



SAT/SMT Solving and Applications in Rewriting

René Thiemann ¹ Sarah Winkler ²

¹University of Innsbruck

²Free University of Bolzano

Outline

1. Solution of Exercise of Session 1
2. Beyond SAT: Motivation via Knuth–Bendix Orders
3. Applying SMT Solvers
4. SMT Solving, DPLL(T) and CDCL(T)
5. Further Reading

Exercise

- LPO is a relation on terms \succ_{LPO} (\succ for short), parametrized by precedence p

$$\frac{s_i = t \vee s_i \succ t}{s = f(s_1, \dots, s_n) \succ t} \quad (\text{sub})$$

$$\frac{p(f) > p(g) \quad \forall i \in \{1, \dots, m\}. s \succ t_i}{s = f(\dots) \succ g(t_1, \dots, t_m) = t} \quad (\text{prec})$$

$$\frac{\forall j \in \{1, \dots, i-1\}. s_j = t_j \quad s_i \succ t_i \quad \forall j \in \{i+1, \dots, n\}. s \succ t_j}{s = f(s_1, \dots, s_n) \succ f(t_1, \dots, t_n) = t} \quad (\text{lex})$$

- given some argument filter π define

- $\pi(x) = x$

- $$\pi(f(t_1, \dots, t_n)) = \begin{cases} \pi(t_i), & \text{if } \pi(f) = i \\ f([\pi(t_j) \mid i \leftarrow [1..n], i \in \pi(f)]), & \text{if } \pi(f) \text{ is a set} \end{cases}$$

- exercise: given s and t , encode “ $\exists \pi p. \pi(s) \succ_{LPO(p)} \pi(t)$ ” as SAT problem

Towards an Encoding of $\ulcorner \pi(s) \succ \pi(t) \urcorner$

- encoding is based on variables $i \in \pi(f)$, $\text{set}(f)$, $\ulcorner \pi(s_i) \succ \pi(t_j) \urcorner$, $\ulcorner \pi(s_i) = \pi(t_j) \urcorner$ (and precedence encoding as before)
- $\ulcorner \pi(s_i) \succeq \pi(t_j) \urcorner := \ulcorner \pi(s_i) \succ \pi(t_j) \urcorner \vee \ulcorner \pi(s_i) = \pi(t_j) \urcorner$
- add the following constraint about $\ulcorner \pi(s_i) = \pi(t_j) \urcorner$
- $\ulcorner \pi(x) = \pi(y) \urcorner \rightarrow \perp$ if $x \neq y$
- $\ulcorner \pi(f(s_1, \dots, s_n)) = \pi(t_j) \urcorner \rightarrow \neg \text{set}(f) \rightarrow i \in \pi(f) \rightarrow \ulcorner \pi(s_i) = \pi(t_j) \urcorner$
- $\ulcorner \pi(s_i) = \pi(g(t_1, \dots, t_m)) \urcorner \rightarrow \neg \text{set}(g) \rightarrow j \in \pi(g) \rightarrow \ulcorner \pi(s_i) = \pi(t_j) \urcorner$
- $\neg \text{set}(f) \rightarrow \ulcorner \text{exactlyOne}(1 \in \pi(f), \dots, n \in \pi(f)) \urcorner$ for n -ary f
- $\ulcorner \pi(f(\dots)) = \pi(y) \urcorner \rightarrow \neg \text{set}(f)$
- $\ulcorner \pi(x) = \pi(g(\dots)) \urcorner \rightarrow \neg \text{set}(g)$
- $\ulcorner \pi(f(\dots)) = \pi(g(\dots)) \urcorner \rightarrow \neg \text{set}(f) \vee \neg \text{set}(g)$ if $f \neq g$
- $\ulcorner \pi(f(s_1, \dots, s_n)) = \pi(f(t_1, \dots, t_n)) \urcorner \rightarrow \text{set}(f) \rightarrow i \in \pi(f) \rightarrow \ulcorner \pi(s_i) = \pi(t_i) \urcorner$

