Programming SMT solvers

Haniel Barbosa

Compilers Laboratory

UFMG

EVCOMP 2020
2020–02–20, Belo Horizonte, Brazil
A Truism

Software has become critical to modern life

- Communication (internet, voice, video, ...)
- Transportation (air traffic control, avionics, cars, ...)
- Health Care (patient monitoring, device control, ...)
- Finance (automatic trading, banking, ...)
- Defense (intelligence, weapons control, ...)
- Manufacturing (precision milling, assembly, ...)
- Process Control (oil, gas, water, ...)
- ...
Embedded Software

Software is now embedded everywhere
Embedded Software

Software is now embedded everywhere
Some of it is critical

Failing software costs money and life!
Failing Software Costs Money

- Expensive recalls products with embedded software
- Lawsuits for loss of life or property damage
  - Car crashes (e.g., Toyota Camry 2005)
- Thousands of dollars for each minute of down-time
  - (e.g., Denver Airport Luggage Handling System)
- Huge losses of monetary and intellectual investment
  - Rocket boost failure (e.g., Ariane 5)
- Business failures associated with buggy software
  - (e.g., Ashton-Tate dBase)
Failing Software Costs Lives

- Potential problems are obvious:
  - Software used to control nuclear power plants
  - Air-traffic control systems
  - Spacecraft launch vehicle control
  - Embedded software in cars

- A well-known and tragic example:
  Therac-25 radiation machine failures
The Peculiarity of Software Systems

Software seems particularly prone to faults

Tiny faults can have catastrophic consequences

- Ariane 5
- Mars Climate Orbiter, Mars Sojourner
- Pentium-Bug
- ...

Rare bugs can occur

- avg. lifetime of a passenger plane: 30 years
- avg. lifetime of a car: < 10 years, but already > 1.2B cars in 2014

Logic and implementation errors represent security exploits

- (too many to mention)
How to Ensure Software Correctness?

A Central Strategy: **Testing**
(others: SW processes, reviews, libraries, ...)

Testing against inherent SW errors ("bugs")
- Design test configurations that hopefully are representative and
- ensure that the system behaves as intended on them

Testing against external faults
- Inject faults (memory, communication) by simulation or radiation
Limitations of Testing

▷ Testing can show the presence of errors, but not their absence
(exhaustive testing viable only for trivial systems)

▷ Representativeness of test cases/injected faults is subjective
How to test for the unexpected? Rare cases?

▷ Testing is labor intensive, hence expensive
For Your Consideration: Formal Methods

Rigorous techniques and tools for the development and analysis of computational (hardware/software) systems

- Complement other analysis and design methods
- Help find bugs in code and specification
- Reduce development, and testing, cost
- Ensure certain properties of the formal system model
- Should be highly automated since it is based on mathematics and symbolic logic (formal)
For Your Consideration: Formal Methods

Rigorous techniques and tools for the development and analysis of computational (hardware/software) systems

▷ Complement other analysis and design methods
▷ Help find bugs in code and specification
▷ Reduce development, and testing, cost
▷ Ensure certain properties of the formal system model
▷ Should be highly automated since it is based on mathematics and symbolic logic (formal)

Main artifacts
▷ (1) A formal specification of the system requirements
▷ (2) A formal execution model of the system implementation

Use tools to mechanically verify that (2) satisfies (1)
Automated reasoning

- Formal Verification
- Program Analysis
- Automatic Testing
- Program Synthesis
Research statement

My goal is to **increase the applicability** of formal methods via:

**Automated reasoning**

- The Computer Science and Mathematical Logic field concerned with
  - *deduction* (premises entail Truth of conclusion)
  - *models* (witness of Truth)
  - *proofs* (convincing argument of Truth)

- We devise algorithms to solve problems stated in formal languages
My goal is to increase the applicability of formal methods via:

Automated reasoning

- The Computer Science and Mathematical Logic field concerned with:
  - deduction (premises entail Truth of conclusion)
  - models (witness of Truth)
  - proofs (convincing argument of Truth)
- We devise algorithms to solve problems stated in formal languages

Satisfiability modulo theories (SMT)

- Sweetspot between expressive logics and efficient decision procedures
Automated reasoning

