Limitations of SMT Solvers

Ruzica Piskac
Max-Planck Institute for Software Systems, Germany

Literature:

- This talk will be based on
  - Talking to SMT developers
  - Talking to people working in software model checking
  - Own experience in using SMT solvers

  - Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, Albert Rubio: "*Challenges in Satisfiability Modulo Theories*". RTA 2007: 2-18

SMT Solvers

What are they and how do they work?

SMT Solvers

- Used as a core engine in many tools in
  - Program analysis
  - Software engineering
  - Program model checking
  - Hardware verification, …
- Combine propositional satisfiability search techniques with specialized theory solvers
  - Linear arithmetic
  - Bit vectors
  - Uninterpreted functions with equality

Lazy approach to SMT

$$x < 1 \wedge (x \geq 1 \vee x \geq 2 \vee x \leq 3)$$

atoms are abstracted as Boolean variables

$$p_1 \wedge (\neg p_1 \vee p_2 \vee p_3)$$

A SAT solver returns

$$P_1 = true, p_2 = true$$

We create a formula corresponding to this assignment

$$x < 1 \wedge x \geq 2$$

A theory solver checks its satisfiability

UNSAT

Lazy approach to SMT

$$x < 1 \wedge (x \geq 1 \vee x \geq 2 \vee x \leq 3)$$

The formula is added to the original formula to prevent its further derivations

$$x \geq 1 \vee x < 2$$

Its negation is a valid formula

$$x < 1 \wedge x \geq 2$$

UNSAT

Lazy approach to SMT

$$x < 1 \wedge (x \geq 1 \vee x \geq 2 \vee x \leq 3) \wedge (x \geq 1 \vee x < 2)$$

$$p_1 \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2)$$

$p_1 = true, p_2 = false,$
$p_3 = true$

$$x < 1 \wedge x < 2 \wedge x \leq 3$$

SAT

$x = 0$

Combining Different Theories

- Based on the Nelson-Oppen combination procedures
  - Theories need to be disjoint, i.e. they share only the equality symbol
  - Theories need to be stably infinite, i.e. if a formula is satisfiable in some model of a theory, then it is also satisfiable in a model of infinite cardinality
  - For complexity purposes, it is desirable that a theory is convex:
    - $S \models x_1 = y_1 \vee \ldots \vee x_n = y_n$ then $S \models x_i = y_i$ for some I
    - Non-convex theory: linear integer arithmetic
    - $1 \leq x \leq 2 \models x = 1 \vee x = 2$

Nelson Oppen Combination Procedure

- Step 1:
  - Purification = converting formula into an equisatisfiable formula which is a conjunction of formulas, each belonging to a different theory
- Step 2 (loop):
  - Deduction and propagation = theory solvers deduce equalities between shared variables and propagate those equalities to other conjuncts. Repeat the process
  - If any theory solver returns UNSAT, return UNSAT
  - Otherwise return SAT

Nelson-Oppen Procedure-Example

$x + 2 = y \wedge f(read(write(a, x, 3), y\text{-}2)) \neq f(y - x + 1)$

Linear Integer Arithmetic	Arrays	EUF

Nelson-Oppen Procedure-Example

$x + 2 = y \wedge f(read(write(a, x, 3), y-2)) \neq f(y - x + 1)$

Linear Integer Arithmetic	Arrays	EUF

Nelson-Oppen Procedure-Example

$f(read(write(a, x, 3), y-2)) \neq f(y - x + 1)$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$		

Nelson-Oppen Procedure-Example

$$f(read(write(a, x, 3), y-2)) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$		

Nelson-Oppen Procedure-Example

$$f(read(write(a, x, u_1), y\text{-}2)) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$		

Nelson-Oppen Procedure-Example

$$f(read(write(a, x, u_1), y\text{-}2)) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$		

Nelson-Oppen Procedure-Example

$$f(read(write(a, x, u_1), u_2)) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$		

Nelson-Oppen Procedure-Example

$$f(read(write(a, x, u_1), u_2)) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$		

Nelson-Oppen Procedure-Example

$$f(u_3) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$	$u_3 = read(write(a, x, u_1), u_2)$	

Nelson-Oppen Procedure-Example

$$f(u_3) \neq f(y - x + 1)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$	$u_3 = read(write(a, x, u_1), u_2)$	

