1 Constraints for Argument Filterings*

Harald Zankl, Nao Hirokawa, and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, Austria
{Harald.Zankl,Nao.Hirokawa,Aart.Middeldorp}@uibk.ac.at

1.1 Introduction

Argument filterings are a key ingredient of the dependency pair method. Finding a suitable argument filtering that enables the constraints originating from the dependency pair method to be solved by a strictly monotone base order is a challenging search problem. In Section 1.2 we propose a simple encoding of argument filterings in propositional logic which can be easily combined with propositional encodings of simplification orders [1,3,5,6], resulting in a propositional formula with the property that any satisfying assignment corresponds to an argument filtering and the parameters of the encoded order which solve the constraints and vice-versa. We describe such a combination with the embedding order in Section 1.3. In Section 1.4 we mention some optimizations which reduce the size of the obtained propositional formulas. In order to test the effectiveness of our approach, we implemented this combination on top of the recursive SCC algorithm of [2]. For satisfiability checking we use two different methods, the state-of-the-art SAT solver MiniSat and a simple OCaml package for manipulating OBDDs. The results are compared with the divide and conquer algorithm implemented in T$_{\text{T}}$T and described in Section 1.5. In Section 1.6 we show how to recast a powerful result concerning argument filterings and usable rules [4] as a propositional formula, resulting in a free implementation.

1.2 Representing Argument Filterings

An *argument filtering* for a signature $\mathcal{F}$ is a mapping π that assigns to every n-ary function symbol $f \in \mathcal{F}$ an argument position $i \in \{1,\dots,n\}$ or a (possibly empty) list $[i_1,\dots,i_m]$ of argument positions with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. The signature $\mathcal{F}_\pi$ consists of all function symbols f such that $\pi(f)$ is some list $[i_1,\dots,i_m]$, where in $\mathcal{F}_\pi$ the arity of f is m. Every argument filtering π induces a mapping from $\mathcal{T}(\mathcal{F},\mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\pi,\mathcal{V})$, also denoted by π: $\pi(t) = t$ if $t \in \mathcal{V}$, $\pi(t) = \pi(t_i)$ if $t = f(t_1,\dots,t_n)$ with $\pi(f) = i$, and $\pi(t) = f(\pi(t_{i_1}),\dots,\pi(t_{i_m}))$ if $t = f(t_1,\dots,t_n)$ with $\pi(f) = [i_1,\dots,i_m]$.

Definition 1. Let $\mathcal{F}$ be a signature. The set of propositional variables $\{X_f \mid f \in \mathcal{F}\} \cup \{X_f^i \mid f^{(n)} \in \mathcal{F} \text{ and } 1 \leqslant i \leqslant n\}$ is denoted by $\mathcal{X}_\mathcal{F}$. Let π be an argument filtering for $\mathcal{F}$. The induced assignment α_π is defined as follows:

$$\alpha_\pi(X_f) = \begin{cases} \mathsf{true} & \text{if } \pi(f) = [i_1,\dots,i_m] \\ \mathsf{false} & \text{if } \pi(f) = i \end{cases} \qquad \text{and} \qquad \alpha_\pi(X_f^i) = \begin{cases} \mathsf{true} & \text{if } i \in \pi(f) \\ \mathsf{false} & \text{if } i \notin \pi(f) \end{cases}$$

for all n-ary function symbols $f \in \mathcal{F}$ and $i \in \{1,\dots,n\}$. Here $i \in \pi(f)$ if $\pi(f) = i$ or $\pi(f) = [i_1,\dots,i_m]$ and $i_k = i$ for some $1 \leqslant k \leqslant m$.

* This research is supported by FWF (Austrian Science Fund) project P18763.

Definition 2. An assignment α for $\mathcal{X}_\mathcal{F}$ is said to be *argument filtering consistent* if for every n-ary function symbol $f \in \mathcal{F}$ such that $\alpha \not\vDash X_f$ there is a unique $i \in \{1, \ldots, n\}$ such that $\alpha \vDash X_f^i$.

