Satisfiability Modulo Theories
Lezione 1 - Overview

(slides revision: Saturday 14th March, 2015, 11:46)

Roberto Bruttomesso

Seminario di Logica Matematica
(Corso Prof. Silvio Ghilardi)

20 Ottobre 2011

FAQ

Le slides seguono la dispensa (in fase di scrittura !)

"Satisfiability Modulo Theories"

e sono entrambi disponibili da
`http://www.oprover.org/roberto/teaching/smt`, dove trovate anche i puntatori ai tool e agli esempi usati qui

Per chi cercasse un libro su questi argomenti, forse quello che si avvicina di piu' e'

`Decision Procedures - An Algorithmic Point of View`

(`www.decision-procedures.org`)

Per ricevimento la mia stanza e' la S206, II piano, via Comelico, previa richiesta via email `roberto.bruttomesso@gmail.com`

Outline

Intro

Satisfiability Modulo Theories, SMT, studies **practical methods** to **solve** logical formulæ

These formulæ are defined/interpreted modulo a **background theory**, such as Linear Real Arithmetic ($\mathcal{LRA}$), Arrays ($\mathcal{A}$), Bit-Vectors ($\mathcal{BV}$), etc.

For instance, we want to determine the satisfiability modulo $\mathcal{LRA}$ of

$$(x + y \leq 0) \wedge (x = 0) \wedge (\neg a \vee (x = 1) \vee (y \geq 0)) \tag{1}$$

where x, y are arithmetic variables, while a is a Boolean variable. Intuitively, (1) is satisfiable iff we can find values for x and y in $\mathbb{R}$ and for a in $\mathbb{B}$ such that it evaluates to $\top$

Definitions (syntax)

In SMT a theory $\mathcal{T}$ is defined over a **signature** Σ, which is a set of function and predicate symbols such as $\{0, 1, \ldots, +, -, \ldots, \leq\}$. The equality symbol $=$ is assumed to be included in every signature.

Variables and function symbols in Σ can be used to build **theory-terms** ($\mathcal{T}$-term): a $\mathcal{T}$-term is either a variable or, recursively, an application of a function symbol in Σ to terms

Predicate symbols in Σ can be used to build **theory-atoms** ($\mathcal{T}$-atom): a $\mathcal{T}$-atom is the application of a predicate symbol in Σ to $\mathcal{T}$-terms

A **theory-literal** ($\mathcal{T}$-literal) is either a $\mathcal{T}$-atom or its negation

A formula is any Boolean combination of $\mathcal{T}$-atoms and Boolean variables

Definitions (semantic)

In SMT the **interpretation** of the symbols in Σ is fixed, and it corresponds to the usual semantic of the operators. For instance, in $\mathcal{LIA}$ (Linear Integer Arithmetic):

- numerals are mapped to the corresponding value in $\mathbb{Z}$
- $+$ is interpreted as the function

$$(0,0) \mapsto 0$$
$$(0,1) \mapsto 1$$
$$\ldots$$

The only unspecified entities are variables, for which we have to build an **assignment**, a mapping from variables to concrete values in $\mathbb{Z}$

Now everything is specified and we can evaluate $\mathcal{T}$-terms, $\mathcal{T}$-atoms and formulæ. For instance, the $\mathcal{T}$-atom

$$(x + y \leq 0) \wedge (x = 0) \wedge (\neg a \vee (x = 1) \vee (y \geq 0))$$

evaluates to $\top$ under the assignment $\{x \mapsto 0, y \mapsto 0, a \mapsto \bot\}$, and it evaluates to $\bot$ under the assignment $\{x \mapsto 5, y \mapsto -10, a \mapsto \bot\}$

We say that a formula φ is satisfiable modulo $\mathcal{T}$, if there is an assignment M that evaluates φ to $\top$. In that case we say that M is a **model**

Solving SMT formulæ by reduction to SAT

Approaches to solve SMT formulæ are based on the observation that SMT can be **reduced** to SAT, i.e., the purely Boolean Satisfiability Problem
Consider for instance the $\mathcal{LIA}$ formula

$$\varphi \equiv (x - y \leq 0) \wedge (y - z \leq 0) \wedge ((z - x \leq -1) \vee (z - x \leq -2))$$

