Decision Procedures for Verification

Combinations of decision procedures (2)

27.01.2015

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Until now:

Decidable subclasses of FOL

Decision procedures for single theories

Uninterpreted function symbols

Decision procedures for numeric domains

Combinations of theories

The Nelson-Oppen combination procedure.

Combination of theories over disjoint signatures

The Nelson/Oppen procedure

Given: $\mathcal{T}_1, \mathcal{T}_2$ first-order theories with signatures Σ_1, Σ_2

Assume that $\Sigma_1 \cap \Sigma_2 = \emptyset$ (share only $\approx$)

P_i decision procedures for satisfiability of ground formulae w.r.t. $\mathcal{T}_i$

ϕ quantifier-free formula over $\Sigma_1 \cup \Sigma_2$

Task: Check whether ϕ is satisfiable w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_2$

Note: Restrict to conjunctive quantifier-free formulae

$\phi \mapsto DNF(\phi)$

$DNF(\phi)$ satisfiable in $\mathcal{T}$ iff one of the disjuncts satisfiable in $\mathcal{T}$

The Nelson-Oppen algorithm

ϕ conjunction of literals

Step 1. Purification $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \phi \mapsto (\mathcal{T}_1 \cup \phi_1) \cup (\mathcal{T}_2 \cup \phi_2)$:

where ϕ_i is a pure Σ_i-formula and $\phi_1 \wedge \phi_2$ is equisatisfiable with ϕ.

Step 2. Propagation.

The decision procedure for ground satisfiability for $\mathcal{T}_1$ and $\mathcal{T}_2$ fairly exchange information concerning entailed unsatisfiability of constraints in the shared signature i.e. clauses over the shared variables.

until an inconsistency is detected or a saturation state is reached.

The Nelson-Oppen algorithm

ϕ conjunction of literals

Step 1. Purification $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \phi \mapsto (\mathcal{T}_1 \cup \phi_1) \cup (\mathcal{T}_2 \cup \phi_2)$:

where ϕ_i is a pure Σ_i-formula and $\phi_1 \wedge \phi_2$ is equisatisfiable with ϕ.

> not problematic; requires linear time

Step 2. Propagation.

The decision procedure for ground satisfiability for $\mathcal{T}_1$ and $\mathcal{T}_2$ fairly exchange information concerning entailed unsatisfiability of constraints in the shared signature

i.e. clauses over the shared variables.

until an inconsistency is detected or a saturation state is reached.

> not problematic; termination guaranteed
> Sound: if inconsistency detected input unsatisfiable
> Complete: under additional assumptions

Implementation

ϕ conjunction of literals

Step 1. Purification: $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \phi \mapsto (\mathcal{T}_1 \cup \phi_1) \cup (\mathcal{T}_2 \cup \phi_2)$,
where ϕ_i is a pure Σ_i-formula and $\phi_1 \wedge \phi_2$ is equisatisfiable with ϕ.

Step 2. Propagation: The decision procedure for ground satisfiability for $\mathcal{T}_1$ and $\mathcal{T}_2$ fairly exchange information concerning entailed unsatisfiability of constraints in the shared signature
i.e. clauses over the shared variables.

until an inconsistency is detected or a saturation state is reached.

How to implement Propagation?

Guessing: guess a maximal set of literals containing the shared variables; check it for $\mathcal{T}_i \cup \phi_i$ consistency.

Backtracking: identify disjunction of equalities between shared variables entailed by $\mathcal{T}_i \cup \phi_i$; make case split by adding some of these equalities to ϕ_1, ϕ_2. Repeat as long as possible.

Implementation of propagation

Guessing variant

Guess a maximal set of literals containing the shared variables V (arrangement: $\alpha(V, E) = (\bigwedge_{(u,v)\in E} u \approx v \wedge \bigwedge_{(u,v)\notin E} u \not\approx v)$, where E equivalence relation); check it for $\mathcal{T}_i \cup \phi_i$ consistency.