Nelson-Oppen Procedure-Example

$$f(u_3) \neq f(u_4)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$	$u_3 = read(write(a, x, u_1), u_2)$	

Nelson-Oppen Procedure-Example

$$f(u_3) \neq f(u_4)$$

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$	$u_3 = read(write(a, x, u_1), u_2)$	

Nelson-Oppen Procedure-Example

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$	$u_3 = read(write(a, x, u_1), u_2)$	$f(u_3) \neq f(u_4)$

Nelson-Oppen Procedure-Example

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$ $u_2 = x$ $u_4 = u_1$	$u_3 = read(write(a, x, u_1), u_2)$	$f(u_3) \neq f(u_4)$

Nelson-Oppen Procedure-Example

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$ $u_2 = x$ $u_4 = u_1$	$u_3 = read(write(a, x, u_1), u_2)$ $u_2 = x$ $u_4 = u_1$	$f(u_3) \neq f(u_4)$ $u_2 = x$ $u_4 = u_1$

Nelson-Oppen Procedure-Example

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$ $u_2 = x$ $u_4 = u_1$	$u_3 = read(write(a, x, u_1), u_2)$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$	$f(u_3) \neq f(u_4)$ $u_2 = x$ $u_4 = u_1$

Nelson-Oppen Procedure-Example

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$	$u_3 = read(write(a, x, u_1), u_2)$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$	$f(u_3) \neq f(u_4)$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$

Nelson-Oppen Procedure-Example

Linear Integer Arithmetic	Arrays	EUF
$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$	$u_3 = read(write(a, x, u_1), u_2)$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$	$f(u_3) \neq f(u_4)$ $u_2 = x$ $u_4 = u_1$ $u_3 = u_1$ UNSAT

Shaz’s Example

$$x = y \ \wedge \ f(x - y) \neq f(y - x)$$

Linear Integer Arithmetic	EUF

Shaz's Example

$$x = y \ \wedge \ f(x - y) \neq f(y - x)$$

Linear Integer Arithmetic	**EUF**
$x = y$ $u_1 = x - y$ $u_2 = y - x$	$f(u_1) \neq f(u_2)$

Shaz’s Example

$$x = y \;\wedge\; f(x - y) \neq f(y - x)$$

Linear Integer Arithmetic	EUF
$x = y$ $u_1 = x - y$ $u_2 = y - x$ $u_1 = u_2$	$f(u_1) \neq f(u_2)$

Shaz's Example

$$x = y \wedge f(x - y) \neq f(y - x)$$

Linear Integer Arithmetic	**EUF**
$x = y$ $u_1 = x - y$ $u_2 = y - x$ $u_1 = u_2$	$f(u_1) \neq f(u_2)$ $u_1 = u_2$

Shaz's Example

$$x = y \wedge f(x - y) \neq f(y - x)$$

Linear Integer Arithmetic	**EUF**
$x = y$ $u_1 = x - y$ $u_2 = y - x$ $u_1 = u_2$	$f(u_1) \neq f(u_2)$ $u_1 = u_2$ UNSAT

Congruence Closure Algorithm

- Used for checking satisfiability of EUF formulas
- Given a set of equalities, the congruence closure algorithm computes the smallest set of implied equlities
- Usually based an efficient DAG implementation
- The basic rule for deduction
  - $x_1 = y_1, \ldots, x_n = y_n \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Limitations of SMT Solvers

… there are no limitations…. ☺

“System Out of Resources”

- Generated formulas are too large
  - Encoding is often done automatically and it becomes costly to solve
  - This also influences the size of the generated proof (in case of UNSAT) – computation of minimal infeasible subset
- Finding a tailor-made encoding for a specific problem can drastically decrease the size of the formulas
- Empirical results show that the splits should be done at the leaves of the search
- One should also consider eager splitting on the literals that do not appear in the input formula

Lack of Decision Procedures

- Research and development of theory solvers are guided by the needs of users
- Some decidable theories do not have efficient theory solvers
  - Floating point arithmetic (work in progress, a PhD student @NYU)
  - Real algebra (work in progress, a PhD student @RWTH Aachen)
- Is decidability overrated? – some SMT solvers provide a limited support for undecidable theories (Z, +, *)

Handling of Quantifiers

- Some SMT solvers provide a support for quantifiers

$$\forall x_1, x_2, x_3 : (\mathrm{subtype}(x_1, x_2) \land \mathrm{subtype}(x_2, x_3) \rightarrow \mathrm{subtype}(x_1, x_3))$$

- Basic idea:
  - Select a number of ground atoms
  - Instantiate the formula with those ground atoms and check satisfiability of the new formula

$$\boxed{\neg P(f(a)) \land \forall x.\, P(x)} \xrightarrow{a,\ f(a)} \boxed{\neg P(f(a)) \land P(a) \land P(f(a))}$$

  - If it is unsatisfiable return UNSAT; otherwise ???

