Decision Procedures in Verification

Combinations of decision procedures (3)

28.01.2013

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Until now:

Logical Theories: generalities

- Theory of Uninterpreted Function Symbols
- Decision procedures for numeric domains

  Difference logic

  Linear arithmetic: Fourier-Motzkin
- Combinations of decision procedures

  Definitions

  The Nelson/Oppen Procedure

  DPLL(T) (idea)

Motivation

Question:

how to check satisfiability (modulo theories) for sets of clauses?

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:

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

Lazy theory learning + no repetitions

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

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

Termination

Idea: $DPLL(T)$ terminates if no clause is learned infinitely many times, since only finitely many such new clauses (built over input literals) exist.

Theorem. There exists no infinite sequence of the form

$$\emptyset || F \Rightarrow S_1 \Rightarrow S_2 ...$$

if no clause C is learned by Reset & Learn/Lazy Theory Learning infinitely many times along a sequence.

A similar termination result holds also for the DPLL(T) approach with Theory Propagation.

Termination

Theorem. There exist no infinite sequences of the form $\emptyset || F \Rightarrow S_1 \Rightarrow S_2 \ldots$

Proof. (Idea) We define a well-founded strict partial ordering $\succ$ on states, and show that each rule application $M||F \Rightarrow M'||F'$ is decreasing with respect to this ordering, i.e., $M||F \succ M'||F'$.

Let M be of the form $M_0, L_1, M_1, \ldots L_p, M_p$, where $L_1, \ldots, L_p$ are all the decision literals of M. Similarly, let M' be $M'_0, L'_1, M'_1, \ldots L'_{p'}, M'_{p'}$.

Let N be the number of distinct atoms (propositional variables) in F.

(Note that p, p' and the length of M and M' are always smaller than or equal to N.)

Termination

Theorem. There exist no infinite sequences of the form $\emptyset || F \Rightarrow S1 \Rightarrow \ldots$

Proof. (continued)
Let $m(M)$ be $N - \text{length}(M)$ (nr. of literals missing in M for M to be total).

Define: $M_0 L_1 M_1 \ldots L_p M_p || F \succ M'_0 L'_1 M'_1 \ldots L'_{p'} M'_{p'} || F'$ if

(i) there is some i with $0 \leq i \leq p, p'$ such that

$m(M_0) = m(M'_0), \ldots m(M_{i-1}) = m(M'_{i-1}), m(M_i) > m(M'_i)$ or

(ii) $m(M_0) = m(M'_0), \ldots m(M_p) = m(M'_p)$ and $m(M) > m(M')$.

Comparing the number of missing literals in sequences is a strict ordering (irreflexive and transitive) and it is well-founded, and hence this also holds for its lexicographic extension on tuples of sequences of bounded length.

No learning/forgetting: It is easy to see that all Basic DPLL rule applications are decreasing with respect to $\succ$ if fail is added as an additional minimal element. (The rules UnitPropagate and Backjump decrease by case (i) of the definition and Decide decreases by case (ii).)

Termination

Theorem. There exist no infinite sequences of the form $\emptyset || F \Rightarrow S1 \Rightarrow \ldots$

Note: Combine learning with basic DPLL(T): no clause learned infinitely many times.

Forget: For this termination condition to be fulfilled, applying at least one rule of the Basic DPLL system between any two Learn applications does not suffice. It suffices if, in addition, no clause generated with Learning is ever forgotten.

Soundness, Correctness, Termination

Lemma. If $\emptyset || F \Rightarrow^* M || F'$ then:

(1) All atoms in M and all atoms in F' are atoms of F.

(2) M: no literal more than once, no complementary literals

(3) F' is logically equivalent to F

(4) if $M = M_0 L_1 M_1 \ldots L_n M_n$ where L_i all decision literals
then $F, L_1, \ldots, L_i \models M_i$.

Lemma. If $\emptyset || F \Rightarrow^* M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Lazy Theory Learning, then:

(1) All literals of F' are defined in M

(2) There is no clause C in F' such that $M \models \neg C$

(3) M is a model of F.

