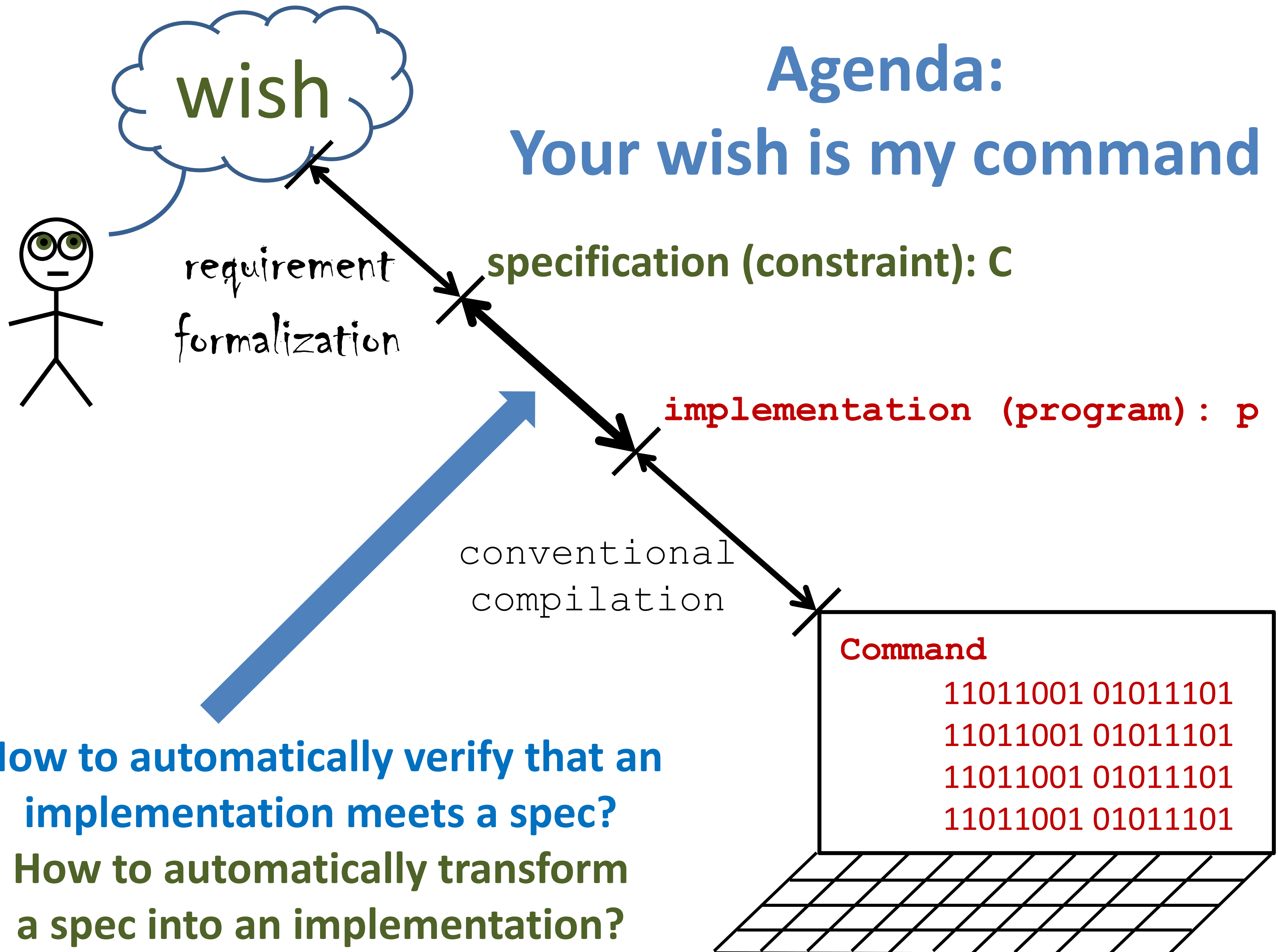
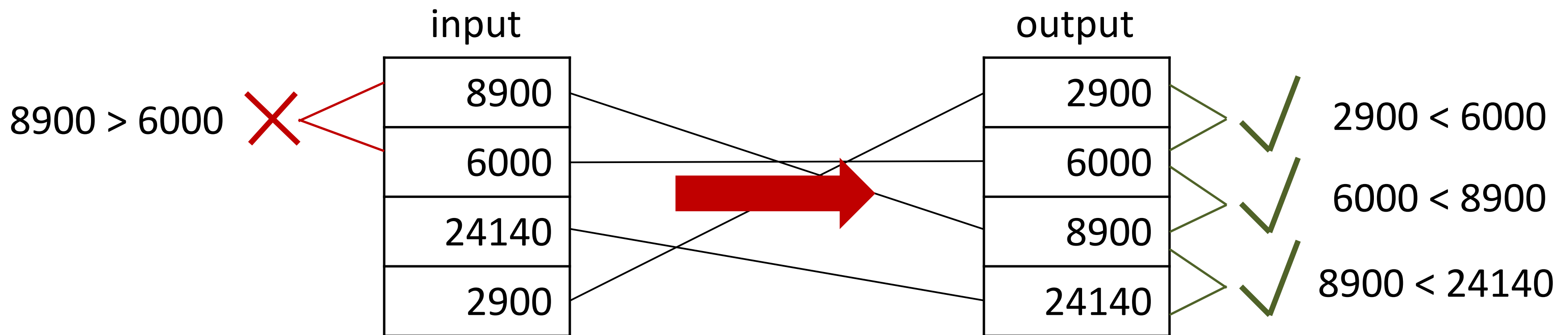
Synthesis and
Satisfiability Modulo Theory Solvers

Viktor Kuncak
EPFL
Laboratory for Automated Reasoning and Analysis

http://lara.epfl.ch http://leon.epfl.ch
http://cvc4.cs.nyu.edu/web/


wish
Agenda:
Your wish is my command
requirement
formalization
specification (constraint): C
implementation (program): p
conventional
compilation
Command
11011001 01011101
11011001 01011101
11011001 01011101
11011001 01011101
How to automatically verify that an
implementation meets a spec?
How to automatically transform
a spec into an implementation?

