MKBOOL: A Multi-Completion System for Boolean Constrained Reduction Orders

Haruhiko Sato and Masahito Kurihara [*]

Abstract—**A multi-completion system MKB developed by Kurihara and Kondo accepts as input a set of reduction orders in addition to equations and efficiently simulates parallel processes each of which executes the standard completion procedure with one of the given orderings. In this paper, we describe a new system MKBOOL, which is a refinement of MKB in the sense that it restricts the reduction orders to those classes which can be represented by boolean constraints on some domains. Such classes include the recursive path orders (with status) for a finite signature. Our implementation represents a conjunction of the constraints by a binary decision diagram. The comprehensive experiments with the implementation on a set of well-known test problems show that in exchange for that restriction, MKBOOL runs more efficiently than MKB for most of the problems.**

Keywords: Term Rewriting System, Knuth-Bendix Completion, Multi-Completion, Recursive Path Ordering

1 Introduction

Term rewriting systems [1, 10] play an important role in various areas, such as automated theorem proving, functional and logic programming languages, and algebraic specification of abstract data types. In many applications, termination and confluence are crucially important properties of term rewriting systems. A term rewriting system which has both of these properties is said to be convergent.

In order to compute a convergent term rewriting system, the standard completion procedure (KB) was proposed by Knuth and Bendix [5] and has been widely improved since then. Given a term rewriting system R (or a set of equations E) and a reduction order which is used to check termination, the procedure verifies the convergence of R analytically. If R is not convergent, the procedure tries to construct a convergent term rewriting system which is equationally equivalent to R (or E) by adding or modifying rewrite rules. The result of the computation is either success, failure, or divergence [1, 2, 5, 10]. The existence of divergence implies that the procedure can be non-terminating, thus the procedure is a semi-algorithm.

The success of the procedure heavily depends on the choice of the appropriate reduction order to be supplied. Such a choise is often difficult for general users unless they have good insight in termination proof techniques. Unfortunately, one cannot try out potentially-appropriate reduction orders one by one (*sequentially*), because one of those runs may lead to indefinite, divergent computation and inhibit the exploration of the remaining possibilities.

This means that we have to consider, more or less, *parallel* execution of completion procedures each running with one of the orders. When the number of those orderings is very large, however, it is clear that the direct implementation of this idea on a fixed number of workstations would end up with serious inefficiently because of the large number of processes run on each machine.

Kurihara and Kondo [6] partially solved this problem by developing a completion procedure called MKB, which, accepting as input a *set* of reduction orders as well as equations, efficiently simulates (in a single process) parallel execution of KB procedures each working with one of those orders. The key idea is the development of the data structure for storing a pair $s : t$ of terms associated with the information to show which processes contain the rule $s \rightarrow t$ (or $t \rightarrow s$) and which processes contain the equation $s \leftrightarrow t$. This structure makes it possible to define a meta-inference system for MKB that effectively simulates a lot of closely-related inferences made in different processes all in a simgle operation. We call this type of procedure a *multi-completion* procedure.

In order to orient an equation $s \leftrightarrow t$, MKB needs to check $s \succ t$ for all orders $\succ$ one by one. It causes a significant reduction in performance when the number of orders is very large. To settle this problem, the new system MKBOOL presented in this paper represents a set of reduction orders by a boolean constraint on the basic objects (such as a precedence and a status) underlying the orders. Using this representation, the system can compute directly the set of all reduction orders $\succ$ such that $s \succ t$ for a given equation $s \leftrightarrow t$.

In this paper, we describe a general framework for MKBOOL, followed by a particular boolean encoding for recursive path orders (with status). The comprehensive experiments on a set of well-known test problems show that MKBOOL with this encoding runs more efficiently than MKB for most of the problems.