Soundness, Correctness, Termination

Lemma. If $\emptyset || F \Rightarrow^* M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Lazy Theory Learning, then M is a $\mathcal{T}$-model of F.

Theorem. The Lazy Theory learning DPLL system provides a decision procedure for the satisfiability in $\mathcal{T}$ of CNF formulae F, that is:

1. $\emptyset || F \Rightarrow^* \mathit{fail}$ if, and only if, F is unsatisfiable in $\mathcal{T}$.
2. $\emptyset || F \Rightarrow^* M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Lazy Theory Learning, if, and only if, F is satisfiable in $\mathcal{T}$.

Proof

(1) If $\emptyset || F \Rightarrow^* \mathit{fail}$ then there exists state $M || F'$ with $\emptyset || F \Rightarrow^* M || F' \Rightarrow \mathit{fail}$, there is no decision literal in M and $M \models \neg C$ for some clause C in F. By the construction of M, $F \models M$, so $F \models \neg C$. Thus F is unsatisfiable.

To prove the converse, if $\emptyset || F \not\Rightarrow^* \mathit{fail}$ then by there must be a state $M || F'$ such that $\emptyset || F \Rightarrow^* M || F'$. Then $M \models F$, so F is satisfiable.

Soundness, Correctness, Termination

Lemma. If $\emptyset || F \Rightarrow^* M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Lazy Theory Learning, then M is a $\mathcal{T}$-model of F.

Theorem. The Lazy Theory learning DPLL system provides a decision procedure for the satisfiability in $\mathcal{T}$ of CNF formulae F, that is:

1. $\emptyset || F \Rightarrow^*$ *fail* if, and only if, F is unsatisfiable in $\mathcal{T}$.
2. $\emptyset || F \Rightarrow^* M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Lazy Theory Learning, if, and only if, F is satisfiable in $\mathcal{T}$.

Proof

2. If $\emptyset || F \Rightarrow^* M || F$ then F is satisfiable. Conversely, if $\emptyset || F \not\Rightarrow^* M || F$ then $\emptyset || F \Rightarrow^*$ *fail*, so F is unsatisfiable.

Termination, Soundness and Completeness

DPLL($\mathcal{T}$) with (eager) theory propagation

Lemma. If $\emptyset || F \Rightarrow M || F$ then M is $\mathcal{T}$-consistent.

Proof. This property is true initially, and all rules preserve it, by the fact that $M \models_{\mathcal{T}} L$ if, and only if, $M \cup \neg L$ is $\mathcal{T}$-inconsistent: the rules only add literals to M that are undefined in M, and Theory Propagate adds all literals L of F that are theory consequences of M, before any literal $\neg L$ making it $\mathcal{T}$-inconsistent can be added to M by any of the other rules.

Termination, Soundness and Completeness

DPLL($\mathcal{T}$) with (eager) theory propagation

Definition. A DPLL($\mathcal{T}$) procedure with Eager Theory Propagation for $\mathcal{T}$ is any procedure taking an input CNF F and computing a sequence $\emptyset || F \Rightarrow^* S$ where S is a final state wrt. Theory Propagate and the Basic DPLL system.

Theorem The DPLL system with eager theory propagation provides a decision procedure for the satisfiability in $\mathcal{T}$ of CNF formulae F, that is:

1. $\emptyset || F \Rightarrow^* \mathit{fail}$ if, and only if, F is unsatisfiable in $\mathcal{T}$.
2. $\emptyset || F \Rightarrow^* M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Theory Propagate, if, and only if, F is satisfiable in $\mathcal{T}$.
3. If $\emptyset || F \Rightarrow M || F'$, where $M || F'$ is a final state wrt the Basic DPLL system and Theory Propagate, then M is a $\mathcal{T}$-model of F.

Literature

Full proofs and further details can be found in:

Robert Nieuwenhuis, Albert Oliveras and Cesare Tinelli:
"Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)"

Journal of the ACM, Vol. 53, No. 6, November 2006, pp.937-977.