On the blackboard: Example 10.5 and 10.7 pages 272, 273
Example 10.6 and 10.9 pages 272, 275

from the book "The Calculus of Computation" by A. Bradley and Z. Manna

Advantage: Whenever constraints are represented as Boolean combinations of atoms, one may combine heuristics of SMT solvers with specific features of the theories to be combined to produce the right arrangement efficiently.

Implementation of propagation

Backtracking variant

Identify disjunction of equalities between shared variables entailed by $\mathcal{T}_i \cup \phi_i$; make case split by adding some of these equalities to ϕ_1, ϕ_2.

Repeat as long as possible.

On the blackboard: Example 10.14, page 280-281, and Example 10.13, page 279, from the book "The Calculus of Computation" by A. Bradley and Z. Manna

Advantages:

- it works on the non-disjoint case as well
- can be made deterministic for combinations of convex theories

$\mathcal{T}$ convex iff whenever $\mathcal{T} \models \bigwedge_{i=1}^{n} A_i \rightarrow \bigvee_{j=1}^{m} B_j$
there exists j s.t. $\mathcal{T} \models \bigwedge_{i=1}^{n} A_i \rightarrow B_j$

The Nelson-Oppen algorithm

Termination: only finitely many shared variables to be identified

The Nelson-Oppen algorithm

Termination: only finitely many shared variables to be identified

Soundness: If procedure answers "unsatisfiable" then ϕ is unsatisfiable

Proof: Assume that ϕ is satisfiable. Then $\phi_1 \wedge \phi_2$ satisfiable.

- The procedure cannot answer "unsatisfiable" in Step 2.
- Let $(\mathcal{M}, \beta) \models \phi_1 \wedge \phi_2$. Assume that $(\mathcal{M}, \beta) \models \bigwedge_{(c_i,c_j)\in E} c_i \approx c_j \wedge \bigwedge_{(c_i,c_j)\notin E} c_i \not\approx c_j$

Then
$$(\mathcal{M}_{|\Sigma_1}, \beta) \models \phi_1 \wedge \bigwedge_{(c_i,c_j)\in E} c_i \approx c_j$$
$$(\mathcal{M}_{|\Sigma_2}, \beta) \models \phi_2 \wedge \bigwedge_{(c_i,c_j)\in E} c_i \approx c_j$$

Guessing: $\bigwedge_{(c_i,c_j)\in E} c_i \approx c_j \wedge \bigwedge_{(c_i,c_j)\notin E} c_i \not\approx c_j$ "satisfiable arrangement".

Backtracking: Procedure answers satisfiable on the corresponding branch.

The Nelson-Oppen algorithm

Termination: only finitely many shared variables to be identified

Soundness: If procedure answers "unsatisfiable" then ϕ is unsatisfiable

Completeness: Under additional hypotheses

Completeness

Example:

E_1	E_2
$f(g(x), g(y)) \approx x$	$k(x) \approx k(x)$
$f(g(x), h(y)) \approx y$	
non-trivial	non-trivial

$g(c) \approx h(c) \wedge k(c) \not\approx c$

$g(c) \approx h(c)$	$k(c) \not\approx c$
satisfiable in E_1	satisfiable in E_2

no equations between shared variables; Nelson-Oppen answers "satisfiable"

Completeness

Example:

E_1	E_2
$f(g(x), g(y)) \approx x$	$k(x) \approx k(x)$
$f(g(x), h(y)) \approx y$	
non-trivial	non-trivial

$g(c) \approx h(c) \wedge k(c) \not\approx c$

$g(c) \approx h(c)$	$k(c) \not\approx c$
satisfiable in E_1	satisfiable in E_2

no equations between shared variables; Nelson-Oppen answers "satisfiable"

> A model of E_1 satisfies $g(c) \approx h(c)$ iff $\exists e \in A$ s.t. $g(e) = h(e)$.
>
> Then, for all $a \in A$: $a = f_A(g(a), g(e)) = f_A(g(a), h(e)) = e$