The paper is organized as follows. In Section 2 we briefly review the multi-completion procedure MKB. In Section 3 we propose a general framework for MKBOOL. In Section 4 we present a boolean encoding for recursive path orders, based on

[*]Graduate School of Information Science and Technology, Hokkaido University, Sapporo, 060-0814, Japan, haru@complex.eng.hokudai.ac.jp, kurihara@ist.hokudai.ac.jp

the encodings for the quasi-precedence and the status. In Section 5 we evaluate the performance of MKBOOL by comparing the experimental results with MKB. Finally, we conclude with future work in Section 6.

2 Multi-Completion

Given a set E of equations and a reduction order $\succ$, the standard completion procedure KB tries to compute a convergent set R of rewrite rules that is contained in $\succ$ and that induces the same equational theory as E.

Starting from the initial state $(E_0; R_0) = (E; \emptyset)$, the procedure obeys the inference system defined in Fig. 1 to generate a 'fair' sequence $(E_0; R_0) \vdash (E_1; R_1) \vdash \ldots$ of deduction, where $\rhd$ in the COLLAPSE rule is a well-founded order on terms such as the *encompassment* order. The result of the *successful* sequence is the empty set E_∞ of persisting equations and the convergent set R_∞ of rewrite rules.

A *multi-completion* procedure accepts as input a finite set $O = \{\succ_1, \ldots, \succ_m\}$ of reduction orders as well as a set E of equations. The mission of the procedure is basically the same as KB: it tries to compute a convergent set R of rewrite rules that is contained in *some* $\succ_i$ and that induces the same equational theory as E.

DELETE:	$(E \cup \{t \leftrightarrow t\}; R) \vdash (E; R)$
ORIENT:	$(E \cup \{t \leftrightarrow u\}; R) \vdash (E; R \cup \{t \to u\})$
	if $t \succ u$
SIMPLIFY:	$(E \cup \{t \leftrightarrow u\}; R) \vdash (E \cup \{t \leftrightarrow v\}; R)$
	if $u \to_R v$
COMPOSE:	$(E; R \cup \{t \to u\}) \vdash (E; R \cup \{t \to v\})$
	if $u \to_R v$
COLLAPSE:	$(E; R \cup \{u \to t\}) \vdash (E \cup \{v \to t\}; R)$
	if $l \to r \in R, u \to_{\{l \to r\}} v,$ and $u \rhd l$
DEDUCE:	$(E; R) \vdash (E \cup \{t \leftrightarrow u\}; R)$
	if $t \leftarrow_R v \to_R u$

Figure 1: Standard completion

The multi-completion procedure MKB developed in [6] exploits the data structure called nodes. Let $I = \{1, 2, \ldots, m\}$ be the set of indexes for orders $O = \{\succ_1, \ldots, \succ_m\}$. A *node* is a tuple $\langle t : u, L_1, L_2, L_3 \rangle$, where $t : u$ (called a *datum*) is an ordered pair of terms, and L_1, L_2 and L_3 (called *labels*) are subsets of I such that

- $L_1 \cap L_2 = L_2 \cap L_3 = L_3 \cap L_1 = \emptyset$ and
- $i \in L_1$ implies $t \succ_i u$, and $i \in L_2$ implies $u \succ_i t$.

The node $\langle t : u, L_1, L_2, L_3 \rangle$ is considered to be identical with $\langle u : t, L_2, L_1, L_3 \rangle$.

The MKB procedure is defined by the inference system working on a set N of nodes, as given in Fig.2. Starting from the initial set of nodes,

$$N_0 = \{\langle t : u, \emptyset, \emptyset, I \rangle | t \leftrightarrow u \in E\}$$

the procedure generates a fair sequence $N_0 \vdash N_1 \vdash \ldots$. From a successful sequence, the convergent set of rewrite rules can be extracted by projecting N_∞ onto a successful index i.

