SAT/SMT Solving and Applications in Rewriting

René Thiemann [1] Sarah Winkler [2]

[1]University of Innsbruck

[2]Free University of Bolzano

Outline

1. **Solution of Exercise of Session 3**
2. **Beyond Reduction Orders and Termination**
3. **Logically Constrained Term Rewrite Systems**
4. **Further Reading**

Exercise: Develop an optimized encoder for polynomial constraints

- start from initial polynomial constraints as before
- perform several simplification steps and identify $f_i \geq 1$ conditions
- eliminate common terms
- eliminate $p \geq 0$
- simplify $0 \geq q$ to $q = 0$
- simplify $p + q = 0$ to $p = 0$ and $q = 0$
- simplify $f_i p = 0$ to $p = 0$ whenever $f_i \geq 1$ is known
- whenever $f_i p > q$ conclude $f_i \geq 1$
- whenever $f_i = 0$ is present, substitute and simplify everywhere else
- simplify $f_i p \geq f_i q$ to $p \geq q$ (and also with $>$) whenever $f_i \geq 1$ is known
- ...

Example ($m(s(x), s(y)) \succsim m(x, y) \wedge m(x, z) \succsim x \wedge d(s(x), s(y)) \succ d(m(x, y), s(y))$)

$$m_0 + m_1 s_0 + m_2 s_0 \geq m_0 \qquad m_1 s_1 \geq m_1 \qquad m_2 s_1 \geq m_2$$
$$m_0 + m_2 0_0 \geq 0 \qquad m_1 \geq 1$$
$$d_0 + d_1 s_0 + d_2 s_0 > d_0 + d_1 m_0 + d_2 s_0 \qquad d_1 s_1 \geq d_1 m_1 \qquad d_2 s_1 \geq d_1 m_2 + d_2 s_1$$

$$m_1 s_0 + m_2 s_0 \geq 0 \qquad m_1 s_1 \geq m_1 \qquad m_2 s_1 \geq m_2$$
$$m_0 + m_2 0_0 \geq 0 \qquad m_1 \geq 1$$
$$d_1 s_0 > d_1 m_0 \qquad d_1 s_1 \geq d_1 m_1 \qquad 0 \geq d_1 m_2$$

$$m_1 s_1 \geq m_1 \qquad m_2 s_1 \geq m_2$$
$$m_1 \geq 1$$
$$d_1 s_0 > d_1 m_0 \qquad d_1 s_1 \geq d_1 m_1 \qquad 0 \geq d_1 m_2$$

Example $(m(s(x),s(y)) \succsim m(x,y) \wedge m(x,z) \succsim x \wedge d(s(x),s(y)) \succ d(m(x,y),s(y)))$

$$
\begin{array}{ccc}
 & m_1 s_1 \geq m_1 & m_2 s_1 \geq m_2 \\
 & m_1 \geq 1 & \\
d_1 s_0 > d_1 m_0 & d_1 s_1 \geq d_1 m_1 & 0 \geq d_1 m_2 \\
\hline
d_1 \geq 1 & s_1 \geq 1 & m_2 s_1 \geq m_2 \\
 & m_1 \geq 1 & \\
s_0 > m_0 & s_1 \geq m_1 & 0 \geq m_2 \\
\hline
d_1 \geq 1 & s_1 \geq 1 & \\
 & m_1 \geq 1 & \\
s_0 > m_0 & s_1 \geq m_1 & m_2 = 0
\end{array}
$$

final constraints are linear!

Final Exercise

- knowledge: basics to encode KBO, LPO, polynomial order (POLO) via SAT/SMT
- weighted path order (WPO) combines features of KBO, LPO, and POLO
- task 1: lookup definition of WPO and design encoding
- fact: using only WPO, newcomer termination tool NaTT got 2nd place in termComp
- task 2 (optionally 🙂)
  - implement the encoding in your tool
  - implement dependency pairs and usable rules
  - add CPF generation
  - participate in termComp
- remark: miniTT was implemented from scratch in two days, including library search

Outline

1. Solution of Exercise of Session 3
2. **Beyond Reduction Orders and Termination**
3. Logically Constrained Term Rewrite Systems
4. Further Reading

