Some Elements on Communicating with an SMT

Burkhart Wolff

Projet Genie Logiciel (Room Planner) 2023

Theoretical Background

- In Principle, an “Automated Theorem Prover” ATP
  is a system that automatically attempts to find a proof
  for a statement like

$$\Gamma \vdash_{\Theta} \phi$$

- Where Γ are the set/list of assumptions
- and Θ is the logic in which the statement should hold (PL,EL, FOL,HOL,…)
- and φ is a formula (proposition) that should hold.
- An answer can of an ATP can be: yes, I don’t know (timeout), or a counterexample. We are particularly interested in these.

Recall GLA : Foundations: Proof Systems

- An Inference System (or *Logical Calculus*) allows to infer formulas from a set of *elementary facts* (axioms) and inferred facts by rules:

$$\frac{A_1 \quad \ldots \quad A_n}{A_{n+1}}$$

- "from the *assumptions* A_1 to A_n, you can infer the *conclusion* A_{n+1}." A rule with n=0 is an *elementary fact*. Variables occurring in the formulas A_n can be arbitrarily substituted.
- Assumptions and conclusions are terms in a logic containing variables

Recall GLA : Foundations: Proof Systems

- An Inference System for the equality operator (or "Equational Logic") looks like this:

$$\frac{}{x = x} \qquad \frac{x = y}{y = x} \qquad \frac{x = y \quad y = z}{x = z}$$

$$\frac{x = y \quad P(x)}{P(y)}$$

- where the first rule "reflexivity" is an elementary fact.

Recall GLA : Foundations: Proof Systems

The variables in an inference rule can be replaced by a substitution. The substituted inference rule is called an instance (of this rule).

$$\frac{x = y \quad y = z}{x = z}$$

{x↦1+2, y↦2+1, z↦3}

$$\frac{1+2 = 2+1 \quad 2+1 = 3}{1+2 = 3}$$

{x↦1+2, y↦a, z↦3}

$$\frac{1+2 = a \quad a = 3}{1+2 = 3}$$

{x↦τ*5, y↦5*τ}

$$\frac{\tau * 5 = 5 * \tau \quad 5 * \tau = z}{\tau * 5 = z}$$

Recall GLA : Foundations: Proof Systems

- A *Formal Proof* (or : *Derivation*)

  is a tree with rule instances as nodes

$$\dfrac{\dfrac{\dfrac{f(a,b)=a}{a=f(a,b)} \quad \dfrac{f(a,b)=a \quad f(f(a,b),b)=c}{f(a,b)=c}}{a=c} \quad \dfrac{}{g(a)=g(a)}}{g(a)=g(c)}$$

- The non-elementary facts at the leaves are the *global assumptions* (here $f(a,b) = a$ and $f(f(a,b),b) = c$).

Recall GLA : Foundations: Proof Systems

- As a short-cut, we also write for a derivation:

$$\{A_1, \ldots, A_n\} \vdash A_{n+1}$$

- ... or generally speaking: from global assumptions A to a theorem (in theory E) φ:

$$\Gamma \vdash_E \phi$$

- This is what theorems are: derivable facts from assumptions in a certain logical system ...

Recall Foundations: Proof Systems

- Recall: Some basic notions for statement: $$\Gamma \vdash_{\Theta} \psi$$

  - A formula ψ is *valid* if it evaluates to true for all substitutions of the variables by values
  - A logic (or inference system) Θ is decidable iff there is an algorithm that can infer for any formula ψ and any set of assumptions Γ that it is valid (provided the algorithm is given sufficient ressources).

  - Fact: Propositional logic (PL) is decidable, first-order logic (FOL = PL + quantifiers) is undecidable

Recall Foundations: Proof Systems

- Recall: Some basic notions for statement: $\Gamma \vdash_{\Theta} \psi$

  - A statement is *satisfiable* (sat) if for all variables in Γ et ψ there exists a value that lets the statement evaluate to true
  - For some forms of PL formulas, decidability and satisfiability are interlinked:

$$\Gamma \vdash_{\Theta} \exists\, x.\ P(x)$$

can be represented by the fresh uninterpreted function (skolem) symbol c

$$\{\text{const } c;\ \Gamma;\ P(c)\}\ \text{sat}$$

The Z3 Theorem Prover

- Z3 is an automated theorem prover developed by MicroSoft, but distributed for non-commercial use for free.
- Documentation and even the sources can be found here: https://microsoft.github.io/z3guide/docs/
- Z3 belongs to a class of ATP's called SMT (satisfyability modulo theories), but that's not important
- It supports a number of fragments of PL plus EL.
- It has a command line-interface and an input format for Γ and ϕ

The Z3 Theorem Prover

- Main Reference: https://github.com/Z3Prover/z3

- Programming Manual : https://z3prover.github.io/papers/programmingz3.html

- To start with, it is useful to consider the Introduction and the Documentation, in particular see "Basic Commands" in https://microsoft.github.io/z3guide/docs/logic/basiccommands

- The manual also offers "playgrounds" where one can directly experiment

- It has a command-line interface (described in the SMTLIB2 Format) and a counter-example generator (generating "models")