DELETE:	$N \cup \{\langle t : t, \emptyset, \emptyset, L \rangle\} \vdash N$ if $L \neq \emptyset$
ORIENT:	$N \cup \{\langle t : u, L_1, L_2, L_3 \cup L \rangle\} \vdash$
	$N \cup \{\langle t : u, L_1 \cup L, L_2, L_3 \rangle\}$
	if $L \neq \emptyset, L_3 \cap L = \emptyset,$
	and $t \succ_i u$ for all $i \in L$
REWRITE-1:	$N \cup \{\langle t : u, L_1, L_2, L_3 \rangle\} \vdash$
	$N \cup \left\{ \begin{array}{l} \langle t : u, L_1 \setminus L, L_2, L_3 \setminus L \rangle \\ \langle t : v, L_1 \cap F, \emptyset, L_3 \cap L \rangle \end{array} \right\}$
	if $\langle l : r, L, \ldots, \ldots \rangle \in N, u \to_{\{l \to r\}} v,$
	$u \doteq l,$ and $(L_1 \cup L_3) \cap L \neq \emptyset$
REWRITE-2:	$N \cup \{\langle t : u, L_1, L_2, L_3 \rangle\} \vdash N \cup$
	$\left\{ \begin{array}{l} \langle t : u, L_1 \setminus L, L_2 \setminus L, L_3 \setminus L \rangle \\ \langle t : v, L_1 \cap L, \emptyset, (L_2 \cup L_3) \cap L \rangle \end{array} \right\}$
	if $\langle l : r, L, \ldots, \ldots \rangle \in N, u \to_{\{l \to r\}} v,$
	$u \rhd l,$ and $(L_1 \cup L_2 \cup L_3) \cap L \neq \emptyset$
DEDUCE:	$N \vdash N \cup \{\langle t : u, \emptyset, \emptyset, L \cap L' \rangle\}$
	if $\langle l : r, L, \ldots, \ldots \rangle \in N,$
	$\langle l' : r', L', \ldots, \ldots \rangle \in N, L \cap L' \neq \emptyset,$
	$v \to_{\{l \to r\}} t,$ and $v \to_{\{l' \to r'\}} u$
GC:	$N \cup \{\langle t : u, \emptyset, \emptyset, \emptyset \rangle\} \vdash N$
SUBSUME:	$N \cup \left\{ \begin{array}{l} \langle t : u, L_1, L_2, L_3 \rangle, \\ \langle t' : u', L'_1, L'_2, L'_3 \rangle \end{array} \right\} \vdash$
	$N \cup \{\langle t : u, L_1 \cup L'_1, L_2 \cup L'_2,$
	$(L_3 \cup L'_3) \setminus (L_1 \cup L'_1 \cup L_2 \cup L'_2) \rangle\}$
	if $t : u$ and $t' : u'$ are the same
	(up to renaming of variables).

Figure 2: MKB inference rules

The semantics of MKB is based on the interpretation in which MKB simulates the parallel processes $P_1, \ldots, P_m$ with P_i executing the KB with the ordering $\succ_i$ and the *common* input E. The soundness, completeness, and fairness of MKB are discussed in [6].

3 Multi-Completion for Boolean Constrained Reduction Orders

In this section, we construct a general framework for a new multi-completion system MKBOOL on top of the basic framework of MKB by introducing boolean encoding for a set of reduction orders. In 3.1 we establish an abstract setting for boolean encoding for arbitrary sets. Then in 3.2 we use this setting to define MKBOOL.

3.1 Abstract Boolean Encoding

Let X be a set of *logical variables* and A_X be the set of all *assignments* (or interpretations) for X. A *constraint* F on X

is a restriction on the combinations of the values taken by the logical variables in X. Let C_X be the set of all constraints on X. The constraint F may be represented by a propositional formula (using the connectives such as $\neg, \wedge, \vee$) or a boolean function $F(X)$ (in an expression using the operators such as the negation $\bar{()}$, product$(\cdot)$, and sum$(+)$). We use these representations interchangeably in this paper.