$g(c) \approx h(c) \wedge k(c) \not\approx c$ unsatisfiable

Completeness

Another example

$\mathcal{T}_1$ theory admitting models of cardinality at most 2

$\mathcal{T}_2$ theory admitting models of any cardinality

$$f_1 \in \Sigma_1, f_2 \in \Sigma_2 \quad \text{such that} \quad \mathcal{T}_i \not\models \forall x, y \quad f_i(x) = f_i(y).$$

$$\phi \;=\; f_1(c_1) \not\approx f_1(c_2) \quad \wedge \quad f_2(c_1) \not\approx f_2(c_3) \quad \wedge \quad f_2(c_2) \not\approx f_2(c_3)$$

$$\phi_1 \;=\; f_1(c_1) \not\approx f_1(c_2) \qquad \phi_2 \;=\; f_2(c_1) \not\approx f_2(c_3) \;\wedge\; f_2(c_2) \not\approx f_2(c_3)$$

The Nelson-Oppen procedure returns "satisfiable"

$$\mathcal{T}_1 \cup \mathcal{T}_2 \models \forall x, y, z (f_1(x) \not\approx f_1(y) \;\wedge\; f_2(x) \not\approx f_2(z) \;\wedge\; f_2(y) \not\approx f_2(z) \\ \rightarrow (x \not\approx y \wedge x \not\approx z \wedge y \not\approx z))$$

$f_1(c_1) \not\approx f_1(c_2) \quad \wedge \quad f_2(c_1) \not\approx f_2(c_3) \quad \wedge \quad f_2(c_2) \not\approx f_2(c_3)$ unsatisfiable

Completeness

Cause of incompleteness

There exist formulae satisfiable in finite models of bounded cardinality

Solution: Consider stably infinite theories.

$\mathcal{T}$ is stably infinite iff for every quantifier-free formula ϕ

ϕ satisfiable in $\mathcal{T}$ iff ϕ satisfiable in an infinite model of $\mathcal{T}$.

Note: This restriction is not mentioned in [Nelson Oppen 1979];

introduced by Oppen in 1980.

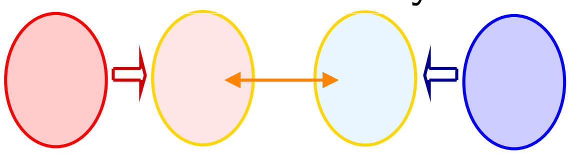
Completeness

Guessing version: C set of constants shared by ϕ_1, ϕ_2

R equiv. relation assoc. with partition of $C \mapsto ar(C, R) = \bigwedge_{R(c,d)} c \approx d \wedge \bigwedge_{\neg R(c,d)} c \not\approx d$

Lemma. Assume that there exists a partition of C s.t. $\phi_i \wedge ar(C, R)$ is $\mathcal{T}_i$-satisfiable. Then $\phi_1 \wedge \phi_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable.

Idea of proof: Let $\mathcal{A}_i \in \text{Mod}(\mathcal{T}_i)$ s.t. $\mathcal{A}_i \models \phi_i \wedge ar(C, R)$. Then $c_{A_1} = d_{A_1}$ iff $c_{A_2} = d_{A_2}$.
Let $i : \{c_{A_1} \mid c \in C\} \to \{c_{A_2} \mid c \in C\}$, $i(c_{A_1}) = c_{A_2}$ well-defined; bijection.
Stable infinity: can assume w.l.o.g. that $\mathcal{A}_1, \mathcal{A}_2$ have the same cardinality
Let $h : \mathcal{A}_1 \to \mathcal{A}_2$ bijection s.t. $h(c_{A_1}) = c_{A_2}$



Use h to transfer the Σ_1-structure on $\mathcal{A}_2$.