Current State

1. synthesize reduction orders (KBO, LPO, POLO with $f_i \geq 1$)
2. synthesize reduction pairs (KBO + AF, LPO + AF, POLO with $f_i \geq 0$)

Using Reduction Orders

- ensure termination by demanding $\ell_i \succ r_i$ for all n rules
- ensure incremental termination: rule removal
  - add n Boolean variables $strict_i$
  - $\bigvee strict_i$
  - $\bigwedge \ell_i \succeq r_i$
  - $\bigwedge (strict_i \to \ell_i \succ r_i)$
  - solve all of these constraints
  - afterwards, remove all strictly oriented rules and continue with remaining rules

works good with POLO, and for KBO and LPO with quasi-precedences

Using Reduction Pairs

- main difference to reduction order: $\succeq$ is closed under contexts, not $\succ$
- applications for termination
  - combine with dependency pairs, use DP framework $\Longrightarrow$ big increase of power

- application for confluence
  - disprove confluence of $\mathcal{R}$ by proving non-joinability
  - given a peak $t_1 \ {}_{\mathcal{R}}^{*}\!\!\leftarrow s \rightarrow_{\mathcal{R}}^{*} t_2$, try to prove that join $t_1 \rightarrow_{\mathcal{R}}^{*} u \ {}_{\mathcal{R}}^{*}\!\!\leftarrow t_2$ is not possible
  - approach via reduction pairs (or more relaxed: discrimination pairs)
    - for $i = 1, 2$ approximate usable rules $\mathcal{U}_i$ for t_i such that $t_i \rightarrow_{\mathcal{R}}^{*} v$ implies $t_i \rightarrow_{\mathcal{U}_i}^{*} v$ for all v
    - find reduction pair such that $\mathcal{U}_1 \subseteq \{\succeq\}$ and $\mathcal{U}_2^{-1} \subseteq \{\succeq\}$ and $t_2 \succ t_1$

- application for infeasibility: given $s, t, \mathcal{R}$, prove that $s\sigma \rightarrow_{\mathcal{R}}^{*} t\sigma$ is not possible
  - approach via reduction pairs (or more relaxed: co-rewrite pairs)
    - find reduction pair such that $\mathcal{R} \subseteq \succeq$ and $t \succ s$ or $\mathcal{R}^{-1} \subseteq \succeq$ and $s \succ t$

Outline

1. Solution of Exercise of Session 3
2. Beyond Reduction Orders and Termination
3. **Logically Constrained Term Rewrite Systems**
   Applications
4. Further Reading

Logically Constrained Rewriting = Term Rewriting + SMT

typically decidable by SMT solvers

Idea

incorporate background theories into rewrite formalism for more convenient/efficient modeling

Terms are built from

- theory symbols $\mathcal{F}_L$, including booleans and $=$ — fixed interpretation in theory
- proper symbols $\mathcal{F}_T$ — free

Logically constrained rewrite rules

$$f(\ell_1, \ldots, \ell_n) \to r \qquad [c]$$

- left-hand side is rooted by proper symbol
- side constraint is theory term of sort `bool`
- set of such rules is logically constrained rewrite system (LCTRS)

Rewrite relation is union of ...

$\rightarrow_{\text{rule}}$ apply rewrite rule if substituted constraint valid

$\rightarrow_{\text{calc}}$ evaluate theory expressions

Example

LCTRS $\mathcal{R}$ over theory of integer arithmetic:

$$\mathsf{fact}(x) \rightarrow 1 \quad [x \leqslant 0] \qquad\qquad \mathsf{fact}(x) \rightarrow \mathsf{fact}(x - 1) \cdot x \quad [x - 1 \geqslant 0]$$

admits following rewrite steps:

$$\begin{aligned}
\mathsf{fact}(2) &\rightarrow_{\text{rule}} \mathsf{fact}(2 - 1) \cdot 2 && 2 - 1 \geqslant 0 \quad \text{valid} \\
&\rightarrow_{\text{calc}} \mathsf{fact}(1) \cdot 2 && \\
&\rightarrow_{\text{rule}} (\mathsf{fact}(1 - 1) \cdot 1) \cdot 2 && 1 - 1 \geqslant 0 \quad \text{valid} \\
&\rightarrow_{\text{calc}} (\mathsf{fact}(0) \cdot 1) \cdot 2 && \\
&\rightarrow_{\text{rule}} (1 \cdot 1) \cdot 2 && 0 \leqslant 0 \quad \text{valid} \\
&\rightarrow_{\text{calc}}^{+} 2 &&
\end{aligned}$$

More formally ...