- Formal Verification
- Program Analysis
- Automatic Testing
- Program Synthesis
Formal Verification

Program Analysis

Automatic Testing

Program Synthesis

SMT Solvers
Propositional formulas in CNF:

\[
C ::= p \mid \neg p \mid C \lor C
\]

\[
\varphi ::= C \mid \varphi \land \varphi
\]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \varphi \) to \( \{ \top, \bot \} \) such that \( M(\varphi) = \top \) (or \( M \models \varphi \)).
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[
C ::= p \mid \neg p \mid C \lor C
\]

\[
\phi ::= C \mid \phi \land \phi
\]

Given a formula \( \phi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \phi \) to \( \{\top, \bot\} \) such that \( M(\phi) = \top \) (or \( M \models \phi \)).

**Example**

Is \( \phi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \) satisfiable?
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[ C ::= p \mid \neg p \mid C \lor C \]

\[ \varphi ::= C \mid \varphi \land \varphi \]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( \mathcal{M} \) mapping every proposition \( \varphi \) to \( \{\top, \bot\} \) such that \( \mathcal{M}(\varphi) = \top \) (or \( \mathcal{M} \models \varphi \)).

**Example**

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \) satisfiable? **Yes**

\[ \mathcal{M}(p) = \top, \mathcal{M}(q) = \top, \mathcal{M}(r) = \bot \quad \Rightarrow \quad \mathcal{M}(\varphi) = \top \]
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[
C ::= p \mid \neg p \mid C \lor C \\
\varphi ::= C \mid \varphi \land \varphi
\]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( \mathcal{M} \) mapping every proposition \( \varphi \) to \( \{ \top, \bot \} \) such that \( \mathcal{M}(\varphi) = \top \) (or \( \mathcal{M} \models \varphi \)).

**Example**

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \land (r \lor \neg q) \) satisfiable?
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[ C ::= p | \neg p | C \lor C \]
\[ \varphi ::= C | \varphi \land \varphi \]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \varphi \) to \( \{\top, \bot\} \) such that \( M(\varphi) = \top \) (or \( M \models \varphi \)).

Example

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \land (r \lor \neg q) \) satisfiable? \( \text{No} \)

No combination of valuations for these propositions such that \( \varphi \) is \( \top \).
SAT solving

- SAT is NP-complete
- Nevertheless tractable in practice by modern SAT solvers, based on *conflict driven clause learning* (CDCL)
  - mid '90s: formulas solvable with thousands of variables and clauses
  - now: formulas solvable with millions of variables and clauses
SAT solving

- SAT is NP-complete
- Nevertheless tractable in practice by modern SAT solvers, based on \textit{conflict driven clause learning} (CDCL)
  - mid '90s: formulas solvable with thousands of variables and clauses
  - now: formulas solvable with \textit{millions} of variables and clauses

\begin{itemize}
  \item[] \textbf{DPLL} vs \textbf{CDCL}
\end{itemize}

- CDCL combines \textit{model search} and \textit{proof search}
  - Smart ideas + clever engineering “tricks”
SAT solving

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

- Limmat (2002)
- Zchaff (2002)
- Berkmin (2002)
- Forklift (2003)
- Siege (2003)
- SatELite (2005)
- Minisat 2 (2006)
- Picosat (2007)
- Rsat (2007)
- Minisat 2.1 (2008)
- Precosat (2009)
- Glucose (2009)
- Clasp (2009)
- Cryptominisat (2010)
- Lingeling (2010)
- Minisat 2.2 (2010)
- Glucose 2 (2011)
- GlueMiniSat (2011)
- Contrasat (2011)
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:
\[ t ::= x \mid f(t, \ldots, t) \]
\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an \textit{interpretation} to all terms and predicates such that \( M \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo \textit{equality} and \textit{arithmetic}?
Satisfiability Modulo Theories (SMT)

First-order formulas

\[ t ::= x \mid f(t, \ldots, t) \]

in CNF:

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \, \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an \textit{interpretation} to all terms and predicates such that \( M \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo \textit{equality} and \textit{arithmetic}?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \neq f(0)) \lor x_3 + x_1 > x_3 + 1 \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x | f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an interpretation to all terms and predicates such that \( M \models T_1, \ldots, T_n \ \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\equiv f(0)) \lor x_3 + x_1 > x_3 + 1) \]

\[ \text{LIA} \quad \text{EUF} \quad \text{LIA} \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x | f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \neq f(0)) \lor x_3 + x_1 > x_3 + 1) \]

\[ \varphi \models_{\text{LIA}} x_1 \simeq 0 \]
First-order formulas
\[ t ::= x \mid f(t, \ldots, t) \]
in CNF:
\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that
\[ \mathcal{M} \models T_1, \ldots, T_n \varphi \]

**Example**

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = \underbrace{(x_1 \geq 0) \land (x_1 < 1)}_{\text{LIA}} \land \underbrace{(f(x_1) \not\equiv f(0))}_{\text{EUF}} \lor \underbrace{x_3 + x_1 > x_3 + 1}_{\text{LIA}} \]

\[ \varphi \models_{\text{LIA}} x_1 \equiv 0 \]
\[ \models_{\text{EUF}} f(x_1) \equiv f(0) \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x | f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models T_1, \ldots, T_n \varphi \)

**Example**

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\equiv f(0)) \lor x_3 + x_1 > x_3 + 1 \]

\[ \begin{align*}
\varphi &\models_{\text{LIA}} x_1 \approx 0 \\
x_1 \approx 0 &\models_{\text{EUF}} f(x_1) \approx f(0) \\
x_1 \approx 0 &\models_{\text{LIA}} x_3 + x_1 \not\approx x_3 + 1
\end{align*} \]
Satisfiability Modulo Theories (SMT)

First-order formulas

\[ t ::= x | f(t, \ldots, t) \]

in CNF:

\[ \varphi ::= p(t, \ldots, t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\equiv f(0)) \lor (x_3 + x_1 > x_3 + 1) \]

\[ \models_{\text{LIA}} x_1 \equiv 0 \]
\[ \models_{\text{EUF}} f(x_1) \equiv f(0) \]
\[ \models_{\text{LIA}} x_3 + x_1 \not> x_3 + 1 \]

Therefore \( \models_{\text{EUF} \cup \text{LIA}} \neg \varphi \)
SMT solving

- Decidability depends on the theories being used

- Efficient decision procedures
  - Equality and uninterpreted functions (Congruence Closure)
  - Linear arithmetic (Simplex)
  - Bit-vectors (Bit-blasting)
  - Combination of theories (Nelson-Oppen)
  - ...

- Boolean search leverages SAT solvers

- Users may define their own theories
  - New operators as uninterpreted functions + Axioms
CDCL($\mathcal{T}$) architecture

- Rewriter simplifies terms
  \[ x + 0 \to x \quad a \not\equiv a \to \bot \quad (\text{str.replace} \ x \ (\text{str.++} \ x \ x \ y)) \to x \]
- SAT solver enumerates models for Boolean skeleton of formula
- Theory solvers check consistency in the theory
- Instantiation module selects relevant instances
Many SMT solvers around

- **CVC4**
  - Primarily developed at Stanford University and The University of Iowa
  - Open source, C++
  - Available at https://github.com/CVC4/CVC4

- **Z3**
  - Primarily developed at Microsoft Research
  - Open source, C++
  - Available at https://github.com/Z3Prover/z3

- Other noteworthy SMT solvers:
  - veriT
  - Yices
  - MathSAT
  - Boolector
  - ...

Programming SMT solvers
How to profit from SMT solvers?

Determine for your favorite application:

- How to encode (parts of) my problem in SMT? Which theories to use?
- Leverage existing solvers
  - Standard input language
  - APIs available in several languages (Python, OCaml, C++, ...)
  - Continuously maintained (CVC4, Z3, ...)
- If need be, how to extend the SMT solver?
  - New formal calculus
  - New algorithm
  - Implementation techniques

Fun!!!
How to profit from SMT solvers?

Determine for your favorite application:

▷ How to encode (parts of) my problem in SMT? Which theories to use?
How to profit from SMT solvers?

Determine for your favorite application:

▷ How to encode (parts of) my problem in SMT? Which theories to use?

▷ Leverage existing solvers
  ▶ Standard input language
  ▶ APIs available in several languages (Python, OCaml, C++, ...)
  ▶ Continuously maintained (CVC4, Z3, ...)