Sorting Specification as a Program



Given a list of numbers, make **this list** **sorted**

wish

```
def sort_spec(input : List, output : List) : Boolean =
  content(output)==content(input)   &&  isSorted(output)
```

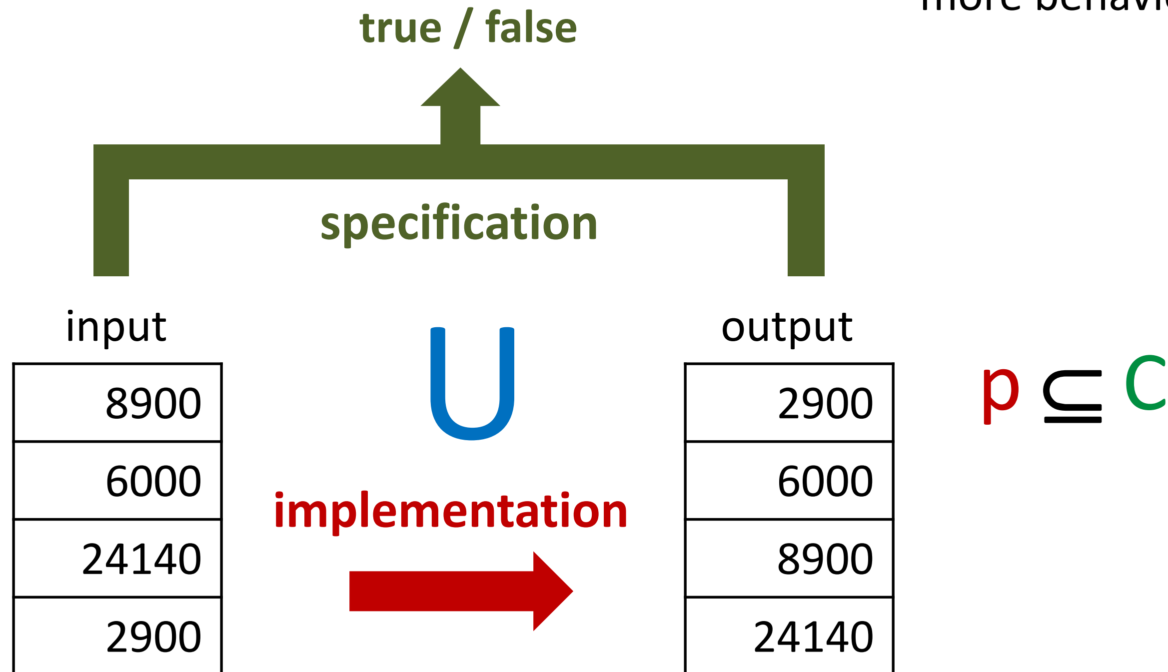
Specification (for us) is a *program* that checks, for a **given input**, whether the **given output** is acceptable

Specification vs Implementation

def C(i : List, o : List) : **Boolean** = **constraint on the output**

content(o)==content(i) && isSorted(o)

more behaviors



fewer behaviors

def p(i : List) : List = **function that computes the output (equality constraint on output)**

sort i using a sorting algorithm

http://leon.epfl.ch system for **Scala**

Scala: invented at EPFL by Prof. Martin Odersky http://scala-lang.org/

- hundreds of thousands of Scala programmers, used in:

Twitter, Foursquare, Coursera , *The Guardian*, *New York Times*, *Huffington Post*, UBS , LinkedIn , Meetup ,Verizon , Intel, …

Typesafe Inc. supports Scala commercially

EPFL: industrial advisory board, courses, open source development

Chisel: "…an open-source hardware construction language developed at UC Berkeley that supports advanced hardware design using highly parameterized generators and layered domain-specific hardware languages." – other Scala DSLs: e.g. OptiML

Apache Spark: "an open-source cluster computing framework with in-memory processing to speed analytic applications up to 100 times faster compared to technologies on the market today. Developed in the AMPLab at UC Berkeley, Apache Spark can help reduce data interaction complexity, increase processing speed and enhance mission-critical applications with deep intelligence."

"…IBM is making a major commitment to the future of Apache Spark, with a series of initiatives announced today. IBM will offer Apache Spark as a service on Bluemix; commit 3,500 researchers to work on Spark-related projects; donate IBM SystemML to the Spark ecosystem; and offer courses to train 1 million data scientists and engineers to use Spark."

DEMO

http://leon.epfl.ch

a) Check assertion while program **p** runs: **C(i,p(i))**

b) Verify whether program always meets the spec: **∀i. C(i,p(i))**

repair

c) Constraint programming: once **i** is known, find **o** to satisfy a given constraint: **find o such that C(i,o)**

d) Synthesis: solve **C** symbolically to obtain program **p** that is correct by construction, for all inputs: **find p such that ∀i.C(i,p(i))** **i.e.** $p \subseteq C$

Approaches and Their Guarantees

both specification C and program p are given:

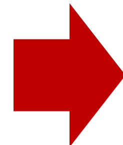
a) Check assertion while program p runs: **C(i,p(i))**

b) Verify that program always meets spec: **$\forall$i. C(i,p(i))**

only specification C is given:

c) Constraint programming: once **i** is known, find **o** to satisfy a given constraint: **find o such that C(i,o)**

run-time



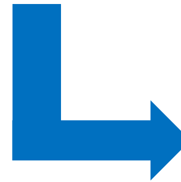
d) Synthesis: solve C symbolically to obtain program p that is correct by construction, for all inputs: **find p such that $\forall$i.C(i,p(i))** **i.e. p $\subseteq$ C**

compile-time

Synthesizing Sort in Leon System

```
def insertSorted(lst : List, v: Int): List = {
  require(isSorted(lst))
  choose { (r: List) ⇒
    isSorted(r) && content(r) == content(lst) ++ Set(v) } }

def sort(lst : List): List = choose { (r: List) ⇒
  isSorted(r) && content(r) == content(lst) }
```



```
def insertSorted(lst: List, v: Int): List = {
  require(isSorted(lst))
  lst match {
    case Nil ⇒ Cons(v, Nil)
    case Cons(h, tail) ⇒
      val r = insertSorted(t, v)
      if (v > h) Cons(h, r)
      else if (h == v) r
      else Cons(v, Cons(h, t)) }

def sort(lst : List): List = lst match {
  case Nil ⇒ Nil
  case Cons(h, t) ⇒ insertSorted(sort(t), h) }
```

http://leon.epfl.ch

OOPSLA 2013:
Synthesis Modulo Recursive Functions

Etienne Kneuss

Ivan Kuraj

Philippe Suter

Recursion Schemas + STE in Action

```
def delete(in1: List, v: Int) = choose {
  (out: List) => content(out) == content(in1) -- Set(v)
}
```

```
def delete(in1: List,  v: Int)  = {
  def rec(in: List, v: Int): List = in match {
    case Cons(h,t) =>
      val r = rec(t,v)
      if (h == v) {
        r
      } else {
        Cons(h, r)
      }
    case Nil =>
      Nil
  } ensuring { content(_) == content(in1) -- Set(v) }
  rec(in1, v)
}
```