We may use a Boolean variable a to mean "$x - y \leq 0$" evaluates to $\top$ in the model.
Similarly we could use b, c, d for the other $\mathcal{T}$-atoms.
First of all, we notice that it **does not** hold in $\mathcal{LIA}$ that

$$x - y \leq 0 \qquad y - z \leq 0 \qquad z - x \leq -1$$

evaluate to $\top$ **at the same time**, because this is not possible in $\mathcal{LIA}$. This translates to the Boolean relation

$$\neg(a \wedge b \wedge c)$$

Similarly we may derive

$$\neg(a \wedge b \wedge d) \qquad \neg(\neg a \wedge \neg b \wedge \neg c) \qquad \neg(\neg a \wedge \neg b \wedge \neg d)$$

Moreover, because of the "structure" of φ, it holds that

$$a \wedge b \wedge (c \vee d)$$

Solving SMT formulæ by reduction to SAT

$$\varphi \equiv (x - y \leq 0) \wedge (y - z \leq 0) \wedge ((z - x \leq -1) \vee (z - x \leq -2))$$

$$\begin{array}{lll} a & \text{represents} & x - y \leq 0 \\ b & \text{represents} & y - z \leq 0 \\ c & \text{represents} & z - x \leq -1 \\ d & \text{represents} & z - x \leq -2 \end{array}$$

Putting all the conditions together we get the Boolean formula

$$\psi \equiv \; a \wedge b \wedge (c \vee d) \; \wedge \; \neg(a \wedge b \wedge c) \; \wedge \; \neg(a \wedge b \wedge d) \; \wedge \; \neg(\neg a \wedge \neg b \wedge \neg c) \; \wedge \; \neg(\neg a \wedge \neg b \wedge \neg d)$$

Because of our translation, we have that φ is $\mathcal{LIA}$-satisfiable if and only if ψ is satisfiable. This is true because

1. we have **exhaustively** encoded incompatible relations between $\mathcal{T}$-atoms
2. we have encoded the structure of φ

Therefore we may run any off-the-shelf SAT-solver to determine the satisfiability of ψ (and therefore that of φ)

Exercizes

1. Check that φ is $\mathcal{LIA}$-unsatisfiable, and that ψ is also unsatisfiable
2. Check that $\neg(\neg a \wedge \neg b \wedge \neg c)$ and $\neg(\neg a \wedge \neg b \wedge \neg d)$ are actually redundant in ψ. Why it is so ?
3. Substitute $z - x \leq -2$ with $z - x \leq 2$ into φ, recompute the correct ψ, and check that φ is $\mathcal{LIA}$-satisfiable and that ψ is also satisfiable
4. Prove that the encoding into SAT is correct and complete, i.e., that if
   - (i) we have **exhaustively** encoded incompatible relations between $\mathcal{T}$-atoms
   - (ii) we have encoded the structure of φ

   then φ is $\mathcal{T}$-satisfiable if and only if ψ is satisfiable

The Eager and Lazy Approaches

Recall that in our reduction to SAT we need to encode

- (i) incompatible relations between $\mathcal{T}$-atoms exhaustively
- (ii) the structure of φ

Condition (ii) is easy to encode. The critical condition is (i). If we have 3 $\mathcal{T}$-atoms a, b, c, then we need to check whether

- a and b are incompatible
- a and $\neg b$ are incompatible
- ...
- a and b and c are incompatible
- ...

Potentially, this leads to checking $O(2^n)$ relations, if n $\mathcal{T}$-atoms are in the formula