Fun!!!
How to profit from SMT solvers?

Determine for your favorite application:

▷ How to encode (parts of) my problem in SMT? Which theories to use?

▷ Leverage existing solvers
  ► Standard input language
  ► APIs available in several languages (Python, OCaml, C++, ...)
  ► Continuously maintained (CVC4, Z3, ...)

▷ If need be, how to extend the SMT solver?
  ► New formal calculus
  ► New algorithm
  ► Implementation techniques

Fun!!!
How to *program* with SMT solvers?

- Common language for SMT solvers: SMT-LIB
How to program with SMT solvers?

- Common language for SMT solvers: SMT-LIB

- PySMT (https://github.com/pysmt/pysmt): a solver-agnostic Python wrapper for SMT

- We will now see how to define SMT problems with PySMT and how to solve some interesting problems via SMT

Many thanks to Aina Niemetz and Mathias Preiner for their SAT/SMT summer school practical session materials
PySMT

PySMT is available at https://github.com/pysmt/pysmt

**Installing locally**

- pip install pysmt
- pysmt-install --btor  
  (install Boolector)
- pysmt-install --z3  
  (install Z3)
- pysmt-install --cvc4  
  (install CVC4)
- pysmt-install --msat  
  (install MathSat)
Include Shortcuts and Typing from PySMT

from pysmt.shortcuts import *
from pysmt.typing import *

- Shortcuts defines wrappers for most commonly used functions
  https://pysmt.readthedocs.io/en/latest/api_ref.html#module-pysmt.shortcuts

- Typing defines SMT types (sorts)
  https://pysmt.readthedocs.io/en/latest/api_ref.html#module-pysmt.typing

**Note:** You can also import functions individually

from pysmt.shortcuts import Symbol
from pysmt.typing import INT
PySMT — Shortcuts

▷ Symbol: create variables and (symbolic) constants

a = Symbol("a")  # By default sort BOOL
x = Symbol("x", INT)  # Integer sort
b = Symbol("b", BVType(32))  # Bit-vector sort of size 32

▷ TRUE, FALSE, Bool, Int, BV: Theory constants

y = Int(0)
z = BV(3, 4)  # Bit-vector value 3, size 4

▷ And, Or, Not, Implies, Iff: Boolean operators

And(LE(y, x), GE(Int(10), x))  # y ≤ x ∧ 10 ≥ x
PySMT — Shortcuts

▷ Equals, NotEquals, AllDifferent  
  LE, GE, LT, GT  
  (Dis)Equality  
  Inequality

▷ Minus, Plus, Times, Div  
  Note: not for bit-vectorss!  
  Arithmetic operators

▷ BVAdd, BVSub, BVMul,  
  BVUDiv, BVSDiv, BVNeg  
  Arithmetic BV operators

▷ BVNot, BVAnd, BVOOr, BVXor  
  BVLShl, BVLShr, BVAShr  
  Bit-wise operators

▷ Ite  
  If-then-else
PySMT — Typing

- **BOOL**: Boolean sort
  - `a = Symbol("a")` # By default sort BOOL
  - `a = Symbol("x", BOOL)`
  - `True()`, `False()` # Boolean values

- **INT**: Integer sort
  - `x = Symbol("x", INT)` # Integer sort
  - `Int(2)` # Integer value

- **REAL**: Real sort
  - `x = Symbol("x", REAL)` # Real sort
  - `Real(2.5)` # Real value
  - `Real(3,2)` # Real value \(\frac{3}{2}\)

- **Uninterpreted sorts**:
  - `u = Type("U")`
PySMT — Typing

- **FunctionType**(return_type, [arg_types]): Functional sort argument sorts to return sort
  
  \[
  u = \text{Type}("U") \\
  fu = \text{FunctionType}(u, [\text{INT}]) \quad \# \text{INT} \rightarrow U
  \]

- **BVType**(size): Bit-vector type of given size
  
  \[
  b = \text{Symbol}("b", \text{BVType}(32)) \quad \# \text{Bit-vector sort of size 32} \\
  \text{BV}(3,32) \quad \# \text{Bit-vector value}
  \]

- **ArrayType**(index_type, element_type): Array sort
  
  \[
  \text{ArrayType(\text{INT},\text{REAL})}
  \]
btor = Solver(name=btor)  # Boolector

cvc4 = Solver(name=cvc4)  # CVC4

msat = Solver(name=msat)  # MathSAT

yices = Solver(name=yices)  # Yices

z3 = Solver(name=z3)  # Z3

cvc4.add_assertion(...)