ADT Induction

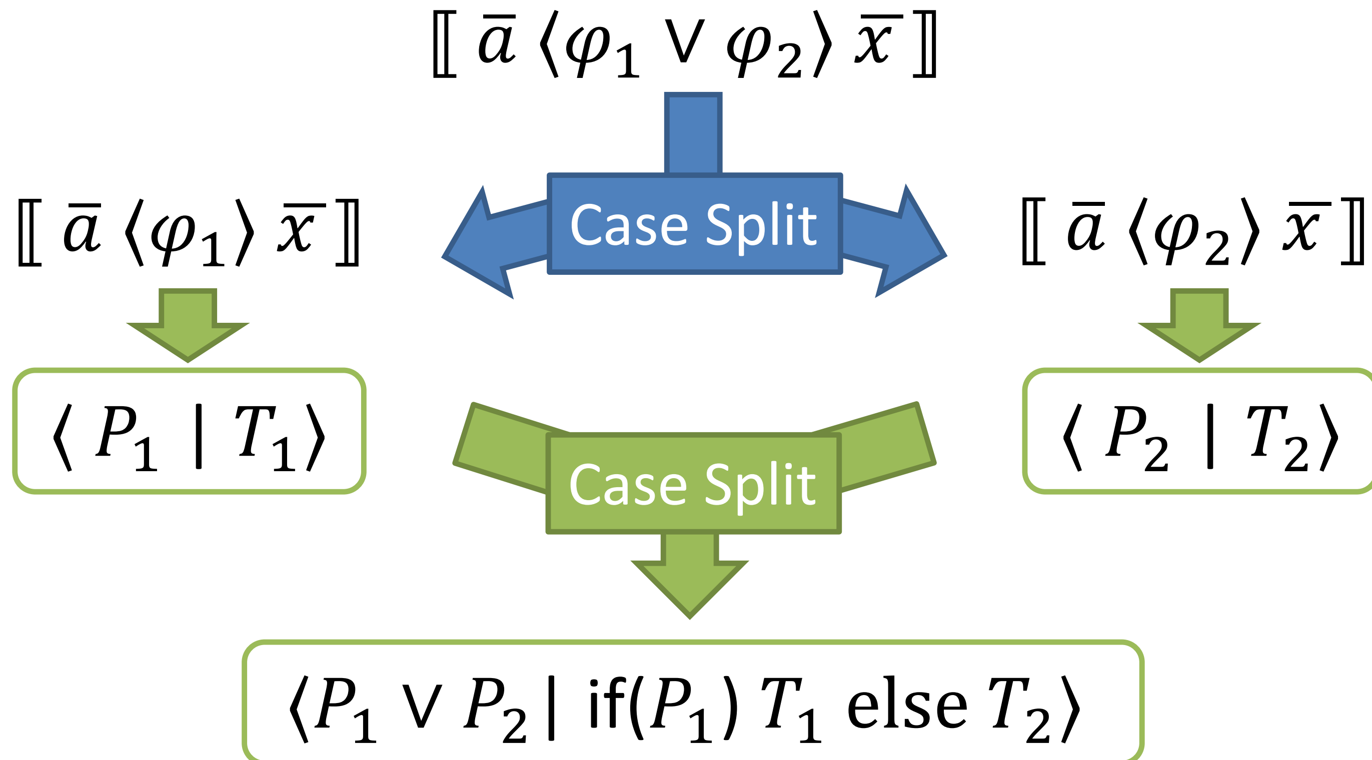
EQ Split

CEGIS

CEGIS

CEGIS

Decomposition Example: Case Split



Symbolic Term Exploration (STE)

Symbolic search over many expressions of bounded size

$T(\bar{b})$ = if (b_0) a_0
elseif (b_1) Nil
elseif (b_2) $\mathrm{Cons}(0, \mathrm{Nil})$
elseif (b_3) ...

SMT solver searches exponentially many expressions given by polynomially many Boolean variables

Concrete execution prunes search space

Leon's verifier validates candidate expressions

- using again SMT solvers

Building Block: Complete Synthesis

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
         h * 3600 + m * 60 + s == totalSeconds
    && h ≥ 0
    && m ≥ 0 && m < 60
    && s ≥ 0 && s < 60    ))
```

spec in integer linear arithmetic

synthesis procedure

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  val t1 = totalSeconds div 3600
  val t2 = totalSeconds -3600 * t1
  val t3 = t2 div 60
  val t4 = totalSeconds - 3600 * t1 - 60 * t3
  (t1, t3, t4)
```

Kuncak, Mayer, Piskac, Suter

PLDI'10
CACM'12

Implementation

V1) Quantifier elimination procedure for Presburger arith.

- optimization for some important cases
- inefficient in general

V2) Automata-based procedure for int. arithmetic with bitwise operations (Hamza, Jobstmann, K., FMCAD'10)

- handles larger subset
- bad at e.g. multiplication by large constants - we do not know good general techniques for sequential circuits

V3) Inside an SMT solver: **Andrew Reynolds**, Morgan Deters, Cesare Tinelli, Clark Barrett, K. – CAV'15

➔ focus today

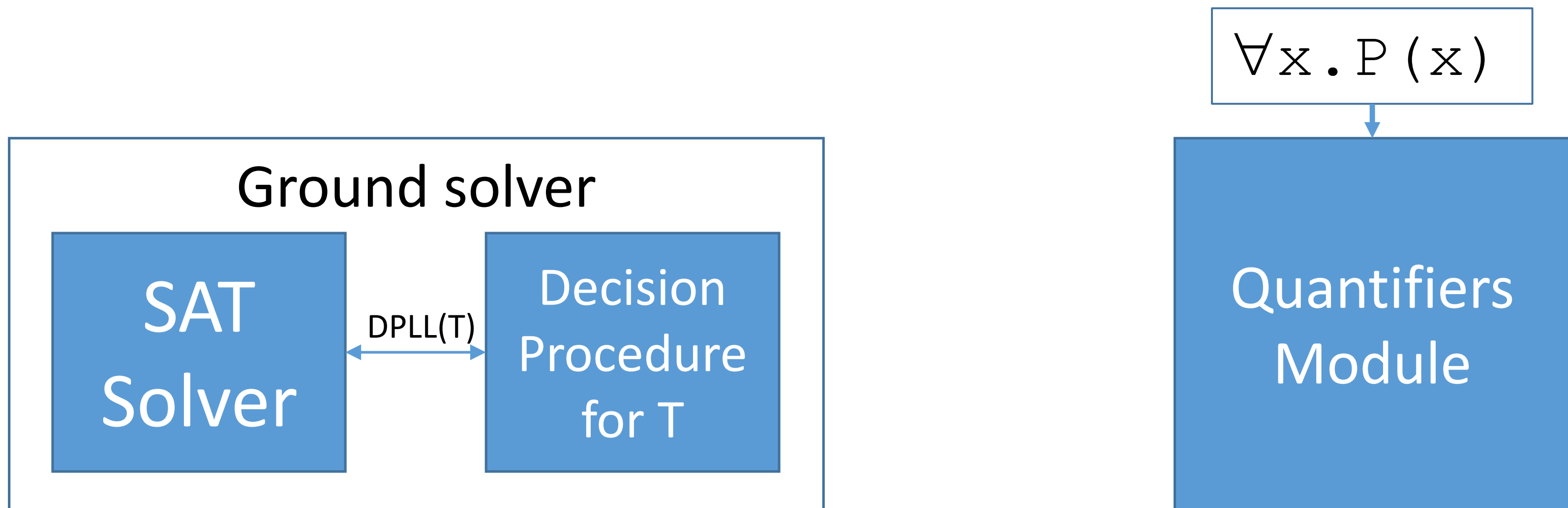
Synthesis Problem for an SMT Solver

- Synthesis Problem : $\exists f . \forall x . P(f, x)$

There exists a function f such that for all x, $P(f, x)$

- Most existing approaches for synthesis
  - Rely on specialized solver that makes subcalls to an SMT Solver
- Goal: approach implemented *entirely inside SMT solver*