Handling of Quantifiers

- For some fragments there are COMPLETE techniques
  - Essentially uninterpreted fragment [Ge, de Moura, CAV'09]:
    - variables can appears only as an argument of uninterpreted function or predicate symbols (NO: P(f(g(x+y)) !)
  - Local Theory Extensions [Jacobs, CAV'09]
    - Local theories: monotone functions, injective functions, guarded boundness condition

$$\forall x.\ g(x) \rightarrow s(x) \leq f(x) \leq t(x)$$

  - Using E-matching to instantiate quantifiers [de Moura, Bjorner, CADE'07]

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \land \forall x.\ \forall y.\ g(x) \leq 0 \land g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$
$f(V_y) \subseteq A_g$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$
$f(V_y) \subseteq A_g$
$V_y = A_f$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \land \forall x.\ \forall y.\ g(x) \leq 0 \land g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$
$f(V_y) \subseteq A_g$
$V_y = A_f$

Solution:

$A_f = \{a\}$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$
$f(V_y) \subseteq A_g$
$V_y = A_f$

Solution:

$A_f = \{a\}$
$V_y = \{a\}$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$
$f(V_y) \subseteq A_g$
$V_y = A_f$

Solution:

$A_f = \{a\}$
$V_y = \{a\}$
$A_g = \{f(a)\}$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

V_x = ground terms for instantiating variable x
A_f = ground terms that will appear as arguments of function f

Constraints on sets V_i and F_j:

$a \in A_f$
$V_x = A_g$
$f(V_y) \subseteq A_g$
$V_y = A_f$

Solution:

$A_f = \{a\}$
$V_y = \{a\}$
$A_g = \{f(a)\}$
$V_x = \{f(a)\}$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \le 0 \wedge g(f(y)) + 1 \le f(y)$$

Instantiated formula

$$f(a) = 0 \wedge g(f(a)) \le 0 \wedge g(f(a)) + 1 \le f(a)$$

SAT

Model:
f(a) = 0, g(f(a)) = -1, a = 1

Instances:

$A_f = \{a\}$
$V_y = \{a\}$
$A_g = \{f(a)\}$
$V_x = \{f(a)\}$

Quantifiers in Essentially Uninterpreted fragment - Example

$$f(a) = 0 \wedge \forall x.\ \forall y.\ g(x) \leq 0 \wedge g(f(y)) + 1 \leq f(y)$$

SAT

Model:
$f(a) = 0$, $g(f(a)) = -1$, $a = 1$

SAT

Model for the original formula:
$\{ a \to 1, f \to \lambda x.\ 0, g \to \lambda x.\ -1\}$