SMT tools

SAT problems

Given: conjunction ϕ of prop. clauses
Task: check if ϕ satisfiable

Method: DPLL

- deterministic choices first
  - unit resolution
  - pure literal assignment
- case distinction (splitting)
- heuristics
  - selection criteria for splitting
  - backtracking
  - conflict-driven learning

SMT tools

SAT problems

Given: conjunction ϕ of prop. clauses
Task: check if ϕ satisfiable

Method: DPLL

- deterministic choices first
  - unit resolution
  - pure literal assignment
- case distinction (splitting)
- heuristics
  - selection criteria for splitting
  - backtracking
  - conflict-driven learning

SMT problems

Given: conjunction ϕ of clauses
Task: check if $\phi \models_{\mathcal{T}} \perp$

Method: DPLL($\mathcal{T}$)

- Boolean assignment found
  using DPLL
- ... and checked for $\mathcal{T}$-satisfiability
- the assignment can be partial
  and checked before splitting
- usual heuristics are used:
  - non-chronological backtracking
  - learning

SMT tools

SAT problems

Given: conjunction ϕ of prop. clauses
Task: check if ϕ satisfiable

Method: DPLL
- deterministic choices first
  unit resolution
  pure literal assignment
- case distinction (splitting)
- heuristics
  selection criteria for splitting
  backtracking
  conflict-driven learning

SMT problems

Given: conjunction ϕ of clauses
Task: check if $\phi \models_{\mathcal{T}} \perp$

Method: DPLL($\mathcal{T}$)
- Boolean assignment found
  using DPLL
- ... and checked for $\mathcal{T}$-satisfiability
- the assignment can be partial
  and checked before splitting
- usual heuristics are used:
  non-chronological backtracking
  learning

Systems implementing such specialized satisfiability problems: Yices, Barcelogic Tools, CVC lite, haRVey, Math-SAT,... are called (S)atisfiability (M)odulo (T)heory solvers.

Satisfiability of formulae with quantifiers

In many applications we are interested in testing the satisfiability of formulae containing (universally quantified) variables.

Examples

- check satisfiability of formulae in the Bernays-Schönfinkel class
- check whether a set of (universally quantified) Horn clauses entails a ground clause
- check whether a property is consequence of a set of axioms

  Example 1: $f : \mathbb{Z} \to \mathbb{Z}$ is monotonely increasing
  and $g : \mathbb{Z} \to \mathbb{Z}$ is defined by $g(x) = f(x + x)$
  then g is also monotonely increasing

  Example 2: If array a is increasingly sorted, and
  x is inserted before the first position i with $a[i] > x$
  then the array remains increasingly sorted.

A theory of arrays

We consider the theory of arrays in a many-sorted setting.

Syntax:

- Sorts: Elem (elements), Array (arrays) and Index (indices, here integers).
- Function symbols: read, write.

  $a(\text{read}) = \mathit{Array} \times \mathit{Index} \to \mathit{Element}$

  $a(\text{write}) = \mathit{Array} \times \mathit{Index} \times \mathit{Element} \to \mathit{Array}$

Theories of arrays

We consider the theory of arrays in a many-sorted setting.

Theory of arrays $\mathcal{T}_{arrays}$:

- $\mathcal{T}_i$ (theory of indices): Presburger arithmetic
- $\mathcal{T}_e$ (theory of elements): arbitrary
- Axioms for read, write

$$\begin{array}{rcl} read(write(a, i, e), i) & \approx & e \\ j \not\approx i \vee read(write(a, i, e), j) & = & read(a, j). \end{array}$$

Theories of arrays

We consider the theory of arrays in a many-sorted setting.

Theory of arrays $\mathcal{T}_{arrays}$:

- $\mathcal{T}_i$ (theory of indices): Presburger arithmetic
- $\mathcal{T}_e$ (theory of elements): arbitrary
- Axioms for read, write

$$\begin{aligned} read(write(a, i, e), i) &\approx e \\ j \not\approx i \vee read(write(a, i, e), j) &= read(a, j). \end{aligned}$$

Fact: Undecidable in general.

Goal: Identify a fragment of the theory of arrays which is decidable.