It is easy to see that α_π is argument filtering consistent.

Definition 3. The propositional formula $\mathsf{AF}(\mathcal{F})$ is defined as $\bigwedge_{f \in \mathcal{F}} \mathsf{AF}(f)$ with

$$\mathsf{AF}(f) = X_f \vee \bigvee_{i=1}^{\mathsf{arity}(f)} \big(X_f^i \wedge \bigwedge_{j \neq i} \neg X_f^j\big).$$

Lemma 1. *An assignment α for $\mathcal{X}_\mathcal{F}$ is argument filtering consistent if and only if $\alpha \vDash \mathsf{AF}(\mathcal{F})$.* $\square$

Definition 4. Let α be an argument filtering consistent assignment for $\mathcal{X}_\mathcal{F}$. The argument filtering π_α is defined as follows: $\pi_\alpha(f) = [i \mid \alpha \vDash X_f^i]$ if $\alpha \vDash X_f$ and $\pi_\alpha(f) = i$ if $\alpha \not\vDash X_f$ and $\alpha \vDash X_f^i$, for all function symbols $f \in \mathcal{F}$.

Example 1. Consider a signature consisting of two binary function symbols f and g. The assignment α with $\alpha(X_\mathsf{f}) = \alpha(X_\mathsf{f}^2) = \alpha(X_\mathsf{g}^1) = \mathsf{true}$ and $\alpha(X_\mathsf{f}^1) = \alpha(X_\mathsf{g}) = \alpha(X_\mathsf{g}^2) = \mathsf{false}$ is argument filtering consistent. The induced argument filtering π_α consists of $\pi_\alpha(\mathsf{f}) = [2]$ and $\pi_\alpha(\mathsf{g}) = 1$.

1.3 Embedding

In the following we define propositional formulas $\ulcorner s \rhd^\pi_\mathsf{emb} t \urcorner$ and $\ulcorner s \trianglerighteq^\pi_\mathsf{emb} t \urcorner$ which, in conjunction with $\mathsf{AF}(\mathcal{F})$, represent all argument filterings π that satisfy $\pi_\alpha(s) \rhd_\mathsf{emb} \pi_\alpha(t)$ and $\pi_\alpha(s) \trianglerighteq_\mathsf{emb} \pi_\alpha(t)$. We start with defining a formula $\ulcorner s =^\pi t \urcorner$ that represents all argument filterings which make s and t equal. (In the sequel we assume that $\wedge$ binds stronger than $\vee$.)

Definition 5. Let s and t be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We define a propositional formula $\ulcorner s =^\pi t \urcorner$ over $\mathcal{X}_\mathcal{F}$ by induction on s and t. If $s \in \mathcal{V}$ then

$$\ulcorner s =^\pi t \urcorner = \begin{cases} \top & \text{if } s = t, \\ \bot & \text{if } t \in \mathcal{V} \text{ and } s \neq t, \\ \neg X_g \wedge \bigvee_{j=1}^{m} \big(X_g^j \wedge \ulcorner s =^\pi t_j \urcorner\big) & \text{if } t = g(t_1, \ldots, t_m). \end{cases}$$

Let $s = f(s_1, \ldots, s_n)$. If $t \in \mathcal{V}$ then $\ulcorner s =^\pi t \urcorner = \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i =^\pi t \urcorner\big)$. If $t = g(t_1, \ldots, t_m)$ with $f \neq g$ then

$$\ulcorner s =^\pi t \urcorner = \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i =^\pi t \urcorner\big) \vee \neg X_g \wedge \bigvee_{j=1}^{m} \big(X_g^j \wedge \ulcorner s =^\pi t_j \urcorner\big).$$

Finally, if $t = f(t_1, \ldots, t_n)$ then

$$\ulcorner s =^\pi t \urcorner = \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i =^\pi t_i \urcorner\big) \vee X_f \wedge \bigwedge_{i=1}^{n} \big(X_f^i \to \ulcorner s_i =^\pi t_i \urcorner\big).$$