On Combining Non-disjoint Theories

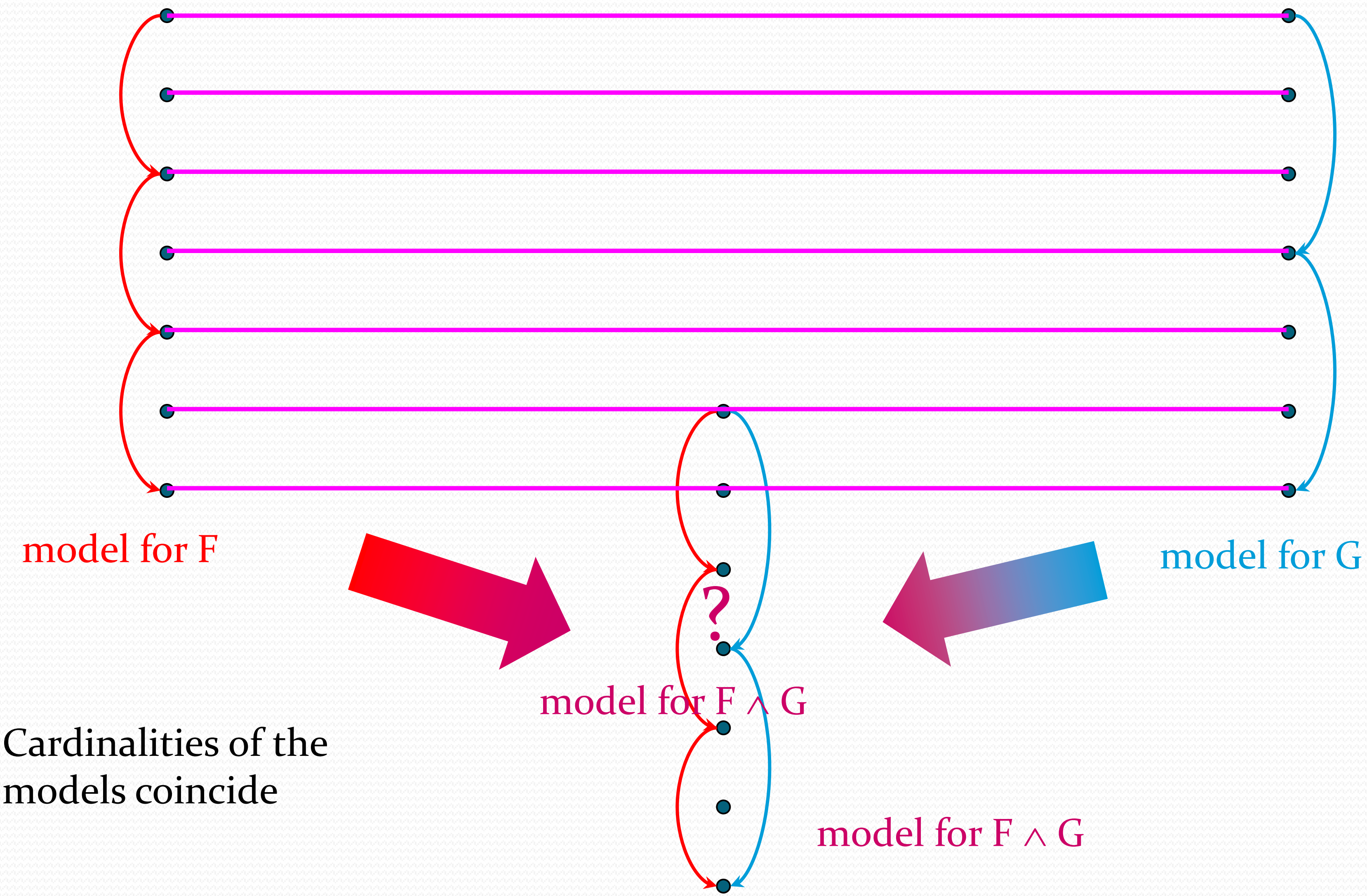
Joint work with Thomas Wies and Viktor Kuncak

Combining Different Theories

- Based on the Nelson-Oppen combination procedures
  - Theories need to be disjoint, i.e. they share only the equality symbol
  - Theories need to be stably infinite, i.e. if a formula is satisfiable in some model of a theory, then it is also satisfiable in a model of infinite cardinality
  - For complexity purposes, it is also good if a theory is convex:
    - $S \models x_1 = y_1 \vee \ldots \vee x_n = y_n$ then $S \models x_i = y_i$ for some I
    - Non-convex theory: linear integer arithmetic
    - $1 \leq X \leq 2 \models X = 1 \vee X = 2$

Amalgamation of Models:

The Disjoint Case



Cardinalities of the models coincide

Generated Verification Condition

$\neg$next0*(root0,n) $\wedge$ x $\notin$ {data0(v) | next0*(root0,v)} $\wedge$
next=next0[n:=root0] $\wedge$ data=data0[n:=x] $\rightarrow$
|{data(v) . next*(n,v)}| =
|{data0(v) . next0*(root0,v)}| + 1

"The number of stored objects has increased by one."

Expressing this VC requires a rich logic

- transitive closure * (in lists and also in trees)
- unconstraint functions (data, datao)
- cardinality operator on sets | ... |

How do we check satisfiability of such a formula?