Definition (Logic and Terms)

- $\mathcal{F}_T$ is sorted term signature
- $\mathcal{F}_L$ is sorted logic signature, including boolean operations ($\mathsf{true}, \mathsf{false}, \wedge, \dots$) and $=$
- $\mathcal{V}al := \mathcal{F}_L \cap \mathcal{F}_T$ is set of constants called **values**
- **terms** $\mathcal{T}(\mathcal{F}_L, \mathcal{V})$ are called **logical** and associated with fixed interpretation in theory $\mathcal{T}$

Definition

- **logically constrained rule** is triple $\ell \to r\ [\varphi]$ where $\mathrm{root}(\ell) \in \mathcal{F}_T \setminus \mathcal{F}_L$ and r has same sort as ℓ, and φ is logical term of sort bool
- **LCTRS** $\mathcal{R}$ is set of logically constrained rules

Example (LCTRS over theory of integer arithmetic)

- $\mathcal{V}al = \{\mathsf{true}, \mathsf{false}\} \cup \{0, 1, -1, 2, \dots\}$, $\mathcal{F}_L = \mathcal{V}al \cup \{+, -, \wedge, \vee, \cdot\}$ and $\mathcal{F}_T = \mathcal{V}al \cup \{\mathsf{fact}\}$
- $0, 1 + 2, x + (y - 1)$ are logical terms
- $\mathsf{fact}(0) \to 0\ [\mathsf{true}]$

Definition (Constrained Rewriting)

- $\mathcal{R}_{calc} = \{f(x_1, \ldots, x_n) \to y \ [y = f(x_1, \ldots, x_n)] \mid f \in \mathcal{F}_L \setminus \mathcal{V}al\}$
- substitution σ **respects** φ if $\varphi\sigma$ valid and $\sigma(x) \in \mathcal{V}al \ \forall x \in \mathcal{V}ar(\varphi)$
- $C[\ell\sigma] \to_{\mathcal{R}} C[r\sigma]$ for LCTRS $\mathcal{R}$ if $\ell \to r \ [\varphi] \in \mathcal{R} \cup \mathcal{R}_{calc}$ and σ respects φ

Example

- $\mathcal{R}_{calc}$ contains e.g.

$$x \wedge y \to z \quad [z = x \wedge y] \qquad\qquad x + y \to z \quad [z = x + y]$$

- for LCTRS $\mathcal{R}$:

$$(1)\ \ \mathsf{f}(x) \to \mathsf{a}\ [x \geqslant 0] \qquad (2)\ \ \mathsf{g}(\mathsf{f}(x), y) \to \mathsf{g}(x, z)\ [x \neq 0 \wedge z > x]$$

  - $\mathsf{f}(\mathsf{f}(3)) \to_{\mathcal{R}} \mathsf{f}(\mathsf{a})$ — $3 \geqslant 0$ valid
  - $\mathsf{g}(\mathsf{f}(1), \mathsf{a}) \to_{\mathcal{R}} \mathsf{g}(1, 23)$ — $1 \neq 0 \wedge 23 > 1$ valid
  - $\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{a})$ is NF — $x \mapsto \mathsf{a}$ not respectful for (2)
  - $\mathsf{g}(3 + 2) \to_{\mathcal{R}} \mathsf{g}(5)$ — $5 = 3 + 2$ valid

Application 1: Program equivalence

Two C programs: are they equivalent?

```
int sum1(int arr[],int n) {
  int ret=0;
  for(int i=0;i<n;i++)
    ret+=arr[i];
  return ret;
}
```

```
int sum2(int *arr, int n) {
  if (n <= 0) return 0;
  return arr[n-1] + sum2(arr, n-1);
}
```

