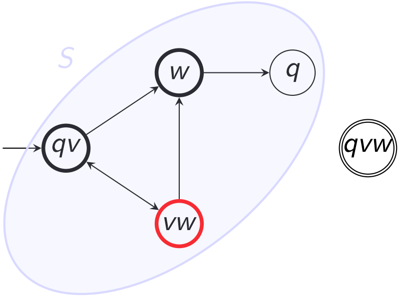
Blind Search

$$\varphi_S = (q \land v \land \neg w) \lor (w \land \neg q \land \neg v)$$



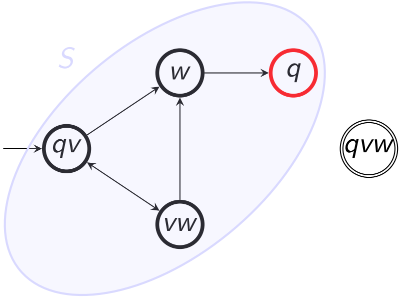
Blind Search

$$\varphi_S = (q \land v \land \neg w) \lor (w \land \neg q \land \neg v) \lor (v \land w \land \neg q)$$



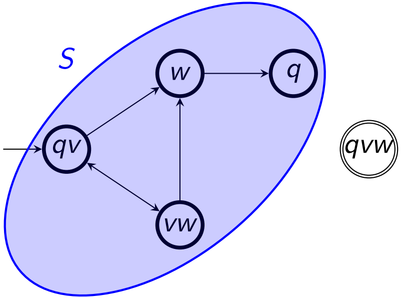
Blind Search

$$\varphi_S = (q \land v \land \neg w) \lor (w \land \neg q \land \neg v) \lor (v \land w \land \neg q) \lor (q \land \neg v \land \neg w)$$



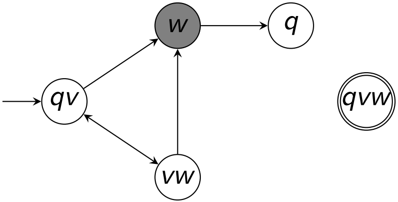
Blind Search

$$\varphi_S = (q \land v \land \neg w) \\ \lor (w \land \neg q \land \neg v) \\ \lor (v \land w \land \neg q) \\ \lor (q \land \neg v \land \neg w)$$



φ_S describes the inductive certificate since $\forall s \in S : s \models \varphi_S$

Heuristic Search

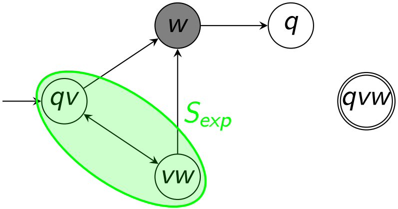


- Infinite heuristic values may prune the search space
  $\rightarrow$ We don't expand all reachable states
  $\rightarrow$ S_{exp} is not inductive

How to regain inductivity?

- Assume we have an inductive set R_{s_d} for each dead-end s_d
- Expanded states lead to expanded states and dead-ends

Heuristic Search

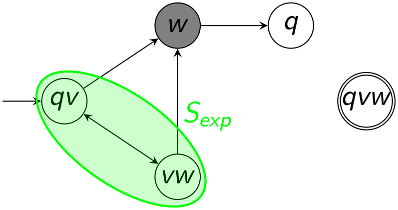


- Infinite heuristic values may prune the search space
  $\rightarrow$ We don't expand all reachable states
  $\rightarrow$ S_{exp} is not inductive

How to regain inductivity?

- Assume we have an inductive set R_{s_d} for each dead-end s_d
- Expanded states lead to expanded states and dead-ends

Heuristic Search

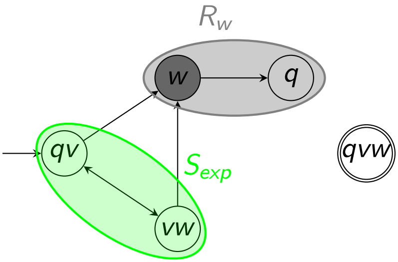


- Infinite heuristic values may prune the search space
  $\rightarrow$ We don't expand all reachable states
  $\rightarrow$ S_{exp} is not inductive

How to regain inductivity?