Definition 3.1 *Let A be a set and X be a set of logical variables. An encoding function for A in X is a mapping* enc*: $A \rightarrow C_X$ such that*

- *for all $a \in A$, $enc(a)$ is satisfiable and*
- *for all $a, a' \in A$, if $a \neq a'$ then $enc(a) \wedge enc(a')$ is unsatisfiable.*

The decoding function dec*: $A_X \rightarrow A$ is derived as follows:*

$$dec(\alpha) = \begin{cases} a & \textit{if } \alpha \vDash enc(a), a \in A \\ \bot & \textit{otherwise} \end{cases}$$

Definition 3.2 *Let B be a subset of A. We extend the definition of the encoding function as follows:*

$$enc(B) = \bigvee_{b \in B} enc(b)$$

Let F be a constraint on X. We extend the definition of the decoding function as follows:

$$dec(F) = \{dec(\alpha) | \alpha \in A_X, \alpha \vDash F\}$$

Definition 3.3 *An* encoding *for a set A is a pair (X, enc) consisting of a set of logical variables X and an encoding function enc for A in X.*

Definition 3.4 *The* base constraint *of an encoding (X, enc) for a set A is defined by $enc(A)$.*

Lemma 3.1 *If (X_1, enc_1) is an encoding for A_1, (X_2, enc_2) is an encoding for A_2, and $(X_1 \cap X_2) = \emptyset$ then (X, enc) is an encoding for $A_1 \times A_2$ where*

$$X = X_1 \cup X_2$$
$$enc((a_1, a_2)) = enc_1(a_1) \wedge enc_2(a_2)$$
$$enc(A_1 \times A_2) = enc_1(A_1) \wedge enc_2(A_2).$$

3.2 MKBOOL

In the following, we consider encodings for a set of reduction orders. Let O be a set of reduction orders, $O_{t,u} = \{\succ \mid \succ \in O, t \succ u\}$, and (X, enc) be an encoding for O. We put some constraint $F_i \in C_X$ in each label field of the node structure of MKB. In this way, we define a new node structure

$$\langle t : u, F_1, F_2, F_3 \rangle$$

with constraint labels F_1, F_2 and F_3, in place of the MKB node structure $\langle t : u, L_1, L_2, L_3 \rangle$ with set labels, where $F_i = enc(L_i), (i = 1, 2, 3)$. The constraint labels of a node n is denoted by $F_1[n], F_2[n]$ and $F_3[n]$, respectively. The labels must satisfy the following conditions.

- $F_1 \wedge F_2, F_2 \wedge F_3, F_3 \wedge F_1$ are unsatisfiable.
- F_1 implies $enc(O_{t,u})$ and F_2 implies $enc(O_{u,t})$.

Clearly, the union and intersection of set labels are represented by the sum and product of the corresponding constraint labels, respectively; and the set difference $L_1 \setminus L_2$ is represented by $F_1 \cdot \overline{F_2}$. With these transformation in mind, we can rewrite MKB to the new system MKBOOL given in Fig. 3. Note that $enc(O_{t,u})$ is used in ORIENT to encode the set of reduction orders $\succ$ in which $t \succ u$.

