A New Format for Rewrite Systems*

Takahito Aoto[1], Nao Hirokawa[2], Dohan Kim[3], Misaki Kojima[4], Aart Middeldorp[3], Fabian Mitterwallner[3], Naoki Nishida[4], Teppei Saito[2], Jonas Schöpf[3], Kiraku Shintani[2], René Thiemann[3], and Akihisa Yamada[5]

[1] Faculty of Engineering, Niigata University, Japan
[2] School of Information Science, JAIST, Japan
[3] Department of Computer Science, University of Innsbruck, Austria
[4] Department of Computing and Software Systems, Nagoya University, Japan
[5] National Institute of Advanced Industrial Science and Technology, Japan

Abstract

We propose a new format for a variety of rewrite systems, to replace the current COPS format used in the Confluence Competition. We include a proposal for logically constrained rewrite system, to prepare for a future competition category.

1 Introduction

The Confluence Competition (CoCo) [5][1] is an annual competition in which software tools try to solve confluence-related problems for a variety of rewrite formalisms. Problems in CoCo are selected from COPS [3],[2] an online database for confluence and related properties in term rewriting.

In the *basic* COPS format the rewrite rules of the problem at hand are specified and the variables are declared, but the function symbols are left implicit. This design decision goes back to the Termination and Complexity Competition (termCOMP) [1][3] and is based on the assumption that the property that needs to be (dis)proved is not affected by additional function symbols. This is true for most CoCo and termCOMP categories, but not for all. For instance, ground-confluence is well-known not to be preserved under signature extension. Since the corresponding GCR category employs the *many-sorted* COPS format, in which every function symbol is specified using a sort declaration, this causes no problem. Other properties that are known not to be closed under signature extension include NFP and UNR [4]. Also, termination under outermost strategies (corresponding to the termCOMP category TRS Outermost) is not preserved under signature extension [2]. For the corresponding categories in CoCo, besides the basic format, problems may be specified in the *extended* COPS format, which allows the declaration of additional function symbols.

We propose a change in the COPS format in which function symbols that can be used to construct terms must be listed, assuming that we have infinitely many variables (of any type) at our disposal, which is the usual convention in the term rewriting literature. In this paper we propose new formats for term rewrite systems (TRSs), conditional term rewrite systems (CTRSs), context-sensitive term rewrite systems (CSTRSs), a combination of the latter two (CSCTRSs), many-sorted term rewrite systems (MSTRSs), and logically constrained term rewrite systems (LCTRSs). For the latter, no previous format is known. We also describe how

*This research was funded by the Austrian Science Fund (FWF) project I5943 and JSPS-FWF JPJSBP120222001.

[1] http://project-coco.uibk.ac.at/
[2] https://cops.uibk.ac.at/
[3] http://termination-portal.org/wiki/Termination_Competition

multiple TRSs are represented. The new format specifies rewrite systems only, thus strategies like innermost and outermost are not part of the format. We take the stance that these should be part of competition categories, to avoid unnecessary duplication in databases.

2 Format

The following code represents the TRS $\{\mathsf{F}(x,x) \to \mathsf{A}, \mathsf{G}(x) \to \mathsf{F}(x,\mathsf{G}(x)), \mathsf{C} \to \mathsf{G}(\mathsf{C})\}$ in our new syntax:

```
; @author Takahito Aoto
; @author Junichi Yoshida
; @author Yoshihito Toyama
; @doi 10.1145/322217.322230
; p. 813, attributed to Barendregt

(format TRS)
(fun F 2)
(fun A 0)
(fun G 1)
(fun C 0)
(rule (F x x) A)
(rule (G x) (F x (G x)))
(rule C (G C))
```

The new format adopts a Lisp/Scheme-like syntax. All specifications are written in the form of S-expressions. A semicolon (`;`) indicates a line comment. The text between a semicolon and the end of the line is regarded as a comment. A line starting with `; @` indicates meta-information like information about authors and references.