An Encoding of $\lceil \pi(s) \succ \pi(t) \rceil$

- add all of the following constraints
- $\lceil \pi(s) \succ \pi(t) \rceil$
- $\lceil \pi(x) \succ \pi(t_j) \rceil \rightarrow \perp$
- $\lceil \pi(f(s_1, \dots, s_n)) \succ \pi(t_j) \rceil \rightarrow \neg \text{set}(f) \rightarrow i \in \pi(f) \rightarrow \lceil \pi(s_i) \succ \pi(t_j) \rceil$
- $\lceil \pi(s_i) \succ \pi(g(t_1, \dots, t_m)) \rceil \rightarrow \neg \text{set}(g) \rightarrow j \in \pi(g) \rightarrow \lceil \pi(s_i) \succ \pi(t_j) \rceil$
- $\lceil \pi(f(s_1, \dots, s_n)) \succ \pi(y) \rceil \rightarrow \text{set}(f) \rightarrow \bigvee_i (i \in \pi(f) \wedge \lceil \pi(s_i) \succeq \pi(y) \rceil)$
- $\lceil \pi(f(s_1, \dots, s_n)) \succ \pi(g(t_1, \dots, t_m)) \rceil \rightarrow \text{set}(f) \rightarrow \text{set}(g) \rightarrow$
 $(\lceil p(f) \succ p(g) \rceil \wedge \bigwedge_j (j \in \pi(g) \rightarrow \lceil \pi(f(s_1, \dots, s_n)) \succ \pi(t_j) \rceil))$
 $\vee \bigvee_i (i \in \pi(f) \wedge \lceil \pi(s_i) \succeq \pi(g(t_1, \dots, t_m)) \rceil) \quad \text{if } f \neq g$
- $\lceil \pi(f(s_1, \dots, s_n)) \succ \pi(f(t_1, \dots, t_n)) \rceil \rightarrow \text{set}(f) \rightarrow$
 $\bigvee_i (i \in \pi(f) \wedge \lceil \pi(s_i) \succeq \pi(f(t_1, \dots, t_n)) \rceil)$
 $\vee \bigvee_i (i \in \pi(f) \wedge \lceil \pi(s_i) \succ \pi(t_i) \rceil \wedge \bigwedge_{j < i} (j \in \pi(f) \rightarrow \lceil \pi(s_j) = \pi(t_j) \rceil) \wedge$
 $\bigwedge_{j > i} (j \in \pi(f) \rightarrow \lceil \pi(f(s_1, \dots, s_n)) \succ \pi(t_j) \rceil))$

Outline

1. Solution of Exercise of Session 1
- 2. Beyond SAT: Motivation via Knuth–Bendix Orders**
3. Applying SMT Solvers
4. SMT Solving, DPLL(T) and CDCL(T)
5. Further Reading

Definition (Knuth–Bendix Order (KBO))

- let $p : \mathcal{F} \rightarrow \mathbb{N}$ be some **precedence**
- let $w_0 \in \mathbb{N} \setminus \{0\}$ and let $w : \mathcal{F} \rightarrow \mathbb{N}$ be some **weight function**
- define $w(t) = \begin{cases} w_0, & \text{if } t \text{ is a variable} \\ w(f) + w(t_1) + \dots + w(t_n), & \text{if } t = f(t_1, \dots, t_n) \end{cases}$
- KBO is a relation on terms \succ_{KBO} (\succ for short), defined by these inference rules

$$\frac{\mathcal{V}(s) \supseteq \mathcal{V}(t) \quad w(s) > w(t)}{s \succ t} \quad \text{(weight)}$$

$$\frac{\mathcal{V}(s) \supseteq \mathcal{V}(t) \quad w(s) \geq w(t)}{s = f(\dots) \succ x = t} \quad \text{(variable)}$$

$$\frac{\mathcal{V}(s) \supseteq \mathcal{V}(t) \quad w(s) \geq w(t) \quad p(f) > p(g)}{s = f(\dots) \succ g(\dots) = t} \quad \text{(precedence)}$$

$$\frac{\mathcal{V}(s) \supseteq \mathcal{V}(t) \quad w(s) \geq w(t) \quad \forall j \in \{1, \dots, i-1\}. s_j = t_j \quad s_i \succ t_i}{s = f(s_1, \dots, s_n) \succ f(t_1, \dots, t_n) = t} \quad \text{(lexicographic)}$$

Definition (Admissibility)

KBO parameters are **admissible** if

- for all constants f : $w(f) \geq w_0$, and
- for all unary symbols f : $w(f) = 0$ implies that $p(f) > p(g)$ for all $g \neq f$

Theorem

every KBO with admissible parameters is a reduction order

Task: given TRS, search for KBO parameters

- search for suitable rules is easier than LPO: no overlap
- problem: calculation and comparisons of weights requires **arithmetic**
- solution:
 - switch from SAT to SMT (SAT modulo theories)
 - required theory for KBO: linear arithmetic

SAT Modulo Theories (SMT)