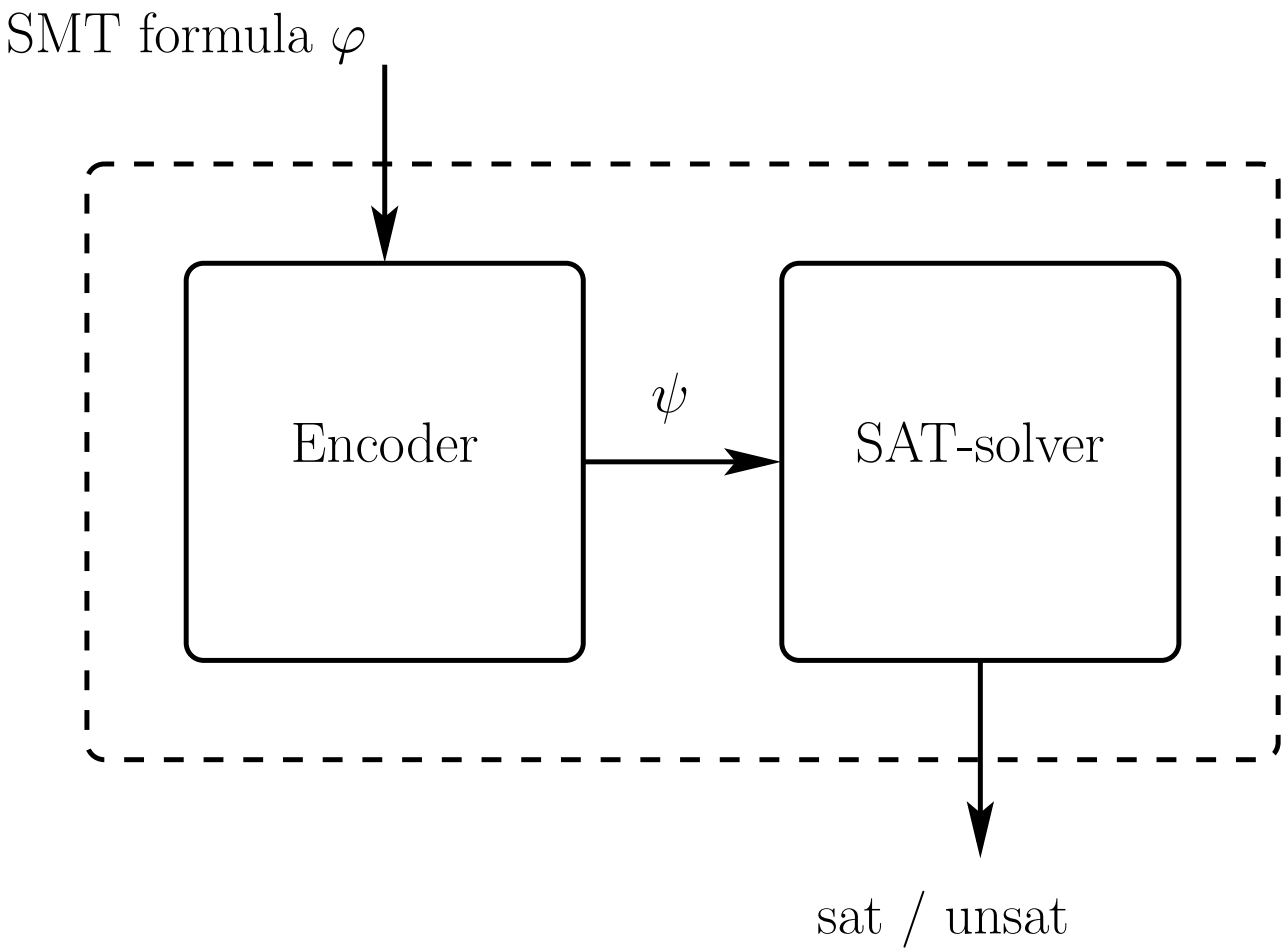
The Eager and Lazy Approaches

There are (at least) two ways to discover incompatibilities

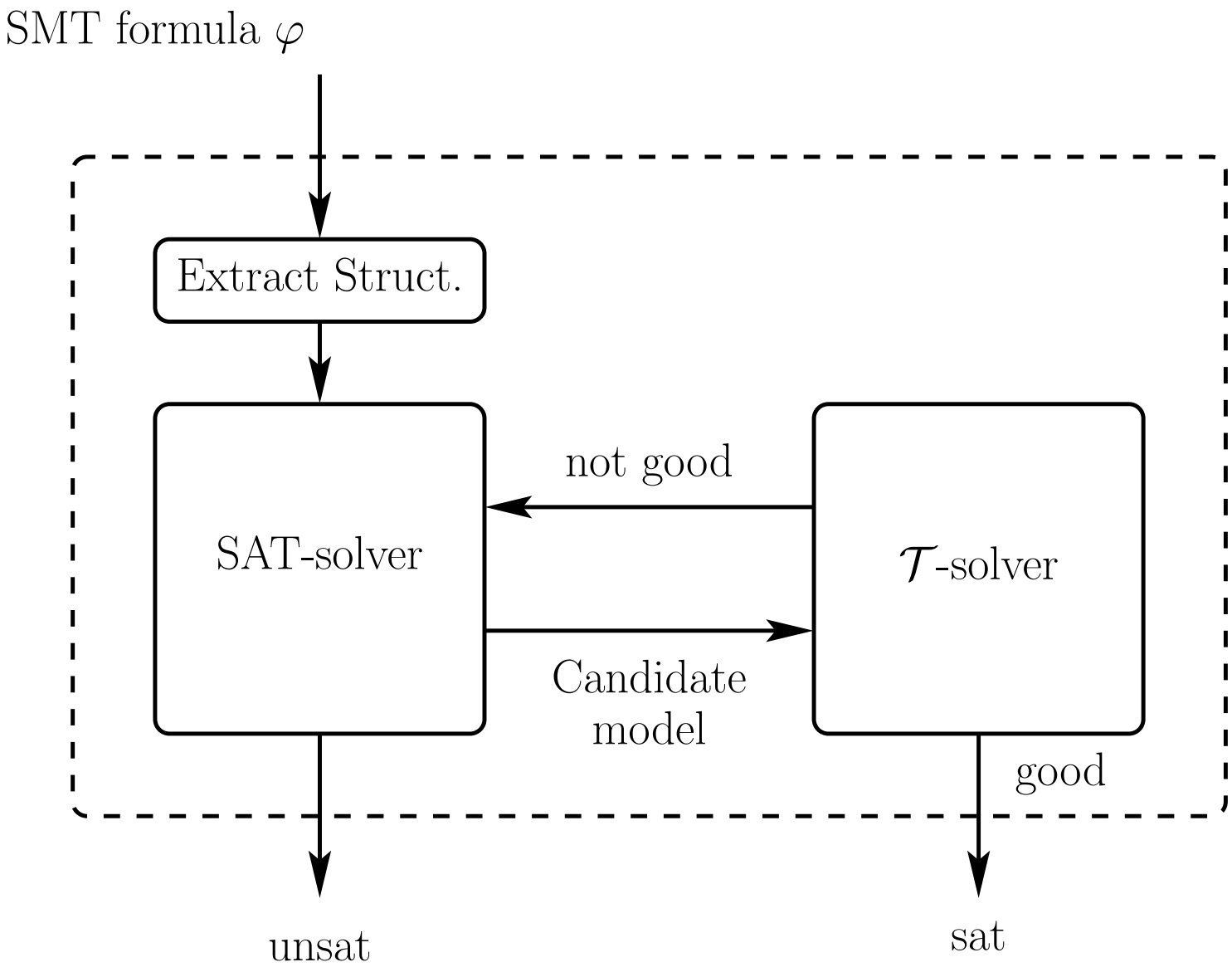
- **eagerly** adding them before calling a SAT-solver (eager approach)
  - \+ Easy to implement: SAT-solver used as black-box
  - \+ Good for bit-vectors theories
  - – Potentially generates too big encoding: needs heuristics to make it efficient
  - – Bad for arithmetic theories
- **lazily**, by adding them during the SAT-solver's search (lazy approach)
  - \+ Generates smaller encodings
  - \+ Good for arithmetic theories
  - \+ Modular approach: allows easy theory combination
  - – Trickier to implement: SAT-solver has to be "openen"

Most of this course will be devoted to the lazy approach, which is nowadays the most successful technique available

The Eager Approach



The Lazy Approach



Plan of (the rest of) the course

- The eager approach: solving bit-vectors
- Modern SAT-solvers: conflict analysis, clause learning, and heuristics
- The Lazy approach: generalities
- A theory-solver for $\mathcal{IDL}$
- A theory-solver for $\mathcal{UF}$
- A theory-solver for $\mathcal{LRA}$
- (see if there is time left)

Outline

SMT-LIB (v2) http://www.smtlib.org

The SMT-LIB initiative

- defines a standard input language for SMT-solvers
- defines theories and logics in which formulæ can be written
- collects benchmarks

The SMT-LIB language allows to write formulæ in a lisp-like format. E.g.:

```
(< (+ x y) 0)
(= (f x y) (g z))
```