A decidable fragment

- **Index guard** a positive Boolean combination of atoms of the form $t \leq u$ or $t = u$ where t and u are either a variable or a ground term of sort Index

  Example: $(x \leq 3 \vee x \approx y) \wedge y \leq z$ is an index guard
  $x \leq c - 1$ (where c is a constant) is an index guard

  Example: $x + 1 \leq y$, $x + 3 \leq y - 1$, $x + x \leq 2$ are not index guards.

- **Array property formula** [Bradley,Manna,Sipma'06]
  $(\forall i)(\varphi_I(i) \rightarrow \varphi_V(i))$, where:

  φ_I: index guard

  φ_V: formula in which any universally quantified i occurs in a direct array read; no nestings

  Example: $c \leq x \leq y \leq d \rightarrow a(x) \leq a(y)$ is an array property formula

  Example: $x < y \rightarrow a(x) < a(y)$ is not an array property formula

Decision Procedure

(Rules should be read from top to bottom)

Step 1: Put F in NNF.

Step 2: Apply the following rule exhaustively to remove writes:

$$\frac{F[\mathit{write}(a, i, v)]}{F[a'] \wedge a'[i] = v \wedge (\forall j. j \neq i \rightarrow a[j] = a'[j])} \quad \text{for fresh } a' \text{ (write)}$$

Given a formula F containing an occurrence of a write term $\mathit{write}(a, i, v)$, we can substitute every occurrence of $\mathit{write}(a, i, v)$ with a fresh variable a' and explain the relationship between a' and a.

Decision Procedure

Step 3 Apply the following rule exhaustively to remove existential quantification:

$$\frac{F[\exists i.\, G[i]]}{F[G[j]]} \text{ for fresh } j \text{ (exists)}$$

Existential quantification can arise during Step 1 if the given formula contains a negated array property.

Decision Procedure

Steps 4-6 accomplish the reduction of universal quantification to finite conjunction.

The main idea is to select a set of symbolic index terms on which to instantiate all universal quantifiers.

Theories of arrays

Step 4 From the output F3 of Step 3, construct the index set $\mathcal{I}$:

$$\begin{aligned}\mathcal{I} \quad = \quad & \{\lambda\}\cup \\ & \{t \mid \cdot[t] \in F3 \text{ such that } t \text{ is not a universally quantified variable}\}\cup \\ & \{t \mid t \text{ occurs as an } \mathit{evar} \text{ in the parsing of index guards}\}\end{aligned}$$

(evar is any constant or unquantified variable.)

This index set is the finite set of indices that need to be examined. It includes all terms t that occur in some $\mathit{read}(a, t)$ anywhere in F (unless it is a universally quantified variable) and all terms t that are compared to a universally quantified variable in some index guard.

λ is a fresh constant that represents all other index positions that are not explicitly in $\mathcal{I}$.

Theories of arrays

Step 5 Apply the following rule exhaustively to remove universal quantification:

$$\frac{H[\forall \bar{i}. F[i] \rightarrow G[i]]}{H\left[\bigwedge_{\bar{i} \in \mathcal{I}^n} (F[\bar{i}] \rightarrow G[\bar{i}])\right]} \quad \text{(forall)}$$

where n is the size of the list of quantified variables $\bar{i}$.

This is the key step.

It replaces universal quantification with finite conjunction over the index set. The notation $\bar{i} \in \mathcal{I}^n$ means that the variables $\bar{i}$ range over all n-tuples of terms in $\mathcal{I}$.

Theories of arrays

Step 6: From the output $F5$ of Step 5, construct

$$F6: \qquad F5 \wedge \bigwedge_{i \in \mathcal{I} \setminus \{\lambda\}} \lambda \neq i$$

The new conjuncts assert that the variable λ introduced in Step 4 is unique: it does not equal any other index mentioned in F5.

Step 7: Decide the TA-satisfiability of $F6$ using the decision procedure for the quantifier free fragment.