Theorem. If $\mathcal{T}_1, \mathcal{T}_2$ are both stably infinite and the shared signature is empty then the Nelson-Oppen procedure is sound, complete and terminating. Thus, it transfers decidability of ground satisfiability from $\mathcal{T}_1, \mathcal{T}_2$ to $\mathcal{T}_1 \cup \mathcal{T}_2$.

Complexity

Main sources of complexity:

(i) transformation of the formula in DNF

(ii) propagation

(a) decide whether there is a disjunction of equalities between variables

(b) investigate different branches corresponding to disjunctions

Complexity

Main sources of complexity:

(i) transformation of the formula in DNF

(ii) propagation

$\mathcal{T}$ is convex	iff	for every quantifier-free formula ϕ,
		$\phi \models \bigvee_i x_i \approx y_i$ implies $\phi \models x_j \approx y_j$ for some j.

$\mapsto$ No branching

Complexity

Main sources of complexity:

(i) transformation of the formula in DNF

(ii) propagation

$\mathcal{T}$ is convex iff for every quantifier-free formula ϕ,
$\phi \models \bigvee_i x_i \approx y_i$ implies $\phi \models x_j \approx y_j$ for some j.

$\mapsto$ No branching

Theorem. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be convex and stably infinite; $\Sigma_1 \cap \Sigma_2 = \emptyset$
If satisfiability of conjunctions of literals in $\mathcal{T}_i$ is in PTIME
Then satisfiability of conjunctions of literals in $\mathcal{T}_1 \cup \mathcal{T}_2$ is in PTIME

Complexity

In general: non-deterministic procedure

Theorem.	Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be convex and stably infinite; $\Sigma_1 \cap \Sigma_2 = \emptyset$
	If satisfiability of conjunctions of literals in $\mathcal{T}_i$ is in NP
	Then satisfiability of conjunctions of literals in $\mathcal{T}_1 \cup \mathcal{T}_2$ is in NP

Extensions of the Nelson-Oppen procedure

- relax the stable infiniteness requirement

- relax the requirement that the theories have disjoint signatures

Extensions of the Nelson-Oppen procedure

- relax the stable infiniteness requirement

  [Tinelli,Zarba'03] One theory "shiny" (for each satisf. constraint we can compute a finite k s.t. the theory has models of every cardinality $\lambda \geq k$)

- relax the requirement that the theories have disjoint signatures

  [Tinelli,Ringeissen'03] Theories sharing absolutely free constructors

  [Ghilardi'04] Model theoretical conditions.

Main idea:

Find situations in which $\mathcal{T}_i$ models of $\phi_i, i = 1, 2$ can be "amalgamated" to a $\mathcal{T}_1 \cup \mathcal{T}_2$ model of $\phi_1 \wedge \phi_2$.

From conjunctions to arbitrary combinations

Until now:

check satisfiability for conjunctions of literals

Question:

how to check satisfiability of sets of clauses?

Overview

- Propositional logic
  - resolution
  - DPLL
- First-order logic
  - resolution

Satisfiability w.r.t. theories

- Ground formulae
  - conjunctions of literals: specialized methods
  - clauses: DPLL(T) $\Leftarrow$ TODAY
- Formulae with quantifiers
  - reduction to SAT for ground formulae instantiation $\Leftarrow$ NEXT WEEK (situations when sound and complete)
  - resolution (mod T)

3.6 The $DPLL(\mathcal{T})$ algorithm

Reminder: Propositional SAT

The DPLL algorithm

A succinct formulation

State: $M||F$,

where:

- M partial assignment (sequence of literals),

  some literals are annotated (L^d: decision literal)

- F clause set.