SMT Solver + Quantified Formulas



SMT solver consists of:

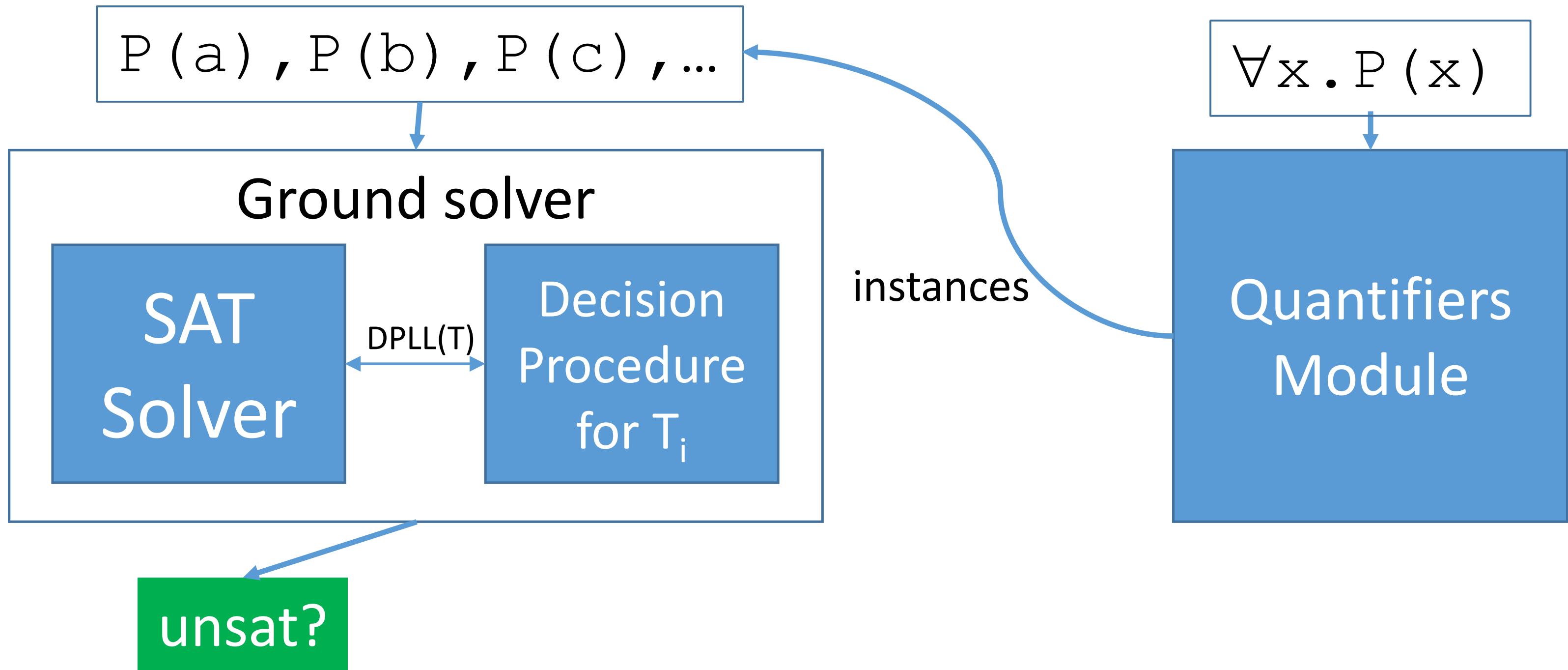
- Ground solver maintains a set of ground (variable-free) constraints
- Quantifiers Module maintains a set of quantified formulas: $\forall x.P(x)$

Using SMT solvers: game-changer in automated software verification

Increasingly relevant in industry:

- symbolic execution of systems code, microcode, word-level reasoning
- Anders Franzén, Alessandro Cimatti, Alexander Nadel, Roberto Sebastiani, Jonathan Shalev: **Applying SMT in symbolic execution of microcode.** FMCAD 2010: 121-128 – **Best Paper Award**

SMT Solver + Quantified Formulas



- Goal : add instances of axioms until ground solver can answer "unsat"

Running Example : Max of Two Integers

$$\exists f. \forall xy. (f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y)=x \vee f(x,y)=y))$$

- Specifies that f computes the maximum of integers x and y
- A solution:

$$f := \lambda xy.ite(x \geq y, x, y)$$

Approach: Refutation-Based Synthesis

$$\neg \exists f. \forall \mathbf{x}. P(f, \mathbf{x})$$

- What if we negate the synthesis conjecture?
- If we are in a *satisfaction-complete* theory T (e.g. LIA, BV):
  - *F is T-satisfiable if and only if $\neg F$ is T-unsatisfiable*

$\Rightarrow$ Will suffice for us to show the above formula is unsat

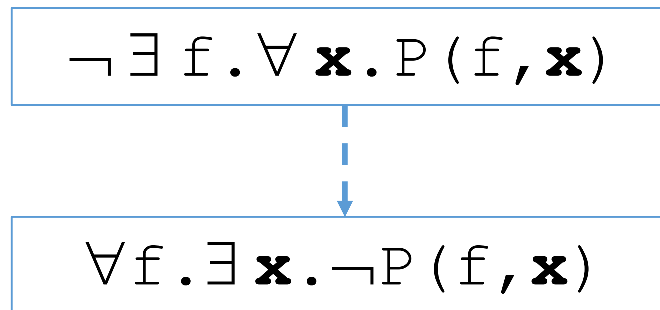
Challenge: Second-Order Quantification

$\neg\exists f.\forall \mathbf{x}.P(f,\mathbf{x})$

$\forall f.\exists \mathbf{x}.\neg P(f,\mathbf{x})$

- Challenge: negation introduces universal ∀ **over function `f`**
  - No SMT solvers directly support second-order quantification