Definition 6. Let s and t be terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$. We define propositional formulas $\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner$ and $\ulcorner s \trianglerighteq^\pi_{\mathsf{emb}} t \urcorner = \ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner \vee \ulcorner s =^\pi t \urcorner$ over $\mathcal{X}_\mathcal{F}$ by induction on s and t. If $s \in \mathcal{V}$ then $\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner = \bot$. Let $s = f(s_1,\ldots,s_n)$. If $t \in \mathcal{V}$ then

$$\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner = X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \trianglerighteq^\pi_{\mathsf{emb}} t \urcorner\big) \vee \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \rhd^\pi_{\mathsf{emb}} t \urcorner\big).$$

If $t = g(t_1,\ldots,t_m)$ with $f \neq g$ then $\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner$ is the disjunction of

$$X_f \wedge \Big(X_g \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \trianglerighteq^\pi_{\mathsf{emb}} t \urcorner\big) \vee \neg X_g \wedge \bigvee_{j=1}^{m} \big(X_g^j \wedge \ulcorner s \rhd^\pi_{\mathsf{emb}} t_j \urcorner\big)\Big)$$

and $\neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \rhd^\pi_{\mathsf{emb}} t \urcorner\big)$. Finally, if $t = f(t_1,\ldots,t_n)$ then

$$\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner = X_f \wedge \Bigg(\bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \trianglerighteq^\pi_{\mathsf{emb}} t \urcorner\big) \vee \bigwedge_{i=1}^{n} \big(X_f^i \to \ulcorner s_i \trianglerighteq^\pi_{\mathsf{emb}} t_i \urcorner\big) \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \rhd^\pi_{\mathsf{emb}} t_i \urcorner\big)\Bigg) \vee \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \rhd^\pi_{\mathsf{emb}} t_i \urcorner\big).$$

The formula $\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner \wedge \mathsf{AF}(\mathcal{F})$ is satisfiable if and only if there exists an argument filtering π such that $\pi(s) \rhd_{\mathsf{emb}} \pi(t)$. Even stronger, $\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner \wedge \mathsf{AF}(\mathcal{F})$ encodes *all* argument filterings π that satisfy $\pi(s) \rhd_{\mathsf{emb}} \pi(t)$. Analogous statements hold for $\ulcorner s =^\pi t \urcorner \wedge \mathsf{AF}(\mathcal{F})$ and $\ulcorner s \trianglerighteq^\pi_{\mathsf{emb}} t \urcorner \wedge \mathsf{AF}(\mathcal{F})$.

Lemma 2. *Let $s,t \in \mathcal{T}(\mathcal{F},\mathcal{V})$. If α is an assignment for $\mathcal{X}_\mathcal{F}$ such that $\alpha \vDash \ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner \wedge \mathsf{AF}(\mathcal{F})$ then $\pi_\alpha(s) \rhd_{\mathsf{emb}} \pi_\alpha(t)$. If π is an argument filtering such that $\pi(s) \rhd_{\mathsf{emb}} \pi(t)$ then $\alpha_\pi \vDash \ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner \wedge \mathsf{AF}(\mathcal{F})$.* □

1.4 Optimizations of the Encoding

The formulas of Section 1.3 are written in a way to make them easily understandable for humans. Concerning efficiency however there are quite some useful optimizations which result in a large speedup. Consider e.g. the case of different function symbols in Definition 5. The original formula

$$\ulcorner s =^\pi t \urcorner = \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i =^\pi t \urcorner\big) \vee \neg X_g \wedge \bigvee_{j=1}^{m} \big(X_g^j \wedge \ulcorner s =^\pi t_j \urcorner\big)$$

can be expressed more concisely as

$$\ulcorner s =^\pi t \urcorner = \bigwedge_{i=1}^{n} \big(X_f^i \to \ulcorner s_i =^\pi t_i \urcorner\big)$$