- Assume we have an inductive set R_{s_d} for each dead-end s_d
- Expanded states lead to expanded states and dead-ends

Heuristic Search

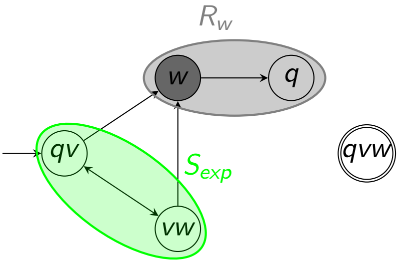


- Infinite heuristic values may prune the search space
  $\rightarrow$ We don't expand all reachable states
  $\rightarrow$ S_{exp} is not inductive

How to regain inductivity?

- Assume we have an inductive set R_{s_d} for each dead-end s_d
- Expanded states lead to expanded states and dead-ends

Heuristic Search

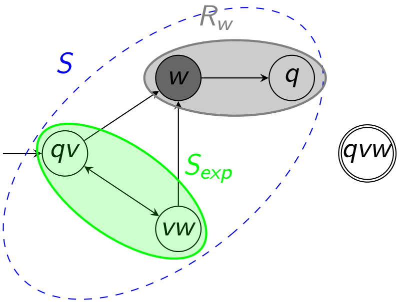


- Infinite heuristic values may prune the search space
  $\rightarrow$ We don't expand all reachable states
  $\rightarrow$ S_{exp} is not inductive

How to regain inductivity?

- Assume we have an inductive set R_{s_d} for each dead-end s_d
- Expanded states lead to expanded states and dead-ends

Heuristic Search



- Infinite heuristic values may prune the search space
  $\rightarrow$ We don't expand all reachable states
  $\rightarrow$ S_{exp} is not inductive

How to regain inductivity?

- Assume we have an inductive set R_{s_d} for each dead-end s_d
- Expanded states lead to expanded states and dead-ends
  $\rightarrow$ $S = S_{exp} \cup R_w$ is inductive

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

$$\varphi_{goal} := \varphi_G \wedge \varphi_S = \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

$$\varphi_{goal} := \varphi_G \wedge \varphi_S = \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

$$\varphi_{goal} := \varphi_G \wedge \varphi_S = \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

$$\varphi_{goal} := \varphi_G \wedge \varphi_S = \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

$$\varphi_{goal} := \varphi_G \wedge \varphi_S$$

$$= \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

$$\varphi_{goal} := \varphi_G \wedge \varphi_S = \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

Validation Formula

Idea: Represent the properties of the inductive certificate in a single formula φ_V

- The planner found a valid inductive certificate iff φ_V unsatisfiable

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

φ_V is unsatisfiable iff subformulas are unsatisfiable

$$\varphi_{goal} := \varphi_G \wedge \varphi_S = \bigwedge_{v \in G} v \wedge \bigvee_{s \in S} \varphi_s$$

However: φ_V is not in CNF $\rightarrow$ We cannot use SAT-solver on φ_V

Split up the Formula

Generally: $\varphi \wedge \bigvee_i \psi_i$ unsatisfiable iff $\varphi \wedge \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G} \wedge \underbrace{\bigvee_{s \in S} \varphi_s}$ unsatisfiable iff $\underbrace{\varphi_G} \wedge \underbrace{\varphi_s}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G} \land \underbrace{\bigvee_{s \in S}} \varphi_s$ unsatisfiable iff $\underbrace{\varphi_G} \land \underbrace{\varphi_s}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\mathbf{CL}} \land \underbrace{\bigvee_{s \in S} \varphi_s}$ unsatisfiable iff $\underbrace{\varphi_G}_{\mathbf{CL}} \land \underbrace{\varphi_s}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\varphi_s}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\varphi_s}_{\textbf{CL}}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \wedge \bigvee_i \psi_i$ unsatisfiable iff $\varphi \wedge \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \wedge \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \wedge \underbrace{\varphi_s}_{\textbf{CL}}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \wedge \bigvee_i \psi_i$ unsatisfiable iff $\varphi \wedge \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \wedge \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \wedge \underbrace{\varphi_s}_{\textbf{CL}}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- φ_{init}
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\varphi_s}_{\textbf{CL}}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- $\varphi_{init} \rightarrow 1$
- φ_{goal}
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\varphi_s}_{\textbf{CL}}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- $\varphi_{init} \rightarrow 1$
- $\varphi_{goal} \rightarrow$ #expanded states
- $\varphi_{inductive}$

Split up the Formula

Generally: $\varphi \land \bigvee_i \psi_i$ unsatisfiable iff $\varphi \land \psi_i$ unsatisfiable $\forall i$