A succinct formulation

UnitPropagation

$M||F, C \vee L \Rightarrow M, L||F, C \vee L$ if $M \models \neg C$, and L undef. in M

Decide

$M||F \Rightarrow M, L^d||F$ if L or $\neg L$ occurs in F, L undef. in M

Fail

$M||F, C \Rightarrow \text{Fail}$ if $M \models \neg C$, M contains no decision literals

Backjump

$$M, L^d, N||F \Rightarrow M, L'||F \quad \text{if} \begin{cases} \text{there is some clause } C \vee L' \text{ s.t.:} \\ F \models C \vee L', M \models \neg C, \\ L' \text{ undefined in } M \\ L' \text{ or } \neg L' \text{ occurs in } F. \end{cases}$$

Example

Assignment:	Clause set:	
$\emptyset$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
P_1	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1 P_2$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1 P_2 P_3$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1 P_2 P_3 P_4$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1 P_2 P_3 P_4 P_5$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1 P_2 P_3 P_4 P_5 \neg P_6$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Backtrack)
$P_1 P_2 P_3 P_4 \neg P_5$	$\|\|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	...

DPLL with learning

The DPLL system with learning consists of the four transition rules of the Basic DPLL system, plus the following two additional rules:

Learn

$M||F \Rightarrow M||F, C$ if all atoms of C occur in F and $F \models C$

Forget

$M||F, C \Rightarrow M||F$ if $F \models C$

In these two rules, the clause C is said to be learned and forgotten, respectively.

SAT Modulo Theories (SMT)

Some problems are more naturally expressed in richer logics than just propositional logic, e.g:

- Software/Hardware verification needs reasoning about equality, arithmetic, data structures, ...

SMT consists of deciding the satisfiability of a ground 1st-order formula with respect to a background theory T

Example 1: $\mathcal{T}$ is Equality with Uninterpreted Functions (UIF):

$$f(g(a)) \not\approx f(c) \;\vee\; g(a) \approx d, \quad g(a) \approx c, \quad c \not\approx d$$

Example 2: for combined theories:

$$A \approx \mathrm{write}(B, a+1, 4), \qquad \mathrm{read}(A, b+3) \approx 2 \;\vee\; f(a-1) \not\approx f(b+1)$$

SAT Modulo Theories (SMT)

The "very eager" approach to SMT

Method:

– translate problem into equisatisfiable propositional formula;

– use off-the-shelf SAT solver

- Why "eager"?

  Search uses all theory information from the beginning

- Characteristics:

  + Can use best available SAT solver

  – Sophisticated encodings are needed for each theory

  – Sometimes translation and/or solving too slow

Main Challenge for alternative approaches is to combine:

- DPLL-based techniques for handling the boolean structure

- Efficient theory solvers for conjunctions of $\mathcal{T}$-literals

SAT Modulo Theories (SMT)

"Lazy" approaches to SMT: Idea

Example: consider $\mathcal{T} = UIF$ and the following set of clauses:

$$\underbrace{f(g(a)) \not\approx f(c)}_{\neg P_1} \vee \underbrace{g(a) \approx d}_{P_2}, \quad \underbrace{g(a) \approx c}_{P_3}, \quad \underbrace{c \not\approx d}_{\neg P_4}$$

1. Send $\{\neg P_1 \vee P_2,\ P_3,\ \neg P_4\}$ to SAT solver

   SAT solver returns model $[\neg P_1, P_3, \neg P_4]$
   Theory solver says $\neg P_1 \wedge P_3 \wedge \neg P_4$ is $\mathcal{T}$-inconsistent

2. Send $\{\neg P_1 \vee P_2,\ P_3,\ \neg P_4,\ P_1 \vee \neg P_3 \vee P_4\}$ to SAT solver

   SAT solver returns model $[P_1, P_2, P_3, \neg P_4]$
   Theory solver says $P_1 \wedge P_2 \wedge P_3 \wedge \neg P_4$ is $\mathcal{T}$-inconsistent

3. Send $\{\neg P_1 \vee P_2, P_3, \neg P_4, P_1 \vee \neg P_3 \vee P_4, \neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4\}$ to SAT solver

   SAT solver says UNSAT

SAT Modulo Theories (SMT)

Optimized lazy approach

LA	• Check T-consistency only of full propositional models
OLA	• Check T-consistency of partial assignment while being built
LA	• Given a T-inconsistent assignment M, add $\neg M$ as a clause
OLA	• Given a T-inconsistent assignment M, find an explanation (a small T-inconsistent subset of M) and add it as a clause
LA	• Upon a T-inconsistency, add clause and restart
OLA	• Upon a T-inconsistency, do conflict analysis of the explanation and Backjump

SAT Modulo Theories (SMT)

"Lazy" approaches to SMT

- Why "lazy"?