- programs can be transformed into LCTRS $\mathcal{R}$ over theory of integer arithmetic and arrays

$sum1(a,n) \to u(a,n,0,0)$

$u(a,n,r,i) \to error\ [i < n \wedge (i < 0 \vee i \geq size(a))]$

$u(a,n,r,i) \to u(a,n,r+select(a,i),i{+}1)\ [i < n \wedge 0 \leq i < size($

$u(a,n,r,i) \to return(a,r)\ [i \geq n]$

$sum2(a,n) \to return(a,0)\ [n \leq 0]$

$sum2(a,n) \to error\ [n-1 \geq size(a)]$

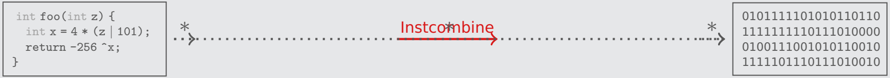
$s = t\ [\varphi]$ is inductive theorem if $s\sigma \leftrightarrow^*_{\mathcal{R}} t\sigma$ for all ground constructor substitutions σ that respect φ

- equivalence holds if $sum1(a,n) = sum2(a,n)\ [0 \leq n \leq size(a)]$ is inductive theorem ✓
- inductive theorem proving for LCTRS is implemented in tool **Ctrl**, based on

C. Fuhs, C. Kop, N. Nishida. Verifying Procedural Programs via Constrained Rewriting Induction. ACM Trans. Comput. Log. 18(2), 2017.

Application 2: Expression simplification in compilers

The Instcombine pass



$$\mathsf{mul}(x, C_1) \to \mathsf{shl}(x, C_2) \quad [\, \mathsf{log2}\ (C_1) = C_2 \wedge \mathsf{isPowerOf2}\ (C_1)\,]$$

- LLVM provides widely used compilation toolchain for various programming languages
- Instcombine pass performs >1000 **algebraic simplifications of expressions**: multiplications to shifts, reordering bitwise operations, . . .
- optimization set is community maintained, interference unclear: termination is crucial

Termination analysis via LCTRSs

each simplification can be modeled as LCTRS rewrite rule over bitvector theory

Definition

LCTRS is terminating if $\to_{\mathsf{rule}} \cup \to_{\mathsf{calc}}$ is well-founded

(Non-)Termination techniques for LCTRSs in Ctrl

- termination via DP framework and polynomial interpretations
  - C. Kop and N. Nishida. Constrained Term Rewriting tooL. Proc. 20th LPAR, pp. 549–557, 2015.
- non-termination via loops
  - N. Nishida and S. Winkler. Loop Detection by Logically Constrained Term Rewriting. Proc. 10th VSTTE, pp. 309–321, 2018.

Example (Loop in Instcombine simplification set)

rewrite rule

$$\mathsf{mul}(\mathsf{sub}(y,x),z) \to \mathsf{mul}(\mathsf{sub}(x,y),\mathsf{abs}(z)) \;\; [z < \mathbf{0}_8 \wedge \mathsf{isPowerOf2}(\mathsf{abs}(z))]$$

admits loop

$$\mathsf{mul}(\mathsf{sub}(\mathbf{1}_8,\mathbf{1}_8),(-\mathbf{128})_8) \to \mathsf{mul}(\mathsf{sub}(\mathbf{1}_8,\mathbf{1}_8),\mathsf{abs}((-\mathbf{128})_8))$$
$$\to_{\mathsf{calc}} \mathsf{mul}(\mathsf{sub}(\mathbf{1}_8,\mathbf{1}_8),(-\mathbf{128})_8)$$

Outline

1. Solution of Exercise of Session 3
2. Beyond Reduction Orders and Termination
3. Logically Constrained Term Rewrite Systems
4. **Further Reading**

Further Reading

- Akihisa Yamada
  Term Orderings for Non-reachability of (Conditional) Rewriting
  Proceedings IJCAR 2022, LNAI 13385, pp. 248−267, 2022
- Takahito Aoto
  Disproving Confluence of Term Rewriting Systems by Interpretation and Ordering
  Proceedings FroCoS 2013, LNAI 8152,pp. 311−326, 2013