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i.i \neq l \rightarrow a[i] = b[i])$$

It contains one array property,

$$\forall i.i \neq l \rightarrow a[i] = b[i]$$

index guard: $i \neq l$ value constraint: $a[i] = b[i]$

Step 1: The formula is already in NNF.

Step 2: We rewrite F as:

$$\begin{aligned} F2 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i.i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge (\forall j.j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Example

Consider the array property formula

$$F : \mathit{write}(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq l \rightarrow a[i] = b[i])$$

Step 2: We rewrite F as:

$$\begin{aligned} F2 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge (\forall j. j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Step 3: F2 does not contain any existential quantifiers $\mapsto$ F3 $=$ F2.

Step 4: The index set is

$$\mathcal{I} = \{\lambda\} \cup \{k\} \cup \{l\} = \{\lambda, k, l\}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i.i \neq l \rightarrow a[i] = b[i])$$

Step 3:

$$\begin{aligned} F3 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i.i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge (\forall j.j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Step 4: $\mathcal{I} = \{\lambda\} \cup \{k\} \cup \{l\} = \{\lambda, k, l\}$

Step 5: we replace universal quantification as follows:

$$\begin{aligned} F5 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge \bigwedge_{i \in \mathcal{I}} (i \neq l \rightarrow a[i] = b[i]) \\ & \wedge a'[l] = v \wedge \bigwedge_{i \in \mathcal{I}} (j \neq l \rightarrow a[j] = a'[j]). \end{aligned}$$

Example

Consider the array property formula

$$F : \mathit{write}(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i])$$

$$\mathcal{I} = \{\lambda\} \cup \{k\} \cup \{l\} = \{\lambda, k, l\}$$

Step 5 (continued) Expanding produces:

$$\begin{aligned} F5' : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq l \rightarrow a[k] = b[k]) \wedge (l \neq l \rightarrow a[l] = b[l]) \\ & \wedge a'[l] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \wedge (k \neq l \rightarrow a[k] = a'[k]) \\ & \wedge (l \neq l \rightarrow a[l] = a'[l]). \end{aligned}$$

Example

Consider the array property formula

$$F : \mathit{write}(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i])$$

$$\mathcal{I} = \{\lambda\} \cup \{k\} \cup \{l\} = \{\lambda, k, l\}$$

Step 5 (continued): Simplifying produces

$$\begin{aligned} F''5 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq l \rightarrow a[k] = b[k]) \\ & \wedge a'[l] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \\ & \wedge (k \neq l \rightarrow a[k] = a'[k]). \end{aligned}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i . i \neq l \rightarrow a[i] = b[i])$$

Step 6 distinguishes λ from other members of I:

$$\begin{aligned} F6 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = b[\lambda]) \\ & \wedge (k \neq l \rightarrow a[k] = b[k]) \\ & \wedge a'[l] = v \wedge (\lambda \neq l \rightarrow a[\lambda] = a'[\lambda]) \\ & \wedge (k \neq l \rightarrow a[k] = a'[k]) \wedge \lambda \neq k \wedge \lambda \neq l . \end{aligned}$$

Example

Consider the array property formula

$$F : write(a, l, v)[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge (\forall i. i \neq l \rightarrow a[i] = b[i])$$

Step 6 Simplifying, we have

$$\begin{aligned} F'6 : \quad & a'[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \\ & \wedge a[\lambda] = b[\lambda] \wedge (k \neq l \rightarrow a[k] = b[k]) \\ & \wedge a'[l] = v \wedge a[\lambda] = a'[\lambda] \\ & \wedge (k \neq l \rightarrow a[k] = a'[k]) \wedge \lambda \neq k \wedge \lambda \neq l. \end{aligned}$$

There are two cases to consider.

(1) If $k{=}l$, then $a'[l]{=}v$ and $a'[k]{=}b[k]$ imply $b[k]{=}v$, yet $b[k]{\neq}v$.

(2) If $k{\neq}l$, then $a[k]{=}v$ and $a[k]{=}b[k]$ imply $b[k]{=}v$, but again $b[k]{\neq}v$.

Hence, F'6 is TA-unsatisfiable, indicating that F is TA-unsatisfiable.