In our case: $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\bigvee_{s \in S} \varphi_s}_{DNF}$ unsatisfiable iff $\underbrace{\varphi_G}_{\textbf{CL}} \land \underbrace{\varphi_s}_{\textbf{CL}}$ unsatisfiable $\forall s \in S$

Trivial SAT-calls, but many:

- $\varphi_{init} \to 1$
- $\varphi_{goal} \to$ #expanded states
- $\varphi_{inductive} \to$ #expanded states $\times$ #actions

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
- ... an **equisatisfiable** CNF formula

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
- ... an **equisatisfiable** CNF formula

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
- ... an **equisatisfiable** CNF formula

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
  ✗ impractical because of exponential blow-up
- ... an **equisatisfiable** CNF formula

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
  ✗ impractical because of exponential blow-up
- ... an **equisatisfiable** CNF formula

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
  ✗ impractical because of exponential blow-up
- ... an **equisatisfiable** CNF formula
  $\rightarrow$ relaxes equivalence, but preserves satisfiability

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
  ✗ impractical because of exponential blow-up
- ... an **equisatisfiable** CNF formula
  $\rightarrow$ relaxes equivalence, but preserves satisfiability
  ✓ increases formula size only linearly

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Transform the Formula

Idea: Transform φ_V into ...

- ... an **equivalent** CNF formula
  ✗ impractical because of exponential blow-up
- ... an **equisatisfiable** CNF formula
  $\rightarrow$ relaxes equivalence, but preserves satisfiability
  ✓ increases formula size only linearly

$$\varphi_V := \varphi_{init} \vee \varphi_{goal} \vee \varphi_{inductive}$$

Use the transformed formula as input to SAT-solver

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \vee w)$

- $(\neg x \vee v \vee w) \wedge (x \vee \neg v) \wedge (x \vee \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \vee \bigvee_i v_i) \wedge (\bigwedge_i (x \vee \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \lor w)$

- $(\neg x \lor v \lor w) \land (x \lor \neg v) \land (x \lor \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \lor \bigvee_i v_i) \land (\bigwedge_i (x \lor \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \lor w)$

- $(\neg x \lor v \lor w) \land (x \lor \neg v) \land (x \lor \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \lor \bigvee_i v_i) \land (\bigwedge_i (x \lor \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \vee w)$

- $(\neg x \vee v \vee w) \wedge (x \vee \neg v) \wedge (x \vee \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \vee \bigvee_i v_i) \wedge (\bigwedge_i (x \vee \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \vee w)$

- $(\neg x \vee v \vee w) \wedge (x \vee \neg v) \wedge (x \vee \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \vee \bigvee_i v_i) \wedge (\bigwedge_i (x \vee \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \vee w)$

- $(\neg x \vee v \vee w) \wedge (x \vee \neg v) \wedge (x \vee \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \vee \bigvee_i v_i) \wedge (\bigwedge_i (x \vee \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \vee w)$

- $(\neg x \vee v \vee w) \wedge (x \vee \neg v) \wedge (x \vee \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $(\neg x \vee \bigvee_i v_i) \wedge (\bigwedge_i (x \vee \neg v_i))$
  $\rightarrow$ can substitute larger subformula at once

Tseitin Encoding

Idea: Substitute all variable pairs with auxiliary variable

simple Tseitin Encoding: e.g. $x \leftrightarrow (v \vee w)$

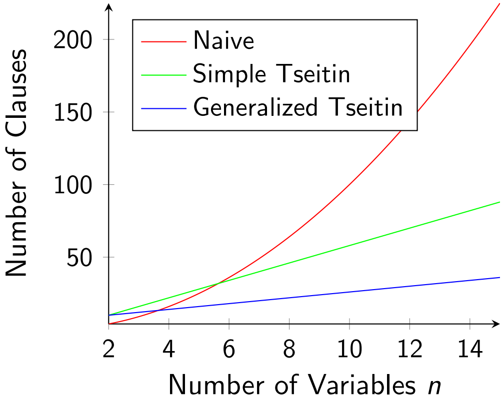
- $(\neg x \vee v \vee w) \wedge (x \vee \neg v) \wedge (x \vee \neg w)$
  $\rightarrow$ substitutes each variable pair

$\rightarrow$ equisatisfiable CNF

generalized Tseitin Encoding: e.g. $x \leftrightarrow (\bigvee_i v_i)$

- $\left(\neg x \vee \bigvee_i v_i\right) \wedge \left(\bigwedge_i (x \vee \neg v_i)\right)$
  $\rightarrow$ can substitute larger subformula at once