$$\begin{array}{ll}
\text{DELETE:} & N \cup \{\langle t : t, 0, 0, F \rangle\} \vdash N \text{ if } F \neq 0 \\
\text{ORIENT:} & N \cup \{\langle t : u, F_1, F_2, F_3 \rangle\} \vdash \\
& N \cup \{\langle t : u, F_1 + F_3 \cdot F, F_2, F_3 \cdot \bar{F} \rangle\} \\
& \text{if } F = enc(O_{t,u}) \neq 0 \\
\text{REWRITE-1:} & N \cup \{\langle t : u, F_1, F_2, F_3 \rangle\} \vdash \\
& N \cup \left\{ \begin{array}{l} \langle t : u, F_1 \cdot \bar{F}, F_2, F_3 \cdot \bar{F} \rangle \\ \langle t : v, F_1 \cdot F, 0, F_3 \cdot F \rangle \end{array} \right\} \\
& \text{if } \langle l : r, F, \ldots, \ldots \rangle \in N, u \rightarrow_{\{l \rightarrow r\}} v, \\
& u \doteq l, \text{ and } (F_1 + F_3) \cdot F \neq 0 \\
\text{REWRITE-2:} & N \cup \{\langle t : u, F_1, F_2, F_3 \rangle\} \vdash N \cup \\
& \left\{ \begin{array}{l} \langle t : u, F_1 \cdot \bar{F}, F_2 \cdot \bar{F}, F_3 \cdot \bar{F} \rangle \\ \langle t : v, F_1 \cdot F, 0, (F_2 + F_3) \cdot F \rangle \end{array} \right\} \\
& \text{if } \langle l : r, F, \ldots, \ldots \rangle \in N, u \rightarrow_{\{l \rightarrow r\}} v, \\
& u \rhd l, \text{ and } (F_1 + F_2 + F_3) \cdot F \neq 0 \\
\text{DEDUCE:} & N \vdash N \cup \{\langle t : u, 0, 0, F \cdot F' \rangle\} \\
& \text{if } \langle l : r, F, \ldots, \ldots \rangle \in N, \\
& \langle l' : r', F', \ldots, \ldots \rangle \in N, F \cdot F' \neq 0, \\
& v \rightarrow_{\{l \rightarrow r\}} t, \text{ and } v \rightarrow_{\{l' \rightarrow r'\}} u \\
\text{GC:} & N \cup \{\langle t : u, 0, 0, 0 \rangle\} \vdash N \\
\text{SUBSUME:} & N \cup \left\{ \begin{array}{l} \langle t : u, F_1, F_2, F_3 \rangle, \\ \langle t' : u', F'_1, F'_2, F'_3 \rangle \end{array} \right\} \vdash \\
& N \cup \{\langle t : u, F_1 + F'_1, F_2 + F'_2, \\
& (F_3 + F'_3) \cdot \bar{F_1} \cdot \bar{F'_1} \cdot \bar{F_2} \cdot \bar{F'_2} \rangle\} \\
& \text{if } t : u \text{ and } t' : u' \text{ are the same} \\
& \text{(up to renaming of variables).}
\end{array}$$

Figure 3: MKBOOL inference rules

Given a set of equations E, we start from the initial set of nodes

$$N_0 = \{\langle t : u, 0, 0, enc(O) \rangle | t \leftrightarrow u \in E\}.$$

Given $\alpha \in A_X$, we define E- and R-projection from N to E and R as follows.

$$E[N,\alpha] = \bigcup_{n \in N} E[n,\alpha]. \quad R[N,\alpha] = \textstyle\bigcup_{n \in N} R[n,\alpha].$$

$$E[n,\alpha] = \begin{cases} \{t \leftrightarrow u\}, & \text{if } \alpha \vDash F_3 \\ \emptyset, & \text{otherwise.} \end{cases}$$

$$R[n,\alpha] = \begin{cases} \{t \rightarrow u\}, & \text{if } \alpha \vDash F_1 \\ \{u \rightarrow t\}, & \text{if } \alpha \vDash F_2 \\ \emptyset, & \text{otherwise.} \end{cases}$$

where $n = \langle t : u, F_1, F_2, F_3 \rangle$.

In order to check the success of the multi-completion, we introduce the following boolean constraint.

$$NoE[N] = enc(O) \cdot \prod_{n \in N} \overline{F_3[n]}.$$

Suppose that the deduction is fair in the sence defined in [6]. If $NoE[N] \neq 0$, there exists $\alpha \in A_X$ such that $\alpha \vDash NoE[N]$. Since this assignment satisfies $E[N,\alpha] = \emptyset$, the KB process working with the reduction order given by $dec(\alpha)$ has no equations. In this case, MKBOOL terminates and returns the convergent set of rewrite rules $R[N,\alpha]$.

MKBOOL can be more efficient than MKB for two reasons. First, in ORIENT rule MKB needs to repeat orientation check for each reduction order. Instead, MKBOOL calculates $enc(O_{t,u})$. If the set of reduction orders O is defined inductively on the structure of terms (like the recursive path orders), $enc(O_{t,u})$ can be also calculated inductively without enumerating the entire set O. Therefore, the computation time for ORIENT rule of MKBOOL only depends on the structure of the datum $t : u$, independent of the number of reduction orders. Second, the binary operations on two labels (such as $L_1 \cap L$ or $F_1 \cdot F$) can be more efficient for a similar reason. The time required for the operation depends on the structure of the constraint labels in MKBOOL. In practice, the amount of time for MKBOOL is far smaller than that for MKB, when the sizes of terms are small and the number of reduction orders to be considered is very large.