- theory defines **sorted** first-order terms with standard semantics
- **theory atom** is term of Boolean sort
- **SMT formulas** are formulas as in propositional case, extended by theory atoms
- **SMT solving**: given SMT formula, determine whether there is a satisfying assignment
- such an assignment may contain theory variables and propositional variables

Example (Theories)

- **LRA**: linear real arithmetic; **NRA**: non-linear real arithmetic
 - domain: \mathbb{R}
 - arbitrary addition; in linear case multiplication only by constants
 - example formula: $a \wedge x + 2y > 5 \rightarrow \neg(2x - 7 = 5z) \vee z \geq \frac{14}{5} \vee \neg b$
 - a solution: $\alpha(a) = \alpha(b) = \text{T}$, $\alpha(x) = 3$, $\alpha(y) = 2$, $\alpha(z) = -\frac{13}{5}$
- **LIA**: linear integer arithmetic; **NIA**: non-linear integer arithmetic
 - same as LRA and NRA, except that theory variables and constants are restricted to \mathbb{Z}

Definition (KBO encoding into LIA)

- encoding uses **integer variables** $p(f)$, $w(f)$, w_0 and $w(t)$
- constraints are now straight-forward for TRS \mathcal{R} over signature \mathcal{F}
- $w(f) \geq 0$ for all $f \in \mathcal{F}$
- $w(f) \geq w_0$ for all constants f
- $w(f) = 0 \rightarrow \bigwedge_{g \neq f} p(f) > p(g)$ for all unary f
- $w(x) = w_0$ for all variables x in \mathcal{R}
- $w(f(t_1, \dots, t_n)) = w(f) + w(t_1) + \dots + w(t_n)$ for all subterms in \mathcal{R}
- $\lceil s > t \rceil := \begin{cases} \perp, & \text{if } \mathcal{V}(s) \not\supseteq \mathcal{V}(t) \\ w(s) > w(t) \vee w(s) \geq w(t) \wedge \langle s > t \rangle, & \text{if } \mathcal{V}(s) \supseteq \mathcal{V}(t) \end{cases}$
- $\langle f(\dots) > x \rangle := \top$
- $\langle x > t \rangle := \perp$
- $\langle f(\dots) > g(\dots) \rangle := p(f) > p(g)$, if $f \neq g$
- $\langle f(s_1, \dots, s_n) > f(t_1, \dots, t_n) \rangle := \begin{cases} \perp, & \text{if } (s_1, \dots, s_n) = (t_1, \dots, t_n) \\ \lceil s_i > t_i \rceil, & \text{if } (s_1, \dots, s_{i-1}) = (t_1, \dots, t_{i-1}) \text{ and } s_i \neq t_i \end{cases}$

Exercise

Design a LIA encoding that searches for an argument filter π and KBO parameters; in particular you need an encoding for constraints of the form

$$\lceil \pi(s) \succ_{KBO} \pi(t) \rceil$$

hints

1

2

Outline

1. Solution of Exercise of Session 1
2. Beyond SAT: Motivation via Knuth–Bendix Orders
- 3. Applying SMT Solvers**
4. SMT Solving, DPLL(T) and CDCL(T)
5. Further Reading

Applying an SMT Solver

Two Alternatives

1 use any SMT solver and standard format

- SMT-Lib is widely used format

<https://smt-lib.org/language.shtml>

- for successful SMT solvers, look at competition

<https://smt-comp.github.io/2024/>

- demos have been tested with Z3 (Microsoft Research)

<https://github.com/Z3Prover/z3>

2 use some language-binding for your programming language and SMT solver

- simple-smt: <https://hackage.haskell.org/package/simple-smt>
- sbv: <https://hackage.haskell.org/package/sbv>
- Z3-bindings: <https://github.com/Z3Prover/z3#z3-bindings>

SMT-LIB 2 Format – KBO for $\{plus(s(x),y) \rightarrow s(plus(x,y)); plus(0,y) \rightarrow y\}$ (1/4)

```
(set-logic QF_LIA) ; QF_LIA, QF_LRA, QF_NIA, QF_NRA; QF = quantifier free
(declare-fun w0 () Int ) ; declare theory variables (type: Int or Real)
; declare propositional variables (type: Bool)

(declare-fun wf_plus () Int ) ; weights for symbols
(declare-fun wf_s () Int )

(declare-fun pf_plus () Int ) ; precedences
(declare-fun pf_s () Int )

(declare-fun w_s__x () Int ) ; weights for terms, all variables = x
(declare-fun w_plus__s__x__x () Int )
(declare-fun w_plus__x__x () Int )
(declare-fun w_s__plus__x__x () Int )
```