All formats have an optional *meta-info* header and a *content* part. The *meta-info* part consists of line comments following a specific format, which are used to embed metadata in a file. They resemble key-value pairs, where the key is some arbitrary string not containing white space and the value is a string not containing newlines. In a meta-info line the key is written using a leading `@` symbol, separated from the value by a single space. For example in the line

```
; @author John Smith
```

the key `author` is followed by the value `John Smith`. Such metadata is used to attribute the problem to one or more authors or contributors, or cite the literature where the system first appeared. It is also easily extensible for future uses, for example to indicate that a specific property (confluence, termination, ...) is satisfied. The *content* part of the format represents the rewrite system, and depends on the type of system represented. The following (extended) BNF indicates the lexical and parsing rules for the common syntax of the new format:

$$
\begin{aligned}
\mathit{identifier} &::= [\texttt{\^{}␣\textbackslash t\textbackslash r\textbackslash n();:}]+ & \mathit{term} &::= \mathit{identifier} \mid \texttt{(}\ \mathit{identifier}\ \mathit{term}{+}\ \texttt{)}\\
\mathit{space} &::= [\texttt{␣\textbackslash t\textbackslash r\textbackslash n}] & \mathit{comment} &::= \texttt{;}\ [\texttt{␣\textbackslash t}]{*}\ (\epsilon \mid [\texttt{\^{}@\textbackslash n}]\ [\texttt{\^{}\textbackslash n}]{*})\ \texttt{\textbackslash n}\\
\mathit{number} &::= [\texttt{0} - \texttt{9}]+ & \mathit{meta\text{-}info} &::= \texttt{;}\ \texttt{␣}\ \texttt{@}\ [\texttt{\^{}\textbackslash n}]{*}\ \texttt{\textbackslash n}\\
\mathit{file} &::= \mathit{meta\text{-}info}{*}\ \mathit{content}\\
\mathit{content} &::= \mathit{TRS} \mid \mathit{CTRS} \mid \mathit{MSTRS} \mid \mathit{LCTRS} \mid \mathit{CSTRS} \mid \mathit{CSCTRS}
\end{aligned}
$$

Note that *space* and *comment* are ignored in the parsing rules. As illustrated by the introductory

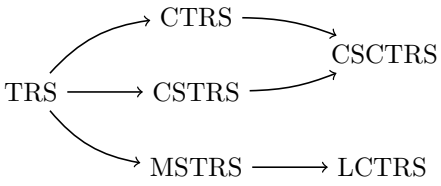

Figure 1: The formats ordered by inclusion.

example, function applications $f(t_1, \ldots, t_n)$, are written as `(`$f\ t_1\ \cdots\ t_n$`)`, while variables and constants are written without parentheses.

The relation between the formats is shown in Figure 1, where an arrow between formats means the target of the arrow is an extension of the source of the arrow.

2.1 TRS Format

The syntax of the TRS format is specified as follows:

$$\begin{aligned} \mathit{TRS} &::= \texttt{(\ format\ TRS\ )}\ \mathit{fun}{*}\ \mathit{rule}{*} \\ \mathit{fun} &::= \texttt{(\ fun}\ \mathit{identifier}\ \mathit{number}\ \texttt{)} \\ \mathit{rule} &::= \texttt{(\ rule}\ \mathit{term}\ \mathit{term}\ \texttt{)} \end{aligned}$$

Function symbols are declared by `(fun` f n`)` together with their arities. Undeclared identifiers are regarded as variables. The format does not exclude ill-formed TRSs such as $\{x \to \mathsf{f}(y)\}$. Validation of specifications is beyond the scope of our proposal.

2.2 CTRS Format

The format for CTRSs reuses the parsing rules for *fun* and *term*, while the remaining rules look as follows:

$$\begin{aligned} \mathit{CTRS} &::= \texttt{(\ format\ CTRS}\ \mathit{cond\text{-}type}\ \texttt{)}\ \mathit{fun}{*}\ \mathit{rule}{*} \\ \mathit{cond\text{-}type} &::= \texttt{oriented} \mid \texttt{join} \mid \texttt{semi-equational} \\ \mathit{rule} &::= \texttt{(\ rule}\ \mathit{term}\ \mathit{term}\ \mathit{cond}{*}\ \texttt{)} \\ \mathit{cond} &::= \texttt{(\ =}\ \mathit{term}\ \mathit{term}\ \texttt{)} \end{aligned}$$

For instance, the one-rule oriented CTRS $\{\mathsf{f}(x) \to \mathsf{f}(y) \Leftarrow \mathsf{g}(x) \approx z, \mathsf{g}(z) \approx \mathsf{h}(y)\}$ is specified as follows:

```
(format CTRS oriented)
(fun f 1)
(fun g 1)
(fun h 1)
(rule (f x) (f y) (= (g x) z) (= (g z) (h y)))
```

2.3 CSTRS format

To specify the replacement map, needed for context-sensitive rewriting, the *fun* parsing rule is extended with an optional argument:

$$\begin{aligned} \mathit{CSTRS} &::= \texttt{(}\ \texttt{format CSTRS}\ \texttt{)}\ \mathit{fun}{*}\ \mathit{rule}{*} \\ \mathit{fun} &::= \texttt{(}\ \texttt{fun}\ \mathit{identifier}\ \mathit{number}\ (\texttt{:replacement-map (}\ \mathit{number}{*}\ \texttt{)})?\ \texttt{)} \end{aligned}$$

and all other grammar rules are the same as in the TRS format. For example,

```
(fun f 3 :replacement-map (1 3))
```

indicates that the first and third argument positions of f are active. Declarations `(fun f` n`)` are equivalent to `(fun` f n `:replacement-map (1` $\cdots$ n`))`, so the full replacement map is default.