Transformation Comparison



Transformation of

$$\left(\bigwedge_{i=1}^{n} v_i\right) \vee \left(\bigwedge_{i=1}^{n} w_i\right)$$

into an equisatisfiable CNF

Experiments

Split up

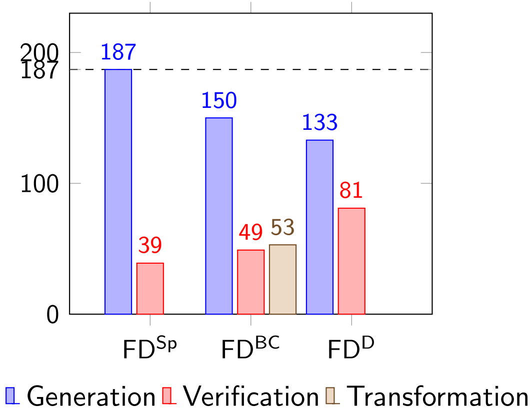
- $\mathrm{FD}^{Sp} \xrightarrow{\text{Task } \& \ \varphi_S} \mathrm{Ver}^{Sp}$
  incremental SAT-solver

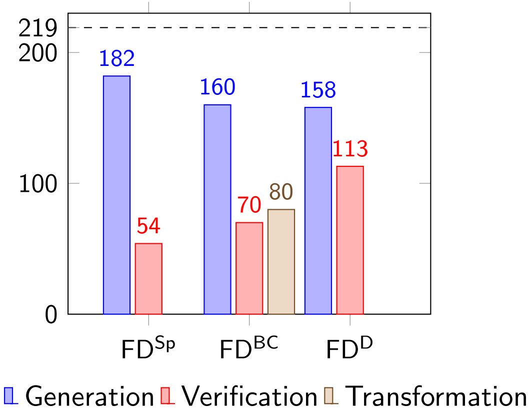
Transform

- $\mathrm{FD}^{BC} \xrightarrow{\text{Circuit}} \mathrm{Trans}^{BC} \xrightarrow{\text{CNF}} \mathrm{Ver}^{BC}$
  bc2cnf
- $\mathrm{FD}^{D} \xrightarrow{\text{CNF}} \mathrm{Ver}^{D}$
  direct transformation