4 Encoding for Recursive Path Orders

The abstract framework for Boolean encoding for reduction orders given in the previous section is a generalization of Boolean encodings known in the literature, such as the atom-based encoding [7] and the symbol-based encoding [4] for the lexicographic path orders (LPO). The encoding for the Knuth-Bendix orders given in [11] is also seen as an instance of this framework.

In this section, we present an encoding for recursive path orders (RPO) [8] with quasi-precedence and status. First, we recall the definition of RPO.

Throughout the section, we let $\mathcal{F} \equiv \{f_1, \ldots, f_p\}, \mathcal{V}, \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a finite set of function symbols, a set of variables, and the set of terms constructed from $\mathcal{F}$ and $\mathcal{V}$, respectively.

4.1 Basic Definitions

Definition 4.1 *The set of permutations of $\{1, \ldots, n\}$ is denoted by $Per(n) = \{\pi_1^n, \ldots, \pi_{n!}^n\}$. (The superscript n is omitted if n is clear from the context.) The set of lexicographic statuses for function symbols with arity n is denoted by $Lex(n) = \{lex_\pi | \pi \in Per(n)\}$. The multiset status is denoted by mul. The set of all statuses for function symbols with arity n is denoted by $Status(n)$:*

$$Status(n) = \begin{cases} Lex(n) & \text{if } n \leq 1 \\ Lex(n) \cup \{mul\} & \text{if } n \geq 2 \end{cases}$$

Definition 4.2 *For a quasi-order $\succsim$ on terms, the quasi-order $\succsim^{s_1,s_2}$ is defined on sequences of terms as follows:*

$$\begin{aligned} &\langle t_1, \ldots, t_m \rangle \succsim^{mul,mul} \langle u_1, \ldots, u_n \rangle \\ &\iff \{t_1, \ldots, t_m\} \succsim_{mul} \{u_1, \ldots, u_n\} \\ &\langle t_1, \ldots, t_m \rangle \succsim^{lex_{\pi_1}, lex_{\pi_2}} \langle u_1, \ldots, u_n \rangle \\ &\iff \langle t_{\pi_1(1)}, \ldots, t_{\pi_1(m)} \rangle \succsim_{lex} \langle u_{\pi_2(1)}, \ldots, u_{\pi_2(n)} \rangle \end{aligned}$$

Definition 4.3 *A quasi-precedence $\succsim$ is a quasi-order on $\mathcal{F}$. A status function τ maps every $f \in \mathcal{F}$ to $Status(n)$ where n is the arity of f. Let t, u be terms. We write $t = f(t_1, \ldots, t_m)$, $u = g(u_1, \ldots, u_n)$ when they are not variables. We define the quasi-recursive path order $\succsim_{rpo}^\tau$ and the recursive path order $\succ_{rpo}^\tau$ as follows:*

$t \succsim_{rpo}^\tau u$ if and only if at least one of the following four conditions are satisfied:

- $t = u$
- $t_i \succsim_{rpo}^\tau u$ *for* $1 \leq \exists i \leq m$
- $f \succ g$ *and* $t \succ_{rpo}^\tau u_j$ *for* $1 \leq \forall j \leq n$
- $f \sim g$ *and* $t \succ_{rpo}^\tau u_j$ *for* $1 \leq \forall j \leq n$,
  and $((\tau(f) = mul$ *and* $\tau(g) = mul)$ *or*
  $(\tau(f) \in Lex(m)$ *and* $\tau(g) \in Lex(n)))$
  and $\langle t_1, \ldots, t_m \rangle \succsim_{rpo}^{\tau(f),\tau(g)} \langle u_1, \ldots, u_n \rangle$

$t \succ_{rpo}^\tau u$ if and only if $(t \succsim_{rpo}^\tau u) \wedge \neg(u \succsim_{rpo}^\tau t)$.