stand for $x + y < 0$ and $f(x, y) = g(z)$ respectively

An SMT-LIB file looks more similar to a **set of commands** for an SMT-solver, rather then a logic formula

SMT-LIB Theories

An SMT-LIB theory consists of some **sorts**, (e.g., `Int`) and of some functions (e.g., $-, +$). Predicates are also considered functions, with codomain in `Bool` (e.g., $<, \leq$). For instance

```
(theory Ints

 :sorts ((Int 0))

 :funs ((NUMERAL Int)
        (- Int Int)
        (- Int Int Int :left-assoc)
        (+ Int Int Int :left-assoc)
        (* Int Int Int :left-assoc)
        (div Int Int Int :left-assoc)
        (mod Int Int Int)
        (abs Int Int)
        (<= Int Int Bool :chainable)
        (<  Int Int Bool :chainable)
        (>= Int Int Bool :chainable)
        (>  Int Int Bool :chainable)
       )

  [...]
)
```

```
(theory Core

 :sorts ((Bool 0))

 :funs ((true Bool)
        (false Bool)
        (not Bool Bool)
        (=> Bool Bool Bool :right-assoc)
        (and Bool Bool Bool :left-assoc)
        (or Bool Bool Bool :left-assoc)
        (xor Bool Bool Bool :left-assoc)
        (par (A) (= A A Bool :chainable))
        (par (A) (distinct A A Bool :pairwise))
        (par (A) (ite Bool A A A))
       )

 [...]
)
```

These definitions can be found at `www.smtlib.org`

The sorts and the function symbols declared in a theory are always **interpreted**. This means that a to specify a model for a formula φ, we just need to specify the assignment of the variables to the concrete values in the sorts.