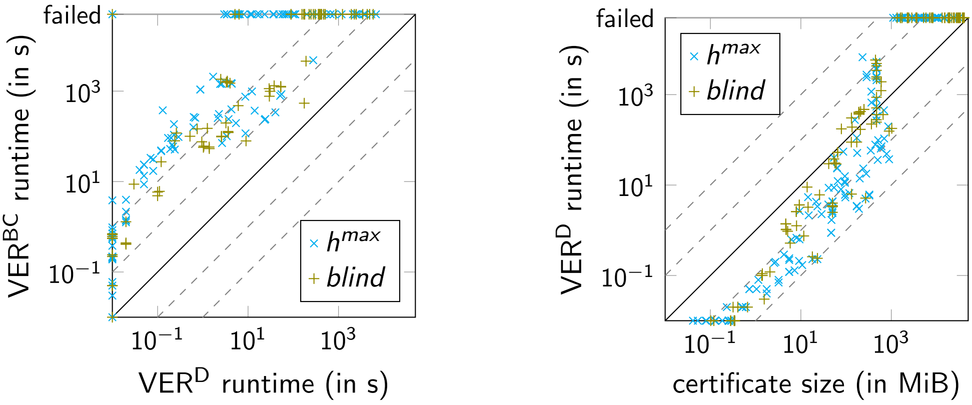
Comparison of *blind* and h^{max}

CNF Coverage: blind vs. h^{max}





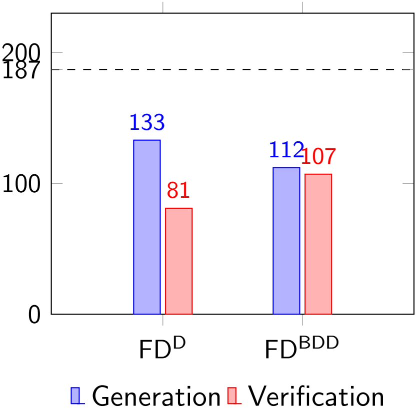
Time Comparison

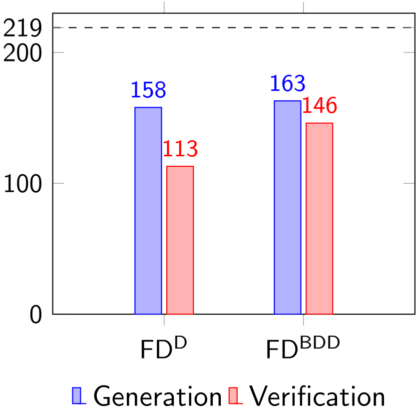


(a) verification runtime

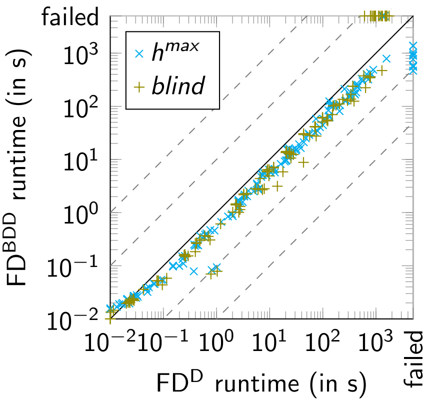
(b) verification efficiency

Coverage: blind vs. h^{max}

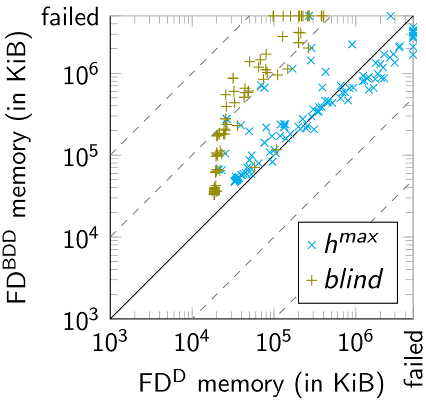




Generation CNF vs. BDD

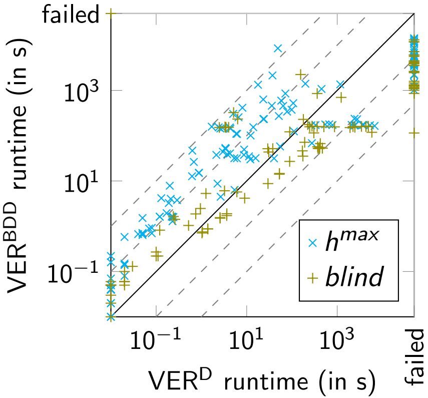


(a) time

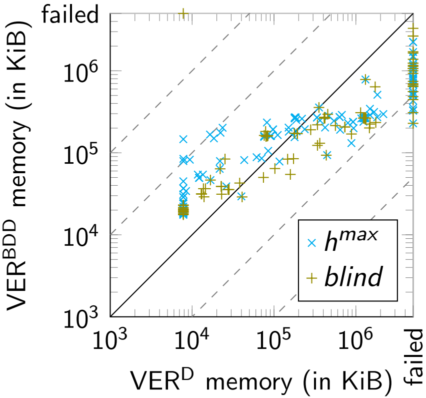


(b) memory

Verification CNF vs. BDD



(a) time



(b) memory

Conclusion

- Inductive Certificates capture unsolvability
- Splitting the SAT-calls avoids inefficiency of SAT
- Tseitin Encoding allows equisatisfiable transformation to CNF
- CNF representation of certificates is practically viable
- Exponential scaling of SAT

Questions?

fabian.kruse@unibas.ch

Failures

	blind		h^{max}	
	memory	time	memory	time
FD^{Sp}	0	0	5	32
FD^{BC}	0	37	25	34
$Trans^{BC}$	96	1	77	3
FD^{D}	1	53	21	40

Table: Reason for failures during generation in tasks where FD generated a certificate

Failures

	blind		h^{max}	
	memory	time	memory	time
$\mathrm{VER}^{\mathrm{Sp}}$	144	4	121	7
$\mathrm{VER}^{\mathrm{BC}}$	4	0	10	0
$\mathrm{VER}^{\mathrm{D}}$	50	0	45	0

Table: Reason for failures during verification in tasks where a certificate was generated

Failures

	blind		h^{max}	
	memory	time	memory	time
FD^{D}	1	53	21	40
FD^{BDD}	54	0	21	35
VER^{D}	50	0	45	0
VER^{BDD}	0	5	0	17

Table: Reasons for failure in tasks that FD solved