Challenge: Second-Order Quantification



- Challenge: negation introduces universal $\forall$ over function `f`
  - No SMT solvers directly support second-order quantification
- However, we can avoid this quantification using two approaches:
  1. When property `P` is single invocation for `f` 🡐 focus now
  2. When `f` is given syntactic restrictions

Single Invocation Properties

$$\forall f.\ \exists xy.\ (f(x,y)<x \lor f(x,y)<y \lor (f(x,y)\neq x \land f(x,y)\neq y))$$

Single Invocation Properties

$$\forall f. \exists xy. (f(x,y) < x \lor f(x,y) < y \lor (f(x,y) \neq x \land f(x,y) \neq y))$$

- *Single invocation* properties
  - Are properties such that:
    - All occurrences of `f` are of a particular form, e.g. `f(x,y)` above
  - Are a common class of properties useful for:
    - Software Synthesis (post-conditions describing the result of a function)
  - Given solution, it can be checked without replicating solution

NOT single invocation: “f is commutative”

Single Invocation Properties

$$\forall f. \exists xy. (f(x,y)<x \lor f(x,y)<y \lor (f(x,y) \neq x \land f(x,y) \neq y))$$

Push $\forall$ quantification inwards

$$\exists xy. \forall g. (g<x \lor g<y \lor (g \neq x \land g \neq y))$$

- Occurrences of $f(x,y)$ are replaced with integer variable g
- Resulting formula is equisatisfiable, and first-order

Single Invocation Properties

$$\forall f.\ \exists xy.\ (f(x,y)<x \lor f(x,y)<y \lor (f(x,y)\neq x \land f(x,y)\neq y))$$

$$\exists xy.\ \forall g.\ (g<x \lor g<y \lor (g\neq x \land g\neq y))$$

Skolemize, for fresh a and b

$$\forall g.\ (g<a \lor g<b \lor (g\neq a \land g\neq b))$$

Solving Max Example

$\forall g.(g<a \lor g<b \lor (g \neq a \land g \neq b))$

Ground solver

Quantifiers Module

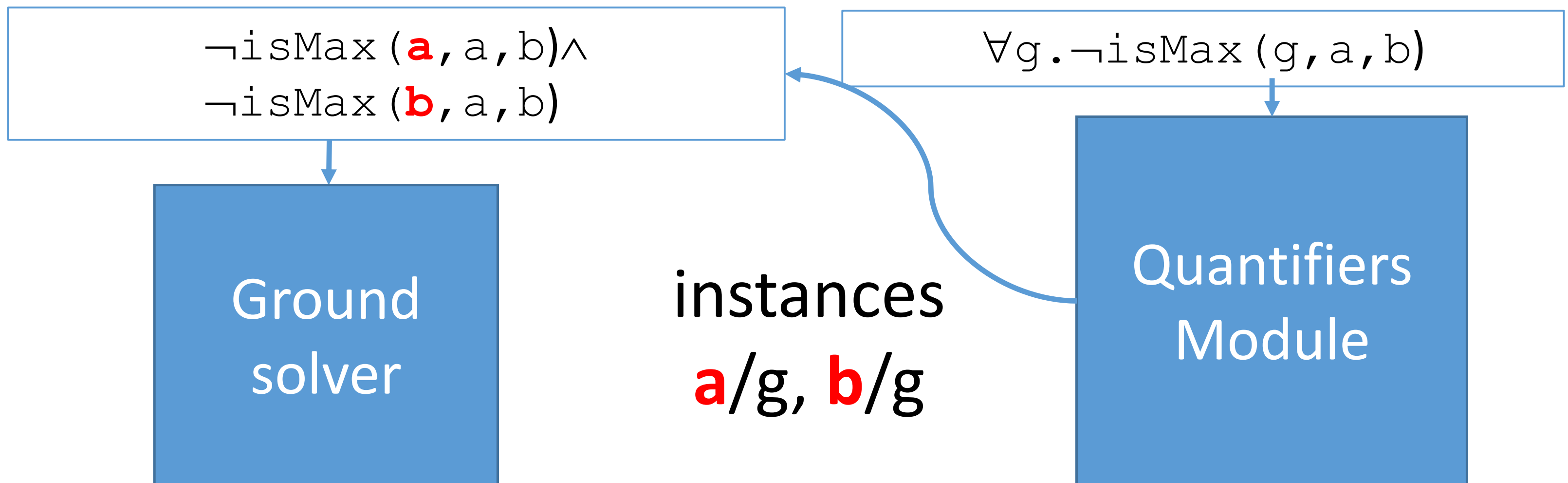
Solving Max Example

$\forall$g.¬isMax(g,a,b)