since we know that $\mathsf{AF}(\mathcal{F})$ must hold anyway. Also the rules of commutativity, distributivity, etc. can drastically decrease the size of the formulas in Definition 6. As an example, note that the two formulas

$$\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner = X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \trianglerighteq^\pi_{\mathsf{emb}} t \urcorner\big) \vee \neg X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \rhd^\pi_{\mathsf{emb}} t \urcorner\big)$$

and

$$\ulcorner s \rhd^\pi_{\mathsf{emb}} t \urcorner = \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i \rhd^\pi_{\mathsf{emb}} t \urcorner\big) \vee X_f \wedge \bigvee_{i=1}^{n} \big(X_f^i \wedge \ulcorner s_i =^\pi t \urcorner\big)$$

are equivalent.

1.5 Experimental Results

We implemented the encoding of the previous section on top of the recursive SCC algorithm with the divide and conquer approach described in [2] for combining constraints in the termination prover T$_{\text{T}}$T. The generated propositional formulas are tested for satisfiability in two different ways: with the state-of-the-art SAT solver MiniSat after a linear translation (described in [5]) to CNF and with a simple package written in OCaml for manipulating OBDDs.[1] We also tested the effect of simplifying the formulas generated by the encoding before determining satisfiability, using a TRS consisting of 25 obvious simplification rules like $\neg\neg x \to x$, $x \wedge \top \to x$, and $x \wedge (x \vee y) \to x$. The results of our experiments are summarized in the table below. We used timeouts of 10 and 60 seconds for each of the 773 TRSs in the 2005 edition of the Termination Problem Data Base. All tests were performed on a server equipped with an Intel® Xeon$^{\text{TM}}$ processor running at a CPU rate of 2.40GHz and 512MB of system memory.

		SAT				OBDD			
	T$_{\text{T}}$T	××	×✓	✓×	✓✓	××	×✓	✓×	✓✓
timeout 10 seconds	5	1	3	0	0	3	3	0	0
total time (in seconds)	88	147	192	61	109	77	122	53	106
timeout 60 seconds	5	0	0	0	0	2	2	0	0
total time (in seconds)	338	151	197	61	109	212	257	53	106

The T$_{\text{T}}$T column refers to the divide and conquer algorithm described in [2]. The first character in the column headings ××, ×✓, ✓× and ✓✓ indicates whether the simplification rules were used and the second character whether the dynamic programming technique to select which constraints to combine next [2, Section 5.3] was used. (All methods succeeded in proving the termination of 188 of the 773 TRSs.)

Several preliminary conclusions can be drawn from the data. The performance of the approach proposed in this note outperforms the divide and conquer algorithm with dynamic programming implemented in T$_{\text{T}}$T, especially for large timeouts. The cost of performing simplifications cannot be ignored but is essential for large timeouts. The dynamic programming technique enables us to share and reuse solutions of constraints. This sounds perfectly suitable for an OBDD approach, but the experimental results do not confirm this. One reason is that the merge order of [2] that we adopted in the experiments is designed for *set* representations. As addressed in [2] we need a good strategy for merging solutions to keep representations small. We anticipate that by developing a suitable strategy for BDDs one will benefit from the dynamic programming technique.

1.6 Extensions

Our approach extends naturally to propositional encodings of other base orders [1,3,5,6]. A different and perhaps more interesting direction is to use the propositional framework to recast existing termination criteria in order to eliminate the often considerable effort to implement these criteria. Consider e.g. the following reformulation of a technique due to [4] for computing a restricted set of usable rules based on a given argument filtering.

Theorem 1. *Let $\mathcal{R}$ be a TRS and $\mathcal{C}$ a set of the dependency pairs. There is no $\mathcal{C}$-minimal rewrite sequence if there exist an argument filtering π and a $\mathcal{C}_\mathcal{E}$-compatible reduction pair $(\gtrsim, >)$ such that $\pi(\mathcal{U}(\mathcal{C}, \pi) \cup \mathcal{C}) \subseteq \gtrsim$ and $\pi(\mathcal{C}) \cap > \neq \varnothing$.* □

[1] Posted on the Caml mailing list by Andrzej Janikowski on October 19, 2005.