SMT-LIB Logics

The difference between "logic" and "theory" might look very subtle. An SMT-LIB logic includes a theory definition, plus it describes some restrictions on how formulæ can be built.

```
(logic QF_LIA

 :theories (Ints)

 :language
 "Closed quantifier-free formulas built
 over an arbitrary expansion of the
 Ints signature with free constant symbols,
 but whose terms of sort Int are all linear,
 that is, have no occurrences of the function
 symbols *, /, div, mod, and abs, except as
 specified the :extensions attribute.
 "

 :extensions
 "Terms with _concrete_ coefficients are also
 allowed, that is, terms of the form c, (* c x),
 or (* x c)  where x is a free constant and c
 is a term of the form n or (- n) for some numeral n.
 "
)
```

```
(logic QF_IDL

 :theories (Ints)

 :language
 "Closed quantifier-free formulas with
 atoms of the form:
 - q
 - (op (- x y) n),
 - (op (- x y) (- n)), or
 - (op x y)
 where
 - q is a variable or free constant symbol of sort Bool,
 - op is <, <=, >, >=, =, or distinct,
 - x, y are free constant symbols of sort Int,
 - n is a numeral.
 "
)
```

In the following we will not be so strict, and we will not make any distinction between "theories" and "logics", calling both "theories". For instance when we will say that we reason modulo the theory $\mathcal{LIA}$ we mean that we are working with `QF_LIA` formulæ

Writing an SMT-LIB file

The logic can be specified with the command

```
(set-logic QF_LIA)
```

Variables are declared with

```
(declare-fun x ( ) Int)
```

A formula is specified with

```
(assert (<= (+ x y) 0))
```

Asks the tool to compute satisfiability of assertions

```
(check-sat)
```

Asks the tool to return a model (in case of sat result)

```
(set-option :produce-models true)
...
(get-value (x y))
```

Disable annoying printouts

```
(set-option :print-success false)
```

Example

```
(set-logic QF_LIA)
(declare-fun x ( ) Int)
(declare-fun y ( ) Int)
(declare-fun a ( ) Bool)
(assert (<= (+ x y) 0))
(assert (= x 0))
(assert (or (not a) (= x 1) (>= y 0)))
(assert (not (= (+ y 1) 0)))
(check-sat)
(exit)
```

which stands for the $\mathcal{LIA}$ formula

$$(x + y \leq 0) \wedge (x = 0) \wedge ((\neg a \vee (x = 1) \vee (y \geq 0)) \wedge \neg(y + 1 = 0)$$

SMT-solvers

An SMT-solver is a tool that can parse and solve an SMT-LIB benchmark.

There are many such tools available online. In this course we will use YICES (developed at SRI, Stanford, closed source), Z3 (developed at MSR, Redmond, closed source) and OPENSMT (developed here, open source). Other available tools are MATHSAT, CVC4, BOOLECTOR, VERIT, STP.

```
roberto@moriarty:examples$ smtlib2yices < test1.smt2
success
success
success
success
success
success
success
success
success
sat
```

SMT-LIB script

The SMT-LIB language allows specification of **scripts**. A script is a benchmark that may contain many `check-sat` commands. Also, it may include `push` and `pop` commands which can be used to control the assertion stack

```
(set-option :print-success false)
(set-logic QF_LIA)
(declare-fun x ( ) Int)
(declare-fun y ( ) Int)
(assert (<= (+ x y) 0))
(assert (= x 0))
(assert (or (= x 1) (>= y 0)))
(check-sat)
(push 1)
(assert (not (= y 0)))
(check-sat)
(pop 1)
(check-sat)
(exit)
```

Exercizes

1. Translate the following $\mathcal{LIA}$ formula SMT-LIB, and evaluate it with an SMT-solver

$$(x - y \leq 0) \wedge (y - z \leq 0) \wedge ((z - x \leq -1) \vee (z - x \leq -2))$$

2. Translate the following $\mathcal{LRA}$ formula SMT-LIB, and evaluate it with an SMT-solver

$$(b \vee (x + y \leq 0)) \wedge (\neg b \vee (x + z \leq 10))$$

3. For the satisfiable formulæ above print out a model

4. For the satisfiable formulæ above, add constraints such that they become unsatisfiable