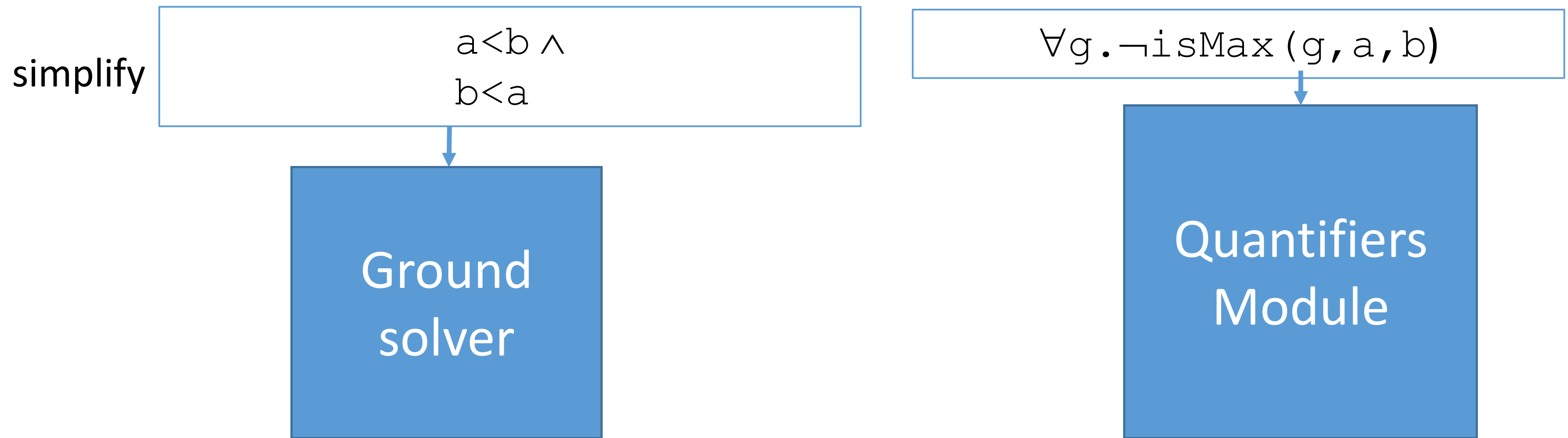
Ground solver

Quantifiers Module

Solving Max Example



Solving Max Example



Solving Max Example

$a<b \wedge$
$b<a$

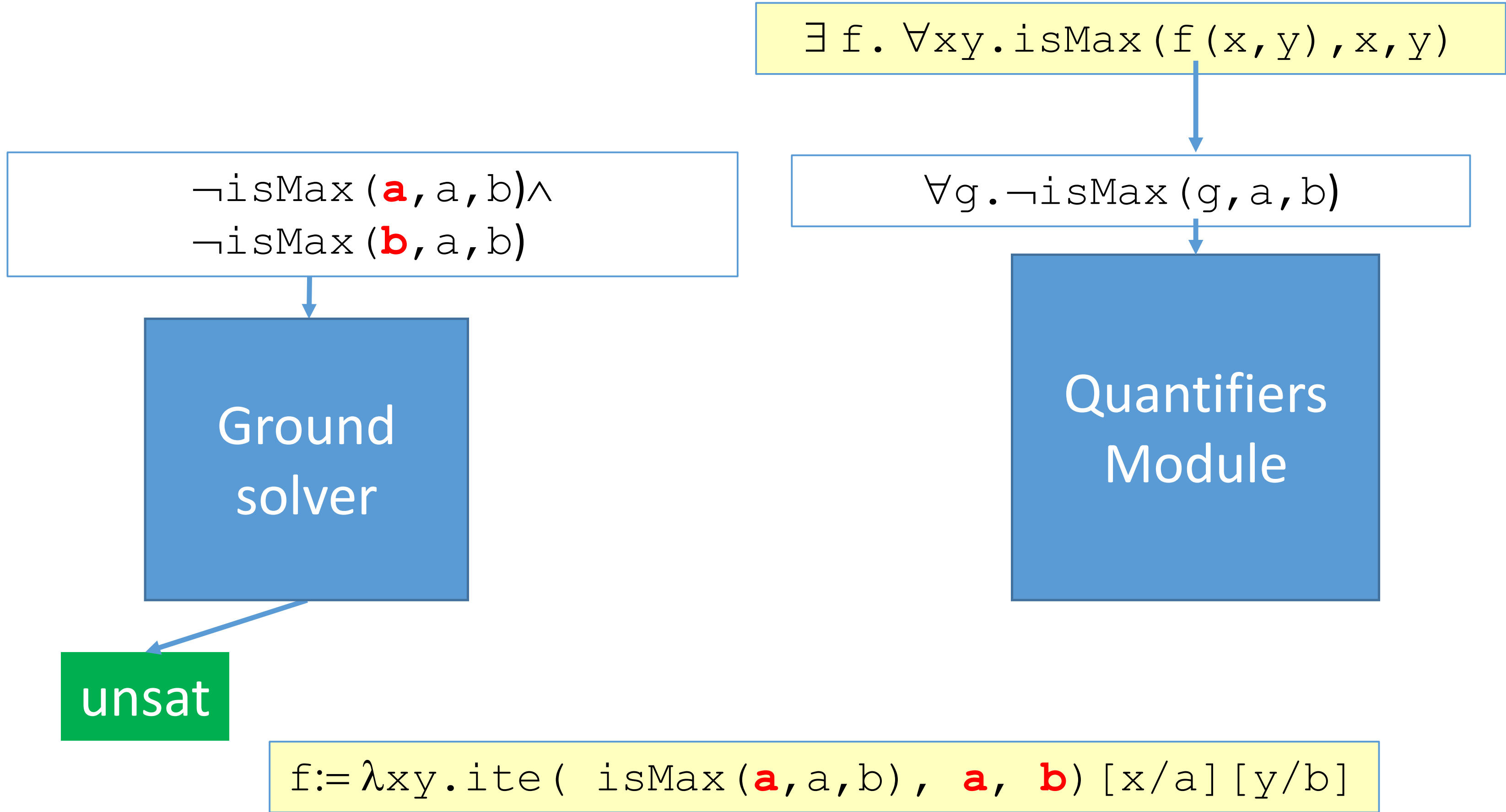
$\forall g.\neg isMax(g,a,b)$

Ground solver

Quantifiers Module

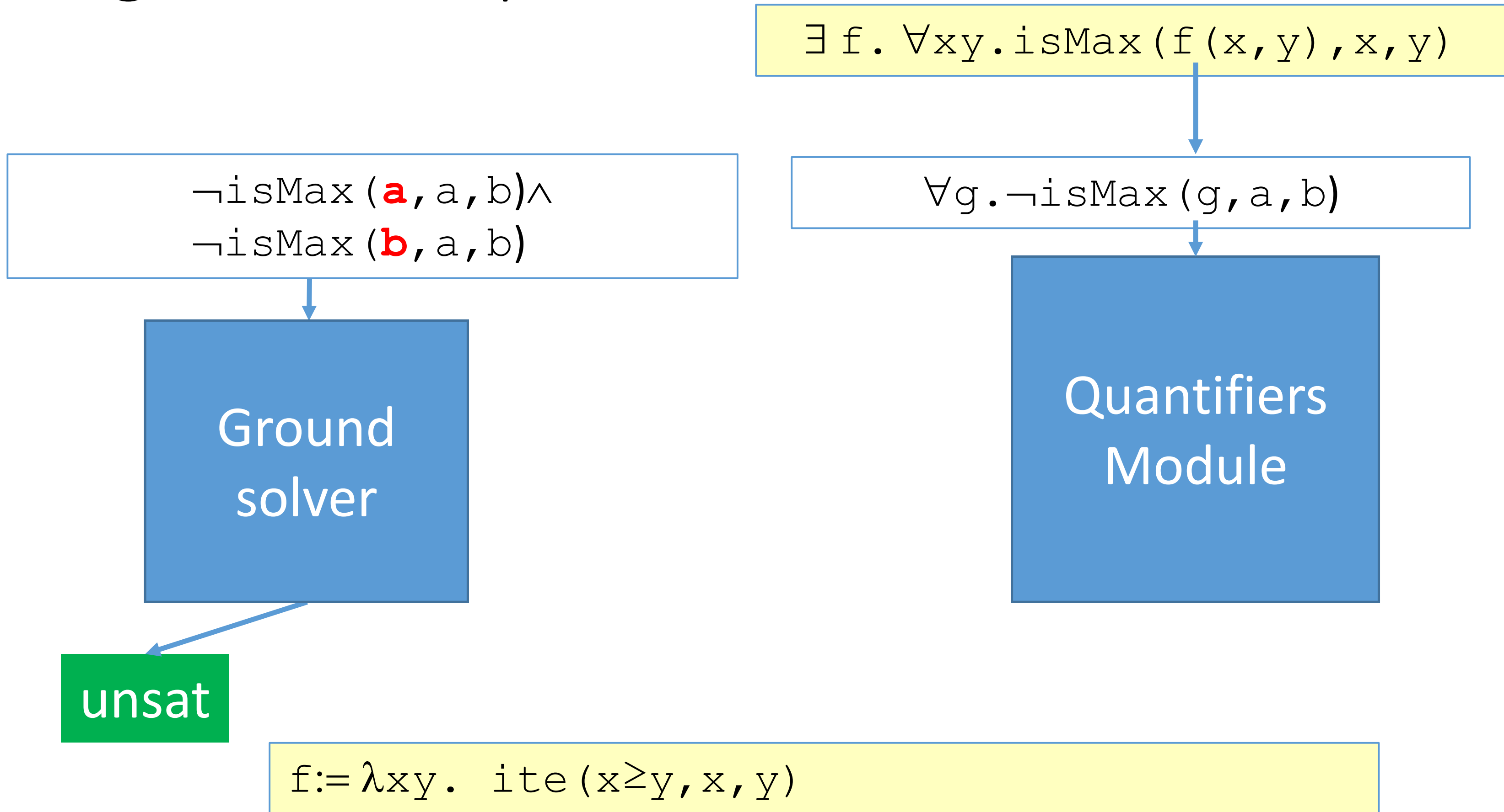
unsat $\Rightarrow$ $\forall g.\neg isMax(g,a,b)$ is unsatisfable
by instances a/g, b/g,
implies original synthesis conjecture has a solution