The Z3 Theorem Prover

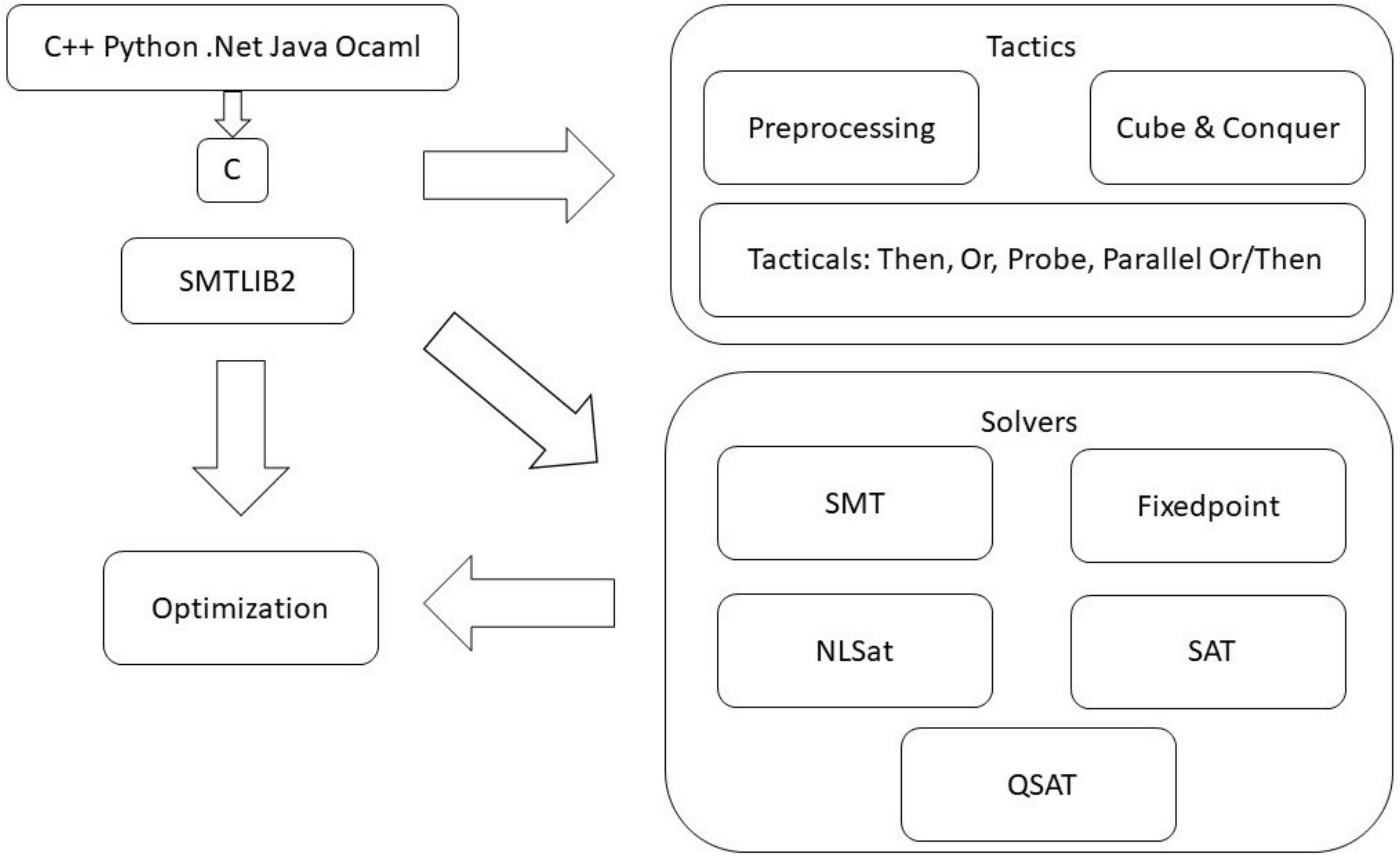
- The manual also offers “playgrounds” where one can directly experiment via editing the window:

```
(declare-const p Bool)
(declare-const q Bool)
(declare-const r Bool)
(define-fun conjecture () Bool
        (=> (and (=> p q) (=> q r))
                (=> p r)))
(assert (not conjecture))
(check-sat)
```

Run

Planning with z3

- Z3 takes as input simple-sorted formulas that may contain symbols with pre-defined meanings defined by a *theory*.
- The architecture includes:
  - Various frontends,
  - The ASCII exchange format SMTLIB2
  - Preprocessing and Special Tactics



Planning with z3

- Z3 takes as input simple-sorted formulas that may contain symbols with pre-defined meanings defined by a *theory*.
- A theory may contain a global theory context
  - such as QF_LIA (quantifier-free linear integer arithmetic), EULA , AUFLIA, Lists, Bv, …
  - sorts,
  - uninterpreted functions and definitions
  - assertions for the logical context
  - and finally an analysis goal, checking satisfiability or model-construction

```
sorts:
  Nat$ = nat
  Num$ = num
```

```
(declare-sort Nat$ 0)
(declare-sort Num$ 0)
(declare-sort Num_num_fun$ 0)
(declare-sort Num_bool_fun$ 0)
(declare-fun x$ () Int)
(declare-fun r1$ () Nat$)
(declare-fun r2$ () Nat$)
(declare-fun r3$ () Nat$)
(declare-fun r4$ () Nat$)
(declare-fun t1$ () Nat$)
```

```
functions:
 one$ = num.One
 suc$ = Suc
 less$ = (<)
 one$a = 1
 plus$ = (+)
```

```
(assert (! (forall ((?v0 Int)) (= (* 1 ?v0) ?v0)) :n
(assert (! (forall ((?v0 Nat$)) (= (times$ (numeral$
```

```
(check-sat)
(get-model)
```

Planning with z3

- Z3 takes as input simple-sorted formulas that may contain symbols with pre-defined meanings defined by a *theory*.
- Z3 allows universal quantification in assumptions Γ, but no existential quantifications. However, Z3 supports uninterpreted functions and the reduction to a satisfiability problem (in fact, this is the preferred format)
- Z3 allows to generate models (examples resp. counter-examples) for its uninterpreted function symbols …