2.4 CSCTRS Format

To also allow conditional rules in context-sensitive TRSs the format below is the straightforward extension of the CTRS and CSTRS formats:

$$\begin{aligned} \mathit{CSCTRS} &::= \texttt{(}\ \texttt{format CSCTRS}\ \mathit{cond\text{-}type}\ \texttt{)}\ \mathit{fun}{*}\ \mathit{rule}{*} \\ \mathit{cond\text{-}type} &::= \texttt{oriented} \mid \texttt{join} \mid \texttt{semi-equational} \\ \mathit{fun} &::= \texttt{(}\ \texttt{fun}\ \mathit{identifier}\ \mathit{number}\ \texttt{:replacement-map (}\ \mathit{number}{*}\ \texttt{)}\ \texttt{)} \\ \mathit{rule} &::= \texttt{(}\ \texttt{rule}\ \mathit{term}\ \mathit{term}\ \mathit{cond}{*}\ \texttt{)} \\ \mathit{cond} &::= \texttt{(}\ \texttt{=}\ \mathit{term}\ \mathit{term}\ \texttt{)} \end{aligned}$$

2.5 MSTRS Format

The format for many-sorted TRSs adapts the TRS format as follows:

$$\begin{aligned} \mathit{MSTRS} &::= \texttt{(}\ \texttt{format MSTRS}\ \texttt{)}\ \mathit{sort}{*}\ \mathit{fun}{*}\ \mathit{rule}{*} \\ \mathit{sort} &::= \texttt{(}\ \texttt{sort}\ \mathit{identifier}\ \texttt{)} \\ \mathit{fun} &::= \texttt{(}\ \texttt{fun}\ \mathit{identifier}\ \mathit{type}\ \texttt{)} \\ \mathit{type} &::= \mathit{identifier} \mid \texttt{(}\ \texttt{->}\ \mathit{identifier}{+}\ \mathit{identifier}\ \texttt{)} \\ \mathit{rule} &::= \texttt{(}\ \texttt{rule}\ \mathit{term}\ \mathit{term}\ \texttt{)} \end{aligned}$$

For example, the $\{\mathsf{N}, \mathsf{L}\}$-sorted signature $\{\mathsf{nil} : \mathsf{N},\ \mathsf{cons} : \mathsf{N} \times \mathsf{L} \to \mathsf{L}\}$ is represented as follows:

```
(sort N)
(sort L)
(fun nil L)
(fun cons (-> N L L))
```

We adopt arrow notation `->` for function types, anticipating that a consistent notation will be used in a new format for higher-order rewrite systems. Function declarations such as `(fun cons 2)` are invalid in the MSTRS format. The sorts of variables used in many-sorted rewrite rules must be inferred from the rules separately. So the following specification is valid in the format.

```
(format MSTRS)
(sort N)
(sort L)
(fun f (-> N N))
(fun g (-> L L))
(rule (f x) x)
(rule (g x) x)
```

Here `x` in the first rule has sort `N` whereas `x` in the second rules has sort `L`.

2.6 LCTRS Format

The format for logically constrained TRSs extends that of many-sorted TRSs:

$$
\begin{aligned}
\textit{LCTRS} &::= \texttt{(}\ \texttt{format LCTRS}\ \texttt{)}\ \textit{smt-theory}*\ \textit{smt-def}*\ \textit{sort}*\ \textit{fun}*\ \textit{rule}* \\
\textit{smt-theory} &::= \texttt{(}\ \texttt{theory}\ \textit{theory}\ \texttt{)} \\
\textit{smt-def} &::= \texttt{(}\ \texttt{define-fun}\ \ldots\ \texttt{)} \\
\textit{rule} &::= \texttt{(}\ \texttt{rule}\ \textit{term}\ \textit{term}\ (\texttt{:guard}\ \textit{formula})?\ (\texttt{:var}\ \textit{vars})?\ \texttt{)} \\
\textit{vars} &::= \texttt{(}\ \textit{var}*\ \texttt{)} \\
\textit{var} &::= \texttt{(}\ \textit{identifier}\ \textit{identifier}\ \texttt{)}
\end{aligned}
$$