Solving Max Example



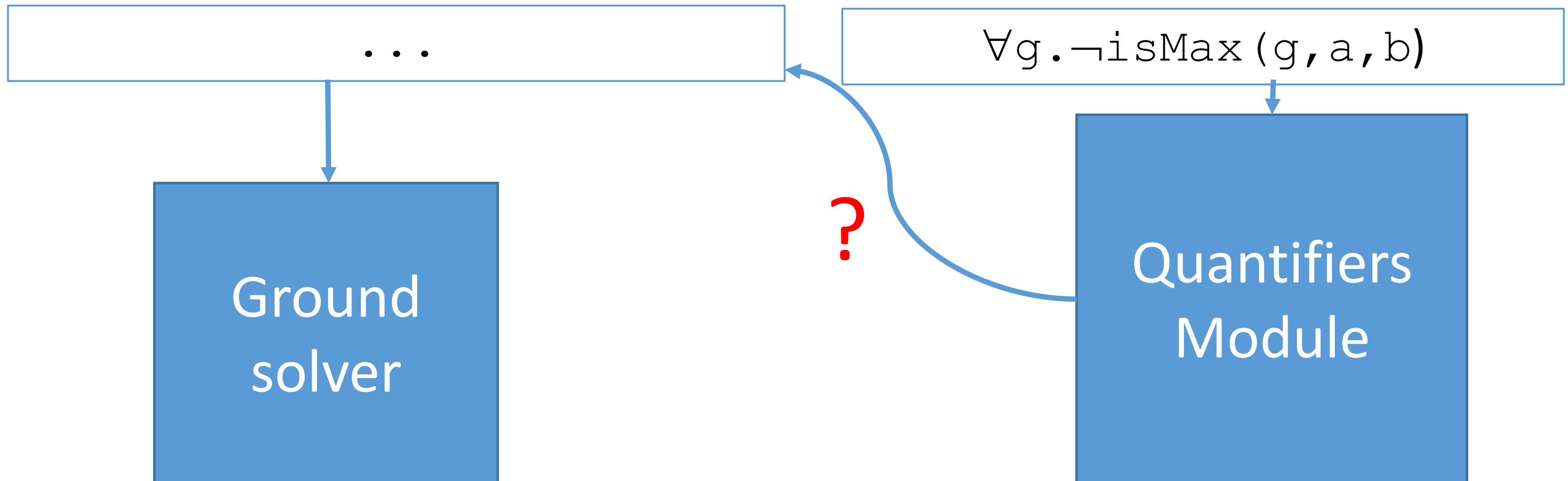
⇒ Extract solution from unsatisfiable core of instantiations a/g, b/g

Solving Max Example

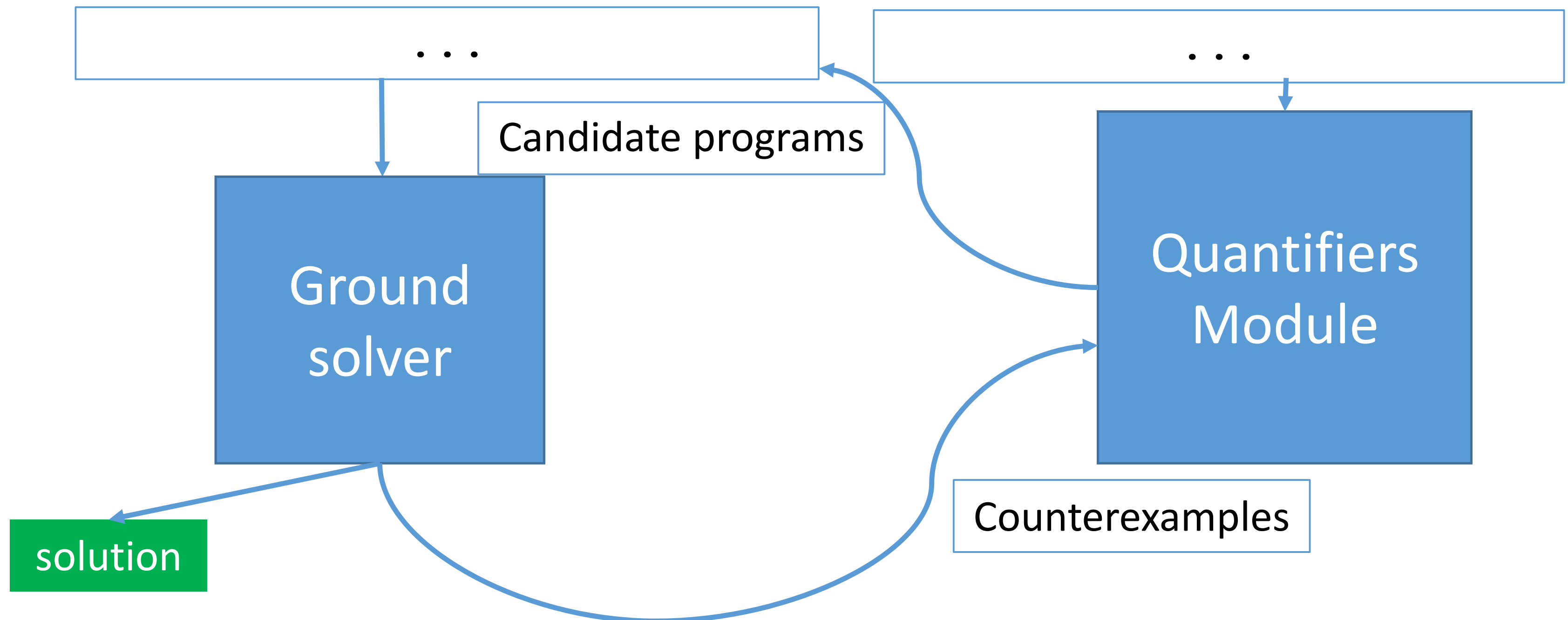


⇒ Desired function, after simplification

How do we Choose Relevant Instances?