Decomposing the Formula

Consider a (simpler) formula

$$|\{data(x).\ next^*(root,x)\}|=k+1$$

Introduce fresh variables denoting sets:

$A = \{x.\ next^*(root,x)\} \wedge$ 1) WS1S

$B = \{y.\ \exists\ x.\ data(x,y) \wedge x \in A\} \wedge$ 2) C^2

$|B|=k+1$ 3) BAPA

Good news: conjuncts are in decidable fragments

Bad news: conjuncts share more than just equality (they share set variables and set operations)

$\Rightarrow$ We cannot apply the Nelson-Oppen procedure

Combining Theories by Reduction

Satisfiability problem expressed in HOL:

(all free symbols existentially quantified)

$\exists$ next,data,k,root. $\exists$ A,B.

A = {x. next*(root,x)} $\wedge$

B = {y. $\exists$ x. data(x,y) $\wedge$ x $\in$ A} $\wedge$

|B|=k+1

1) WS1S

2) C^2

3) BAPA

We assume formulas share only:

- **set variables** (sets of uninterpreted elems)
- set operations and relations

Combining Theories by Reduction

Satisfiability problem expressed in HOL, after moving fragment-specific quantifiers

$\exists$ A,B.

$\boxed{\exists\ next, root.\ A = \{x.\ next^*(root,x)\}}\ \wedge$ F_{WS1S}

$\boxed{\exists\ data.\ B = \{y.\ \exists\ x.\ data(x,y) \wedge x \in A\}}\ \wedge$ F_{C2}

$\boxed{\exists\ k.\ |B|=k+1} \longrightarrow F_{BAPA}$

Extend decision procedures for fragments into **projection procedures** that reduce each conjunct to a decidable shared theory

applies $\exists$ to all non-set variables

Combining Theories by Reduction

Satisfiability problem expressed in HOL,

after moving fragment-specific quantifiers

$\exists$ A,B.

$\boxed{\exists \text{ next,root. } A = \{x.\ next^*(root,x)\}} \wedge$ F_{WS2S}

$\boxed{\exists \text{ data. } B = \{y.\ \exists\ x.\ data(x,y) \wedge x \in A\}} \wedge$

$\boxed{\exists \text{ k. } |B|=k+1}$ $\longrightarrow$ F_{BAPA}

F_{C2}

Check satisfiability of conjunction of projections

$$\exists \text{ A,B. } F_{WS2S} \wedge F_{C2} \wedge F_{BAPA}$$

Conjunction of projections satisfiable → so is original formula

BAPA-Reducibility

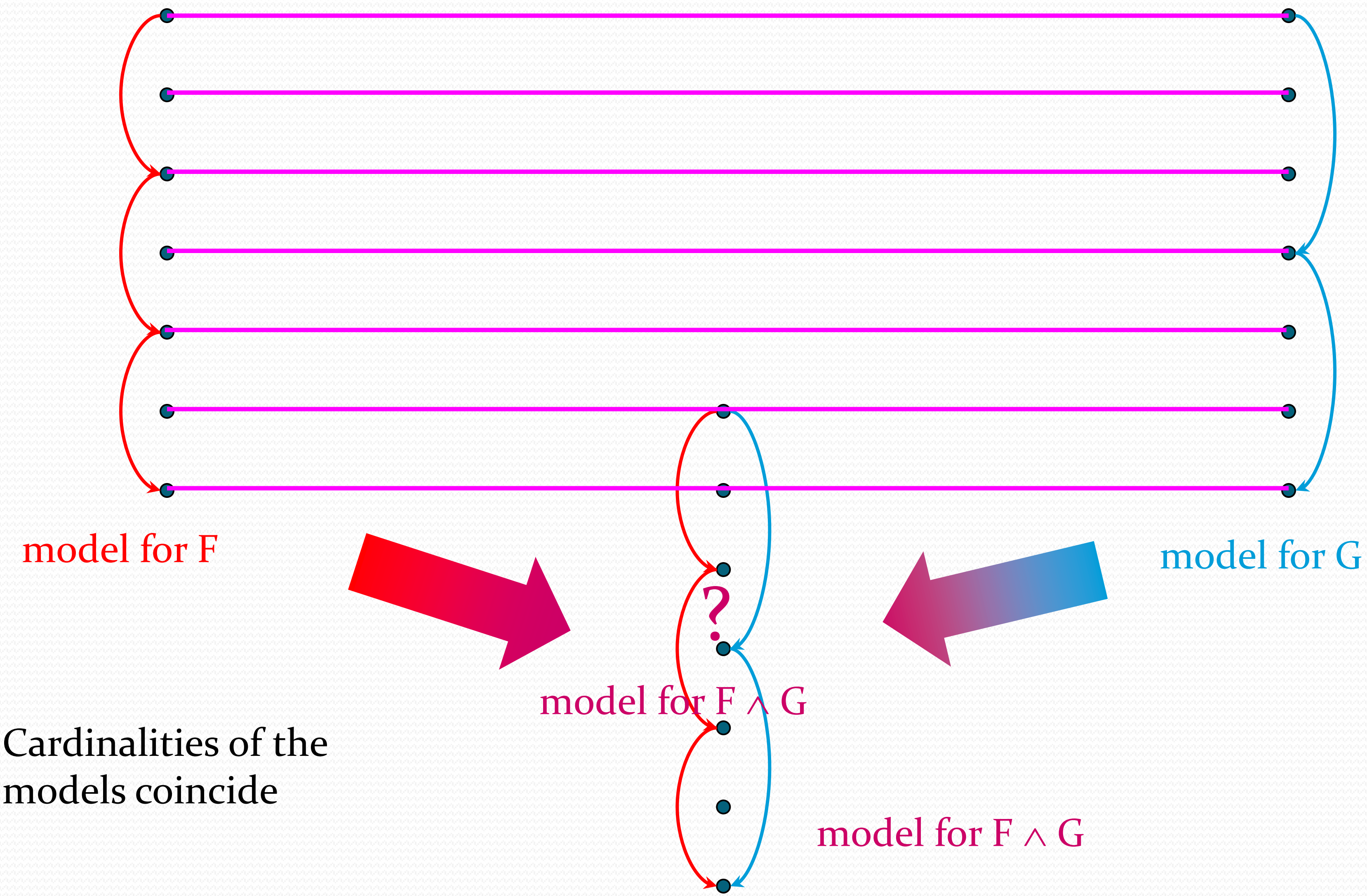
Definition: Logic is **BAPA-reducible** iff there is an algorithm that computes projections of formulas onto set variables, and these projections are BAPA formulas.