The *smt-theory* declaration specifies the theory symbols and types of the LCTRS theory part. This theory should be available in an off-the-shelf SMT solver. Sensible examples can be found on the SMT-LIB webpage.[4] Furthermore, *smt-def* must adhere to the `define-fun` command of SMT-LIB, and *formula* must be an SMT-LIB formula of the corresponding theory. Since SMT-LIB also adopts a syntax based on S-expressions, the lexical and parsing rules remain consistent. The optional sort declaration (`:var` *vars*) is mandatory in the case that sorts of variables cannot be inferred. For example, assuming the theory `Ints`, the variables `x` and `y` in the rule `(rule a b :guard (not (= x y)))` can be of sort Bool or Int. An example of an LCTRS in the new format is given below:

```
(format LCTRS)
(theory Ints)
(define-fun isEven ((x Int)) Bool (= (mod x 2) 0))
(sort NList)
(fun build (-> Int NList NList))     (fun nats (-> NList))
(fun cons  (-> Int NList NList))     (fun nil  (-> NList))
(rule nats (build 0 nil))
(rule (build n xs) (build (+ n 1) (cons n xs))
  :guard (and (isEven n) (>= n 0)))
(rule (build n xs) (build (+ n 1) xs)
  :guard (and (not (isEven n)) (>= n 0)) :var ((n Int) (xs NList)))
```

2.7 Multiple Rewrite Systems

Properties like relative termination and commutation rely on *two* rewrite systems over a common signature. To represent multiple rewrite systems, we use `:number` n in the format specification, where $n > 1$ specifies the number of rewrite systems. Rewrite rules have an optional

[4] https://smtlib.cs.uiowa.edu/theories.shtml

argument `:index` i where $1 \leqslant i \leqslant n$ specifies that the rule belongs to the i-th rewrite system. If the argument is not given, then $i = 1$. We give an example:

```
(format TRS :number 2)
(fun f 1)
(fun h 2)
(fun a 0)
(fun b 0)
(rule a (f b))
(rule (f a) b :index 1)
(rule (h a a) b :index 2)
(rule (f b) b :index 2)
```

This example specifies $\{\mathsf{a} \to \mathsf{f}(\mathsf{b}), \mathsf{f}(\mathsf{a}) \to \mathsf{b}\}$ as the first and $\{\mathsf{h}(\mathsf{a},\mathsf{a}) \to \mathsf{b}, \mathsf{f}(\mathsf{b}) \to \mathsf{b}\}$ as the second TRS over the common signature $\{\mathsf{a} : 0, \mathsf{b} : 0, \mathsf{f} : 1, \mathsf{h} : 2\}$.

3 Future Work

We are planning to adopt the new format in CoCo from 2024. Before that, COPS will be renewed. To support a smooth transition for tool developers, we will offer a tool[5] that converts problems in the old COPS formats to problems in the new syntax. Using the conversion tool, tools that do not support the new format can participate in CoCo with no effort.

COPS is not the only database for rewrite systems. The Termination Problem Database (TPDB) also hosts a large collection of rewrite systems. Having a unified database for rewrite systems with a uniform syntax would be beneficial to the rewriting community. Designing a new format for higher-order rewrite systems is future work.

Acknowledgements

We thank the anonymous reviewers for helpful comments.

References

[1] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In Tomáš Vojnar and Lijun Zhang, editors, *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11429 of *Lecture Notes in Computer Science*, pages 156–166, 2019.

[2] Bernhard Gramlich and Klaus Györgyfalvay. On modularity of termination properties of rewriting under strategies. In *Proceedings of the 12th International Workshop on Termination*, pages 59–63, 2012.

[3] Nao Hirokawa, Julian Nagele, and Aart Middeldorp. Cops and CoCoWeb: Infrastructure for confluence tools. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Proceedings of the 9th International Joint Conference on Automated Reasoning*, volume 10900 of *Lecture Notes in Artificial Intelligence*, pages 346–353, 2018.

[4] Aart Middeldorp. *Modular Properties of Term Rewriting Systems*. PhD thesis, Vrije Universiteit, Amsterdam, 1990.

[5] Aart Middeldorp, Julian Nagele, and Kiraku Shintani. CoCo 2019: Report on the Eighth Confluence Competition. *International Journal on Software Tools for Technology Transfer*, 23(6):905–916, 2021.

[5] A preliminary version of the tool is available at https://ari-informatik.uibk.ac.at/tools/conversion/