Counterexample-Guided Quantifier Instantiation



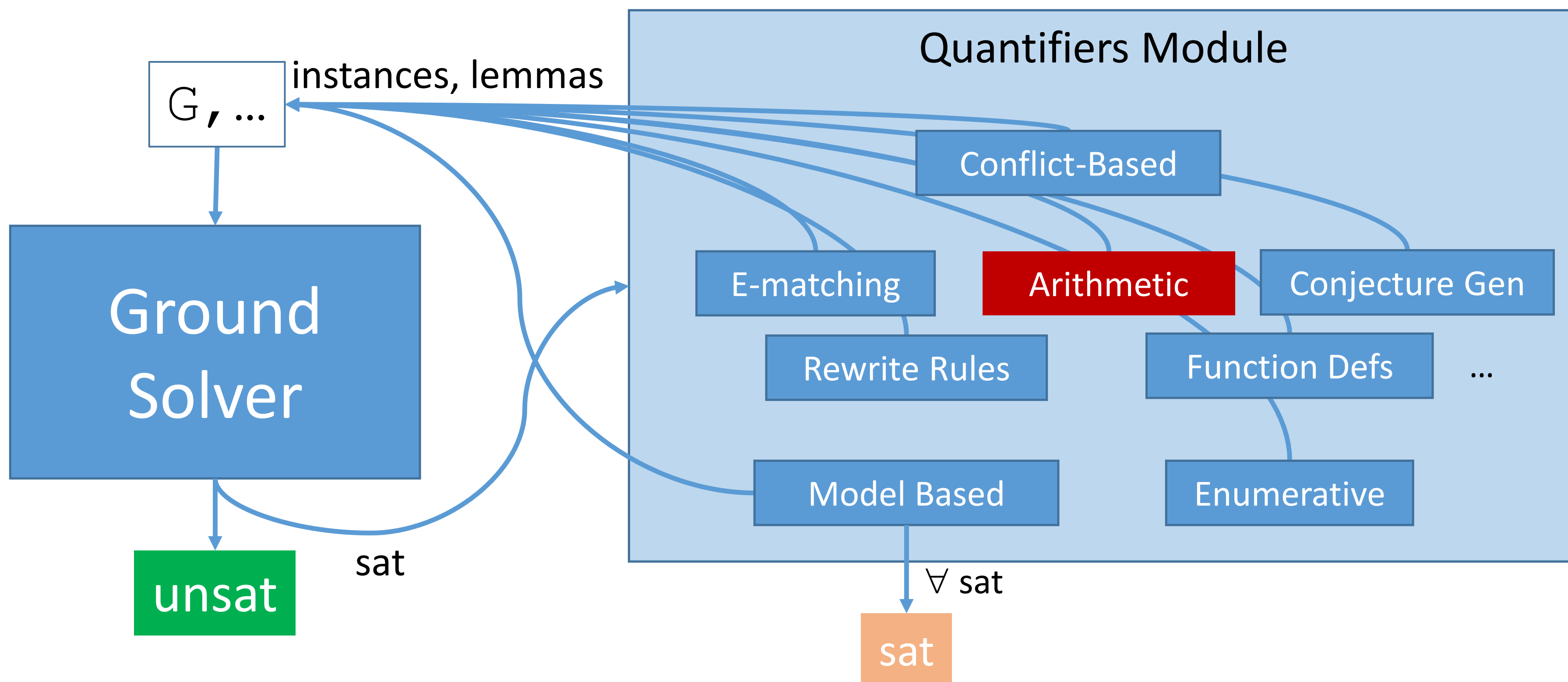
- Instances chosen using *counterexample-guided quantifier instantiation*

  $\Rightarrow$ Follows counterexample-guided inductive synthesis (CEGIS) approach

In work under submission, we provide framework where such CEGIS choices are **complete** for linear arithmetic

Much better scalability than quantifier elimination approaches

Quantifiers Module of CVC4



Specialized technique for quantified arithmetic

arXiv:1510.02642

An Instantiation-Based Approach for Solving Quantified Linear Arithmetic

Andrew Reynolds, Tim King, Viktor Kuncak

CVC4 in Sygus Competition 2015

NSF Expedition on Computer Augmented Programm Engineering (ExCAPE)
led by Rajeev Alur, involves UPenn, MIT, Berkeley, Rice, Illinois, Maryland, UCLA, Michigan
organizes a competition of software synthesis tools

https://excape.cis.upenn.edu/

Our technique Won General and LIA tracks (= 2/3 tracks) in competition

LIA Track	73		
Solver	#solved	total-expr-size	average-expr-size
CVC4-1.5-syguscomp2015-v4	70	43726	624.66
AlchemistCSDT	47	6658	141.66
Alchemist CS	33	866	26.24

In LIA track, solved 70/73 benchmarks, 60 of these in <1 second

- Nearest competitor **AlchemistCSDT** solved 47/73 in a timeout of 1 hour

Max example : Sygus Comp 2015

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cvc4+si	0.0	0.0	0.0	0.0	0.1	0.1	0.2	0.3	0.6	1.0	1.9	3.2	5.3	6.5
AlchemistCSDT	0.2	0.6	1.5	6.4	20.8	132.8	877.9	–	–	–	–	–	–	–
AlchemistCS	0.0	3.7	–	–	–	–	–	–	–	–	–	–	–	–

- Outperforms existing approaches by an order of magnitude or more

⇒Our approach is efficient for synthesizing non-recursive functions that are defined by cases

Implementation available in the main branch of CVC4 SMT solver:

http://cvc4.cs.nyu.edu/web/

Conclusions: Synthesis and SMT Solvers

Leon system for verifying and synthesizing Scala programs

http://leon.epfl.ch

SMT solvers are essential for synthesis and verification

Given support for quantifiers, SMT solver can perform synthesis on its own!

Key challenge: efficient techniques to instantiate quantifiers

- CVC4 solution is state of the art for linear arithmetic: complete and fast

arXiv:1510.02642
An Instantiation-Based Approach for Solving Quantified Linear Arithmetic
Andrew Reynolds, Tim King, Viktor Kuncak