Theorem:

1) WS2S, **2)** C^2, **3)** BAPA, **4)** BSR, **5)** qf-multisets are all BAPA-reducible.

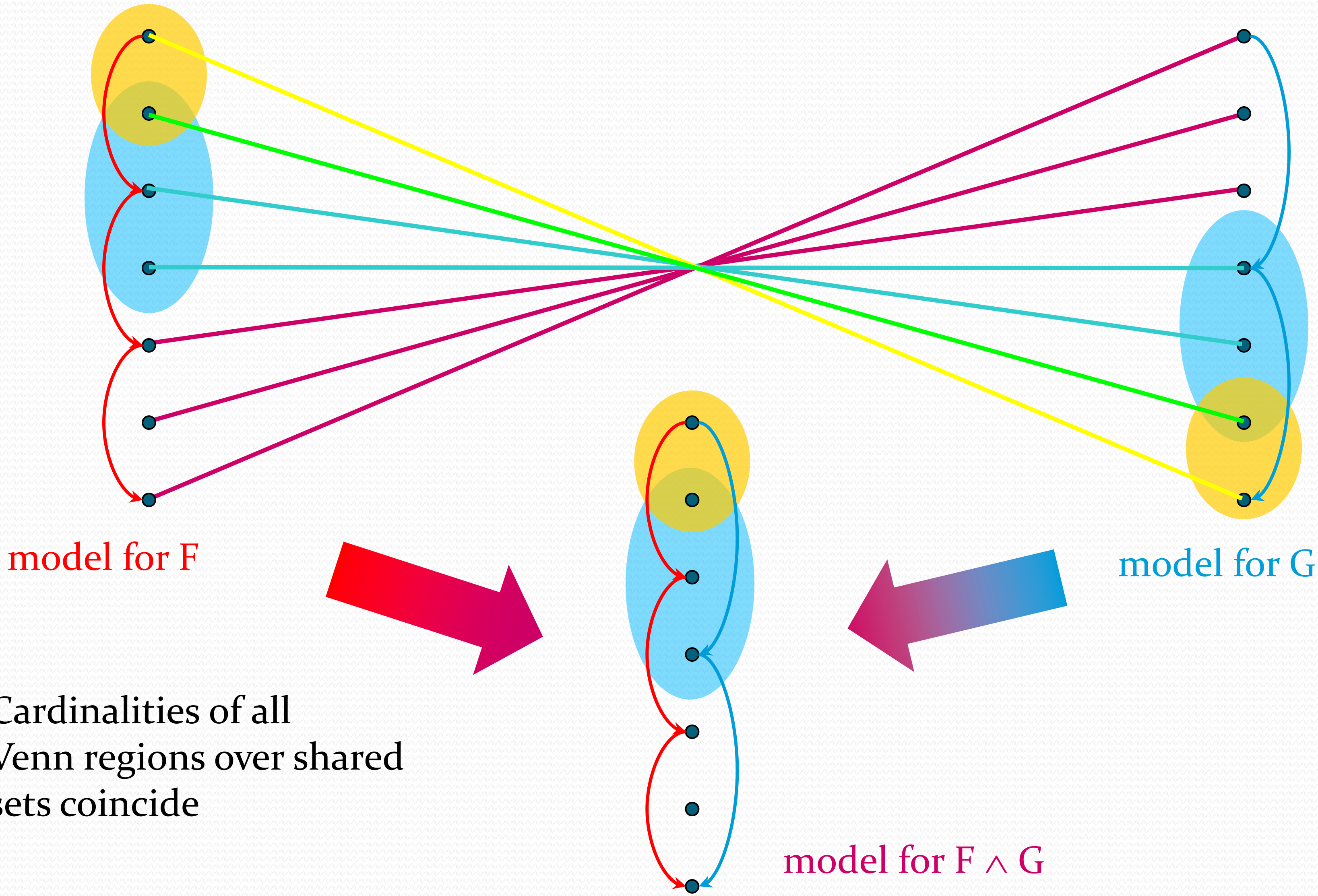
Thus, their set-sharing combination is decidable.