Theory information used only lazily, when checking $\mathcal{T}$-consistency of propositional models

- Characteristics:

  + Modular and flexible
  – Theory information does not guide the search
    (only validates a posteriori)

Tools: CVC-Lite, ICS, MathSAT, TSAT++, Verifun, ...

"Lazy" approaches to SMT

Lazy theory learning:

$$M, L, M_1 || F \Rightarrow \emptyset || F, \neg L_1 \vee \cdots \vee \neg L_n \vee \neg L \quad \text{if} \begin{cases} M, L, M_1 \models F \\ \{L_1, \ldots, L_n\} \subseteq M \\ L_1 \wedge \cdots \wedge L_n \wedge L \models_{\mathcal{T}} \bot \end{cases}$$

Lazy theory learning + no repetitions

$$M, L, M_1 || F \Rightarrow \emptyset || F, \neg L_1 \vee \cdots \vee \neg L_n \vee \neg L \quad \text{if} \begin{cases} \{L_1, \ldots, L_n\} \subseteq M \\ L_1 \wedge \cdots \wedge L_n \wedge L \models_{\mathcal{T}} \bot \\ \neg L_1 \vee \cdots \vee \neg L_n \vee \neg L \notin F \end{cases}$$

DPLL(T) Rules

UnitPropagation

$M || F, C \vee L \Rightarrow M, L || F, C \vee L$ if $M \models \neg C$, and L undef. in M

Decide

$M || F \Rightarrow M, L^d || F$ if L occurs in F, L undef. in M

Fail

$M || F, C \Rightarrow \text{Fail}$ if $M \models \neg C$, no backtrack possible

Backjump

$M, L^d, N || F \Rightarrow M, L' || F$ if $\begin{cases} \text{there is some clause } C \vee L' \text{ s.t.:} \\ F \models C \vee L', M \models \neg C, \\ L' \text{ undefined in } M \\ L' \text{ or } \neg L' \text{ occurs in } F. \end{cases}$

Restart/Learn

$M || F \Rightarrow \emptyset || F, F'$ if $F \models F'$, F' obtained from M, F

TPropagation

$M || F \Rightarrow M, L || F$ if $M \models_{\mathcal{T}} L$

DPLL(T) Example

Consider again same example with UIF:

$$\underbrace{f(g(a)) \not\approx f(c)}_{\neg P_1} \vee \underbrace{g(a) \approx d}_{P_2}, \quad \underbrace{g(a) \approx c}_{P_3}, \quad \underbrace{c \not\approx d}_{\neg P_4}$$

$$\begin{array}{lll}
\emptyset & || \neg P_1 \vee P_2, P_3, \neg P_4 & \Rightarrow (\mathit{UnitPropagation}) \\
P_3 & || \neg P_1 \vee P_2, P_3, \neg P_4 & \Rightarrow (\mathit{TPropagation}) \\
P_3 P_1 & || \neg P_1 \vee P_2, P_3, \neg P_4 & \Rightarrow (\mathit{UnitPropagation}) \\
P_3 P_1 P_2 & || \neg P_1 \vee P_2, P_3, \neg P_4 & \Rightarrow (\mathit{TPropagation}) \\
P_3 P_1 P_2 P_4 & || \neg P_1 \vee P_2, P_3, \neg P_4 & \Rightarrow \mathit{fail}
\end{array}$$

No search in this example