Rather than giving an explicit definition of the set $\mathcal{U}(\mathcal{C},\pi)$ we encode the constraint $\pi(\mathcal{U}(\mathcal{C},\pi)\cup\mathcal{C})\subseteq\gtrsim$ as the conjunction of

$$\bigwedge_{l\to r\in\mathcal{C}}\left(U_{\mathrm{root}(l)}\wedge\ulcorner l\gtrsim^{\pi} r\urcorner\right)\ \wedge\ \bigwedge_{l\to r\in\mathcal{R}}\left(U_{\mathrm{root}(l)}\to\ulcorner l\gtrsim^{\pi} r\urcorner\right)$$

and

$$\bigwedge_{l\to r\in\mathcal{R}\cup\mathcal{C}}\left(U_{\mathrm{root}(l)}\to\bigwedge_{\substack{p\in\mathcal{P}\mathrm{os}_{\mathcal{F}}(r)\\ \mathrm{root}(r|_p)\text{ is defined}}}\left(\bigwedge_{q,\,i\colon qi\leqslant p}X^{i}_{\mathrm{root}(r|_q)}\to U_{\mathrm{root}(r|_p)}\right)\right)$$

Here U_f is a new propositional variable for every defined and every dependency pair symbol f. So by simply adding to the above constraint the encodings of the other (side) conditions we get essentially for free an implementation of a more powerful usable rule criterion than the one currently implemented in T$_{\mathsf{T}}$T (which amounts to $\pi(\mathcal{U}(\mathcal{C})\cup\mathcal{C})\subseteq\gtrsim$).

Example 2. Let $\mathcal{R}$ be the TRS consisting of the two rules

$$\begin{aligned}\mathsf{sum}(x,[\,])&\to x & 0+y&\to y\\ \mathsf{sum}(x,y::z)&\to\mathsf{sum}(x+y,z) & \mathsf{s}(x)+y&\to\mathsf{s}(x+y)\end{aligned}$$

For the dependency pair $\mathsf{SUM}(x,y::z)\to\mathsf{SUM}(x+y,z)$ none of the rewrite rules is usable under an argument filtering π with $\pi(\mathsf{SUM})=[2]$ and the dependency pair simplifies to $\mathsf{SUM}(y::z)\to\mathsf{SUM}(z)$ which can be oriented by $\rhd_{\mathsf{emb}}$ from left to right. Exactly this observation is mirrored in the last conjunction of the advanced usable rules formula that suggests that if a rule is used ($U_{root(l)}$) then a defined symbol f occurring in the right hand side of the rule gives rise to further usable rules if this symbol f "remains" after applying the argument filtering. In the example we have the subformula $U_{\mathsf{SUM}}\to(X^1_{\mathsf{SUM}}\to U_+)$ which says that if the first argument of SUM is not deleted by the argument filtering then U_+ is set to *true* and $+$ gives rise to usable rules.

Concerning the experiments with this "advanced" usable rules criterion we could prove 59 additional TRSs terminating. For the two best performing combinations—SAT/OBDD with simplifications but without dynamic programming—the run times are 120 and 264 seconds with zero and one timeout (60 seconds) respectively.

References

1. M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. In *Proc. 17th RTA*, LNCS, 2006. To appear.
2. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
3. M. Kurihara and H. Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Proc. 17th IEA/AIE*, volume 3029 of *LNCS*, pages 827–837, 2004.
4. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proc. 2nd IJCAR*, volume 3097 of *LNAI*, pages 75–90, 2004.
5. H. Zankl. SAT techniques for lexicographic path orders. Seminar report, 2006. Available at `http://arxiv.org/abs/cs.SC/0605021`.
6. H. Zankl and A. Middeldorp. KBO as a satisfaction problem. Submitted to WST 2006.