We will leave out the superscript τ, if it is understood. Let $Prec^{\mathcal{F}}$ be the set of all *total* quasi-precedences on $\mathcal{F}$. Let $Stat^{\mathcal{F}} \equiv Status(arity(f_1)) \times \cdots \times Status(arity(f_p))$ and $RPO^{\mathcal{F}} \equiv Prec^{\mathcal{F}} \times Stat^{\mathcal{F}}$. An element $(\succsim^{\mathcal{F}}, (s_1, \ldots, s_p)) \in RPO^{\mathcal{F}}$ corresponds to an RPO such that its quasi-precedence is $\succsim^{\mathcal{F}}$ and its status function maps f_i to s_i. In the following, we present encodings for $Prec^{\mathcal{F}}$ and $Stat^{\mathcal{F}}$.

4.2 Encoding for Precendence and Status

We present an encoding for $Prec^{\mathcal{F}}$ by following the approach of Codish *et al.* [4] who have proposed to assign an integer value to function symbols for representing the priority. A quasi-precedence $\succsim$ is defined in terms of the $priority$ by $f \succsim g$ if and only if $priority(f) \geq priority(g)$.

Definition 4.4 *Let $p = |\mathcal{F}|$ and $k = \lceil log_2(p) \rceil$. Then k is the number of bits required for encoding the priority per function symbol. Let $X^f_{prec} = \{b^f_i | 1 \leq i \leq k\}$ be the set of logical variables for encoding the priority of a function symbol f. The variable b^f_k represents the most significant bit. The set of variables for encoding the precedence is denoted by $X_{prec} = \bigcup_{f \in \mathcal{F}} X^f_{prec}$. We define the encoding function for the precedence as follows.*

$$enc_{prec}(\succsim) = \prod_{f,g \in \mathcal{F}, f \neq g, f \succsim g} P^k_{f,g},$$

$$P^k_{f,g} = \begin{cases} 1 & \text{if } k = 0 \\ b^f_k \bar{b}^g_k + (b^f_k b^g_k + \bar{b}^f_k \bar{b}^g_k) \cdot P^{k-1}_{f,g} & \text{if } k > 0 \end{cases}$$

Note that $P^k_{f,g} = 1$ if and only if $priority(f) \geq priority(g)$.

Lemma 4.1 *(X_{prec}, enc_{prec}) is an encoding for $Prec^{\mathcal{F}}$ and $enc_{prec}(Prec^{\mathcal{F}}) = 1$.*

Let us proceed to an encoding of $Status(n)$. If $n \leq 1$ then the encoding is trivial ($X = \emptyset$ and $enc(s) = 1$). From here, we assume $n \geq 2$.

Definition 4.5 *We introduce the set of variables X^f_{stat}.*

$$X^f_{stat} = \{x^f_{mul}\} \cup \{x^f_{ij} | 1 \leq i < j \leq n\}$$

Intuitively, a variable x^f_{mul} means that we use multiset comparison for arguments of f and $x^f_{ij}(i < j)$ or $\bar{x}^f_{ji}(j < i)$ means that the i-th argument is compared before the j-th argument in the lexicographical comparison of the arguments of f. We define the encoding function for the set $Status(arity(f))$ of the statuses of f as follows:

$$enc^f_{stat}(s) = \begin{cases} x^f_{mul} & \text{if } s = mul \\ \bar{x}^f_{mul} P^f_{\pi} & \text{if } s = lex_{\pi} \end{cases}$$

where

$$P^f_{\pi} = \prod_{k=1}^{n-1} \prod_{l=k+1}^{n} L^f_{\pi(k),\pi(l)}, \qquad L^f_{ij} = \begin{cases} x^f_{ij} & \text{if } i < j \\ \bar{x}^f_{ji} & \text{if } j < i \end{cases}$$

By Proposition 1, the encoding for $Stat^{\mathcal{F}}$ is derived from $(X^{f_1}_{stat}, enc^{f_1}_{stat}), \ldots, (X^{f_p}_{stat}, enc^{f_p}_{stat})$. We denote it by (X_{stat}, enc_{stat}), where $X_{stat} = \bigcup_{f \in \mathcal{F}} X^f_{stat}, enc_{stat} = \prod_{f \in \mathcal{F}} enc^f_{stat}$.