SMT-LIB 2 Format – KBO for $\{plus(s(x),y) \rightarrow s(plus(x,y)); plus(0,y) \rightarrow y\}$ (2/4)

```
; assertion are added using S-expressions
; predefined on Booleans: and, or, not, true, false, => (implication), = (equivalence)
; predefined on numbers: numerals, +, *, - (beware: (- 5), not: -5)
; predefined comparisons: =, <, >, >=, <=
```

```
(assert (and (<= 0 pf_plus ) (<= pf_plus 1 ) ) ) ; restrict precedence range
(assert (and (<= 0 pf_s ) (<= pf_s 1 ) ) )
```

```
(assert (> w0 0 ) ) ; admissibility
(assert (>= wf_s 0 ) )
(assert (>= wf_plus 0 ) )
(assert (=> (= wf_s 0 ) (> pf_s pf_plus ) ) )
```

```
(assert (= w_s__x (+ wf_s w0 ) ) ) ; weight computation
(assert (= w_plus__s__x__x (+ wf_plus w_s__x w0 ) ) )
(assert (= w_plus__x__x (+ wf_plus w0 w0 ) ) )
(assert (= w_s__plus__x__x (+ wf_s w_plus__x__x ) ) )
```

```

; rule constraints
(assert (or (> w_plus__s__x__x w_s__plus__x__x )
            (and (>= w_plus__s__x__x w_s__plus__x__x ) (> pf_plus pf_s ) ) ) )
(assert true )

(check-sat )           ; invoke solver

(get-value (w0 ) )    ; extract solution for variable w0
(get-value (wf_plus ) ) ; weight of plus
(get-value (pf_plus ) ) ; precedence of plus
(get-value (wf_s ) )   ; weight of s
(get-value (pf_s ) )   ; precedence of s
; precedence and weight of symbol 0 was not required -> set to defaults: 0 and w0

```

Remarks

- declarations and assertions can be mixed
- SMT-Lib also supports incremental invocations, cf. **assertion stacks, push, pop**

SMT-LIB 2 Format – KBO for $\{plus(s(x),y) \rightarrow s(plus(x,y)); plus(0,y) \rightarrow y\}$ (4/4)

- store encoding in file, e.g., `kbo_plus_encoding.smt2`
- invoke SMT solver, e.g., `z3`

```
$ z3 kbo_plus_encoding.smt2
```

```
sat
((w0 1))
((wf_plus 0))
((pf_plus 2))
((wf_s 1))
((pf_s 1))
```

SMT-Solving via Language-Bindings

- have a look at `demos/miniTT/src/KB0.hs`
- there, `simple-smt` is used
- a bit more complex to use than SAT-MiniSat
 - variables need to be declared before usage
 - variable names must be valid
(encoding must take care of forbidden characters and keywords in SMT, cf. `stringToSMT`, `wSym`, `pSym`, ...)

Design Choices

- which theory to select?

fragment	LRA	LIA	NRA	NIA
quantifier-free	NPC	NPC	$O(2^{2^n})$	undecidable
qf, only conjunctions	P	NPC	$O(2^{2^n})$	undecidable
allowing quantors	decidable	decidable	$O(2^{2^n})$	undecidable

table indicates that LRA is theoretically the easiest (LRA is convex, LIA is not)

- choice should be **backed by experiments**
 - KBO can also be encoded in LRA instead of LIA (`miniTT/src/KB02.hs`)
 - intuition: LRA should be faster than LIA
 - experiments: KBO-LRA is **2x slower** than KBO-LIA on termination problem database
- another tradeoff: **encoding** time vs **solving** time

Outline

1. Solution of Exercise of Session 1
2. Beyond SAT: Motivation via Knuth–Bendix Orders
3. Applying SMT Solvers
4. SMT Solving, DPLL(T) and CDCL(T)
5. Further Reading

SMT Problem

decide satisfiability of formulas in

propositional logic + domain-specific background theories

Two Approaches

① **eager** approach:

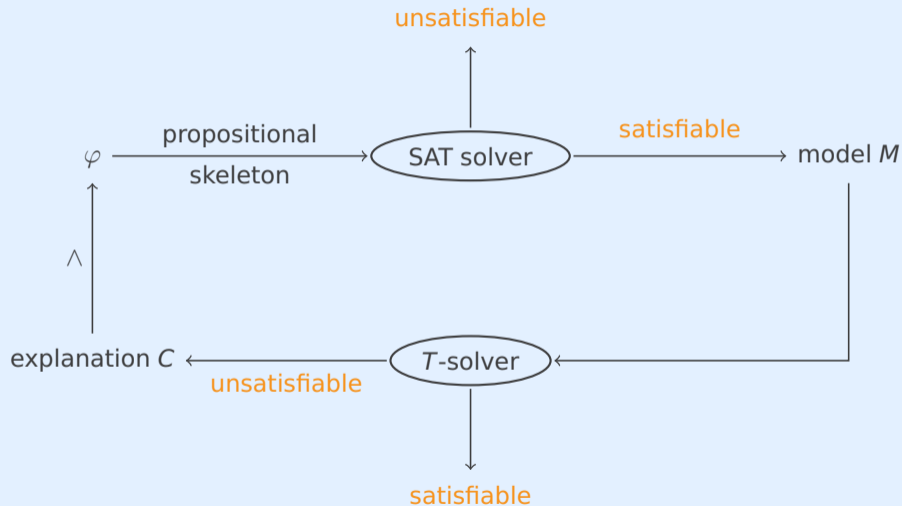
translate formula into equisatisfiable propositional formula

② **lazy** approach:

combine SAT solver with **specialized solvers** for background theories

Terminology

theory solver for T (**T -solver**) is procedure for deciding T -satisfiability of **conjunction** of **quantifier-free** literals



Example

formula $x = 1 \wedge (\neg(y = 1) \vee \neg(x + 2y = 3)) \wedge x + y = 2$ is **unsatisfiable**

a b c d

- input to SAT solver (propositional skeleton)

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

blocking clause

- SAT solver reports **unsatisfiable**

$$a \wedge b \wedge \neg c \wedge d$$

- input to LIA solver

$$x = 1 \wedge y = 1 \wedge x + 2y \neq 3 \wedge x + y = 2$$

- LIA solver reports **unsatisfiable**

most state-of-the-art SMT solvers use **DPLL(T)** (including CDCL extensions)

general framework for lazy SMT solving with theory propagation

Definitions

first-order theory T , formulas F and G , list of literals M

- F is **T -satisfiable** if $F \wedge T$ is satisfiable
- $F \vDash_T G$ if $F \wedge \neg G$ is not T -satisfiable
- $F \equiv_T G$ if $F \vDash_T G$ and $G \vDash_T F$
- $M = l_1, \dots, l_k$ is **T -consistent** if $l_1 \wedge \dots \wedge l_k$ is T -satisfiable

Definition

DPLL(T) consists of DPLL rules **unit propagate**, **decide**, **fail**, **restart** and

- **T -backjump**
$$M \overset{d}{I} N \parallel F, C \implies M I' \parallel F, C$$

if $M \overset{d}{I} N \models \neg C$ and \exists clause $C' \vee I'$ such that

- $F, C \models_T C' \vee I'$ and $M \models \neg C'$

- I' is undefined in M and I' or I'^c occurs in F or in $M \overset{d}{I} N$

- **T -learn**
$$M \parallel F \implies M \parallel F, C$$

if $F \models_T C$ and all atoms of C occur in M or F

- **T -forget**
$$M \parallel F, C \implies M \parallel F$$

if $F \models_T C$

- **T -propagate**
$$M \parallel F \implies M I \parallel F$$

if $M \models_T I$, I is undefined in M , and I or I^c occurs in F

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that **unit propagate**, **decide**, **fail**, **T -backjump** are not applicable

check T -consistency of M with T -solver

- 1 if M is T -consistent then F is T -satisfiable
- 2 if M is not T -consistent then $F \models_T \neg(l_1 \wedge \dots \wedge l_k)$ for some literals l_1, \dots, l_k in M
add blocking clause $\neg l_1 \vee \dots \vee \neg l_k$ by **T -learn** and apply **restart**

Improvements

- 1 apply **fail** or **T -backjump** after **T -learn** (instead of **restart**)
- 2 check T -consistency of M or apply **T -propagate** before **decide**
- 3 find small unsatisfiable cores to minimize k in blocking clauses

Outline

1. Solution of Exercise of Session 1
2. Beyond SAT: Motivation via Knuth–Bendix Orders
3. Applying SMT Solvers
4. SMT Solving, DPLL(T) and CDCL(T)
5. Further Reading

Further Reading

- Harald Zankl, Nao Hirokawa, and Aart Middeldorp
KBO Orientability
Journal of Automated Reasoning 43(2), pp. 173–201, 2009
- Michael Codish, Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann
SAT Solving for Termination Proofs with Recursive Path Orders and Dependency Pairs
Journal of Automated Reasoning, 49(1), pp. 53–93, 2012
- Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli
Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)
Journal of the ACM 53(6), pp. 937–977, 2006