Amalgamation of Models:
The Disjoint Case



Cardinalities of the models coincide

Amalgamation of Models:
The Set-Sharing Case



Cardinalities of all
Venn regions over shared
sets coincide

BAPA-reduction for WS1S

WS1S formula for a regular language

$$F = ((A \wedge \neg B)(B \wedge \neg A))^* \, (\neg B \wedge \neg A)^{\boldsymbol{*}}$$

Formulas are interpreted over finite words

Symbols in alphabet correspond to

$$(\neg A \wedge \neg B),(A \wedge \neg B),(\neg A \wedge B),(A \wedge B)$$

00 10 01 11

Model of formula F

A	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
B	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0

BAPA-reduction for WS1S

WS1S formula for a regular language

$$F = ((A \wedge \neg B)(B \wedge \neg A))^* \, (\neg B \wedge \neg A)^{\boldsymbol{*}}$$

Model of formula F

A	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
B	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0

} w

A,B denote sets of positions in the word w.

00, 10, 01, 11 denote Venn regions over A,B

Parikh image gives card.s of Venn regions

$$\text{Parikh}(w) = \{ 00 \mapsto 7,\ 10 \mapsto 4,\ 01 \mapsto 4,\ 11 \mapsto 0 \}$$

BAPA-reduction for WS1S

Decision procedure for sat. of WS1S:

- construct finite word automaton A from F
- check emptiness of L(A)

Parikh 1966:

Parikh image of a regular language is semilinear and effectively computable from the finite automaton

Construct BAPA formula from Parikh image of the reg. lang.

BAPA-reduction for WS1S

WS1S formula for a regular language

$$F = ((A \wedge \neg B)(B \wedge \neg A))^{*}\ (\neg B \wedge \neg A)^{\boldsymbol{*}}$$

Parikh image of the models of F:

$$\text{Parikh}(F) = \{(q,p,p,0) \mid q,p \geq 0\}$$

00 10 01 11

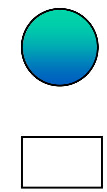
BAPA formula for projection of F onto A,B:

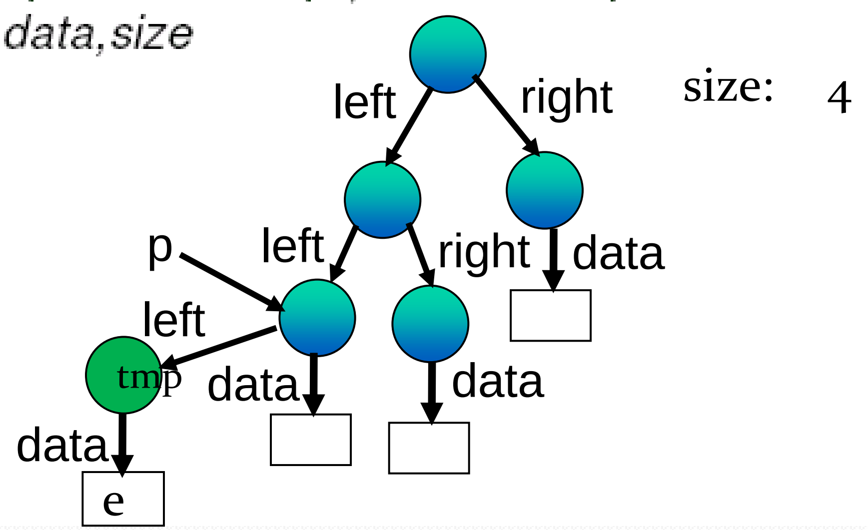
$$|A \cap B^{c}| = |A^{c} \cap B| \wedge |A \cap B| = 0$$

Fragment of Insertion into Tree

```
class Node {Node left,right; Object data;}
class Tree {
    private static Node root;
    private static int size; /*:
    private static specvar nodes :: objset;
    vardefs "nodes=={x. (root,x) ∈ {(x,y). left x = y ∨ right x = y}* }";
    private static specvar content :: objset;
    vardefs "content=={x. ∃ n. n ≠ null ∧ n ∈ nodes ∧ data n = x} " */

    private void insertAt(Node p, Object e) /*:
      requires "tree [ left , right ] ∧ nodes ⊆ Object.alloc ∧ size = card content ∧
                e ∉ content ∧ e ≠ null ∧ p ∈ nodes ∧ p ≠ null ∧ left p = null"
      modifies nodes,content,left,right,data,size
      ensures "size = card content" */
    {
        Node tmp = new Node();
        tmp.data = e;
        p.left = tmp;
        size = size + 1;
    }
}
```





Reduction of VC for insertAt

SHARED SETS: nodes, nodes1, content, content1, {e}, {tmp}

WS2S FRAGMENT:

$\text{tree}[\text{left}, \text{right}] \wedge \text{left } p = \text{null} \wedge p \in \text{nodes} \wedge \text{left tmp} = \text{null} \wedge \text{right tmp} = \text{null} \wedge$
$\text{nodes}=\{x.\ (\text{root},x) \in \{(x,y).\ \text{left } x = y \mid \text{right } x = y\}\hat{}*\} \wedge$
$\text{nodes1}=\{x.\ (\text{root},x) \in \{(x,y).\ (\text{left }(p:=\text{tmp}))\ x = y) \mid \text{right } x = y\}$

CONSEQUENCE: $\boxed{\text{nodes1}=\text{nodes} \cup \{\text{tmp}\}}$

C2 FRAGMENT:

$\text{data tmp} = \text{null} \wedge (\forall y.\ \text{data } y \neq \text{tmp}) \wedge \text{tmp} \notin \text{alloc} \wedge \text{nodes} \subseteq \text{alloc} \wedge$
$\text{content}=\{x.\ \exists n.\ n \neq \text{null} \wedge n \in \text{nodes} \wedge \text{data } n = x\} \wedge$
$\text{content1}=\{x.\ \exists n.\ n \neq \text{null} \wedge n \in \text{nodes1} \wedge (\text{data}(\text{tmp}:=e))\ n = x\}$

CONSEQUENCE: $\boxed{\text{nodes1} \neq \text{nodes} \cup \{\text{tmp}\} \vee \text{content1} = \text{content} \cup \{e\}}$

BAPA FRAGMENT: $e \notin \text{content} \wedge \text{card content1} \neq \text{card content} + 1$

CONSEQUENCE: $\boxed{e \notin \text{content} \wedge \text{card content1} \neq \text{card content} + 1}$

Conjunction of projections unsatisfiable → so is original formula

Conclusions

- SMT solvers = tools for efficiently checking satisfiability of formulas
- Although very powerful tools, there are limitations:
  - Lack of decision procedures
  - Handling of quantifiers
  - The requirements for the Nelson-Oppen combination are too restrictive
- Presented a new combination technique for theories sharing sets by reduction to a common shared theory:
  - Resulting theory is useful for automated verification of complex properties of data structure implementations