########################

with Solver(name=cvc4) as solver:
    solver.add_assertion(...)
from pysmt.shortcuts import *
from pysmt.typing import *

sort_u = Type("U")
sort_fu = FunctionType(sort_u, [INT])

f = Symbol("f", sort_fu)
x1 = Symbol("x1", INT)
x3 = Symbol("x3", INT)

with Solver() as solver:
    solver.add_assertion(And(GE(x1,Int(0)),(LT(x1,Int(0)))))
solver.add_assertion(Or(Equals(Function(f, [x1]), \
    Function(f, [Int(0)])), GT(Plus(x3,x1),Plus(x3,Int(1))))))

    if solver.solve():
        print("sat")
    else:
        print("unsat")
Exercises — Branchless abs(x)

Absolute value abs(x)

\[ x < 0 \ ? \ -x : x \]

Prove that the branchless version of function abs(x) from page 18 of Hacker’s delight\(^1\) are correct.

Alternatives of branchless abs(x) (32 bits)

\[ y := x \gg s 31 \quad \text{(arithmetic right shift, BVAShr in PySMT)} \]

Alternative 1: \( (x \oplus y) - y \)

Alternative 2: \( (x + y) \oplus y \)

Alternative 3: \( x - ((2 \cdot x) \& y) \)

\(^1\)http://www.hackersdelight.org/basics2.pdf
How many combinations of appetizers exist that are exactly worth $15.05? What appetizer combinations are possible?

Note: You can pick more than one appetizer of a kind (5x french fries, ...)
Exercises — Sudoku

Fill the blanks (marked as **STUB**) in `sudoku.py`.

Sudoku rules for 3x3

- Each of the 3x3 squares contains numbers 1-9
- Each number can only appear once in each row, column, and square

**Node:** `sudoku.py` should handle 2x2, 4x4, ...
Exercises — Pseudorandom Number Generator

Given a function `rand()` that generates pseudorandom numbers based on the following linear congruential generator (LCG) algorithm\(^2\).

\[
X_{i+1} := (1019357 \cdot X_i + 30129) \mod (117)
\]

▷ What is the maximum number of consecutive iterations of `rand() % 47` that produce the number 42?

▷ What is the starting seed \(X_0\)?

Fill the blanks (marked as STUB) in `lcg.py`.

C code example

```c
uint32_t rand(uint32_t x) { return (1019357 * x + 30129) % (1 << 17); }
uint32_t x, x0, n = 0;
x = x0 = ?;
while((x = rand(x)) % 47 == 42) { n++; }
```

\(^2\)https://en.wikipedia.org/wiki/Linear_congruential_generator
Exercises — Bounded Model Checking

Fill the blanks (marked as **STUB**) in `bmc.py`.
Check if safety property $P$ holds for 10 iterations.

- Unroll the loop 10 times or until property $P$ is violated
- Check for each iteration if property $P$ holds

### C Code

```c
int main () {
    bool turn; // input
    uint32_t a = 0, b = 0; // states
    for (;;) {
        turn = read_bool ();
        assert (a != 3 || b != 3); // property $P$
        if (turn) a = a + 1; // next(a)
        else b = b + 1; // next(b)
    }
}
```

### Unroll

- $a_0 = 0 \land b_0 = 0$
  ...check if $P$ holds for $a_0, b_0$
- $a_1 = \text{next}(a_0) \land b_1 = \text{next}(b_0)$
  ...check if $P$ holds for $a_1, b_1$
- $a_2 = \text{next}(a_1) \land b_2 = \text{next}(b_1)$
  ...check if $P$ holds for $a_2, b_2$
  ...

Programming SMT solvers
Programming SMT solvers

Haniel Barbosa

EVCOMP 2020
2020–02–20, Belo Horizonte, Brazil