Lemma 4.2 *(X_{stat}, enc_{stat}) is an encoding for $Stat^{\mathcal{F}}$.*

4.3 Encoding for RPO

Finally, we present the inductive definition of $enc(RPO^{\mathcal{F}}_{s,t})$. Let (X, enc) be the encoding for $RPO^{\mathcal{F}}$ derived from (X_{prec}, enc_{prec}) and (X_{stat}, enc_{stat}) defined in the previous section, i.e., $X = X_{prec} \cup X_{stat}, enc = enc_{prec} \cdot enc_{stat}$. In the following definition, the constraint $RPO_{t,u}$ is true iff $t \succ_{rpo} u$ for some RPO $\succ_{rpo}$, and $QRPO_{t,u}$ is true iff $t \succsim_{rpo} u$ for some quasi-RPO $\succsim_{rpo}$.

Definition 4.6 *Let t, u be terms such that $t = f(t_1, \ldots, t_m), u = g(u_1, \ldots, u_n)$ when t and u are not variables and let $k = \lceil log_2(|\mathcal{F}|) \rceil$. We define the following constraints.*

$$RPO_{t,u} = QRPO_{t,u} \cdot \overline{QRPO}_{u,t}$$

$$QRPO_{t,u} =$$
$$\begin{cases} 0 \textit{ if } (t \in \mathcal{V}, t \neq u) \textit{ or } \mathcal{V} \ni u \notin \mathcal{V}ar(t) \\ 1 \textit{ if } t = u \textit{ or } \mathcal{V} \ni u \in \mathcal{V}ar(t) \\ \sum_{i=1}^{m} QRPO_{t_i,u} + P^k_{f,g} \cdot \prod_{j=1}^{n} RPO_{t,u_j} \cdot (\bar{P}^k_{g,f} \\ + \sum_{s_1 \in Status(arity(f))} \sum_{s_2 \in Status(arity(g))} EX^{s_1,s_2}_{t,u}) \\ \qquad \textit{if } t \neq u \textit{ and } t, u \notin \mathcal{V} \end{cases}$$

where

$$LEX_{\langle t_1,\ldots,t_m \rangle, \langle u_1,\ldots,u_n \rangle} =$$
$$\begin{cases} 1 \textit{ if } n = 0 \\ 0 \textit{ if } m = 0 \textit{ and } n > 0 \\ RPO_{t_1,u_1} + QRPO_{t_1,u_1} \cdot LEX_{\langle t_2,\ldots,t_m \rangle, \langle u_2,\ldots,u_n \rangle} \\ \qquad \textit{if } n > 0 \textit{ and } m > 0 \end{cases}$$

$$\begin{aligned} EX^{mul,lex_{\pi}}_{t,u} &= EX^{lex_{\pi},mul}_{t,u} = 0 \\ EX^{lex_{\pi_t},lex_{\pi_u}}_{t,u} &= enc^f_{stat}(lex_{\pi_t}) \cdot enc^g_{stat}(lex_{\pi_u}) \\ &\quad \cdot LEX_{\langle t_{\pi_t(1)},\ldots,t_{\pi_t(m)} \rangle, \langle u_{\pi_u(1)},\ldots,u_{\pi_u(n)} \rangle} \\ EX^{mul,mul}_{t,u} &= enc^f_{stat}(mul) \cdot enc^g_{stat}(mul) \\ &\quad \cdot \prod_{y \in N-M} \sum_{x \in M-N} QRPO_{x,y} \end{aligned}$$

where M and N are multiset: $M = \{t_1, \ldots, t_m\}, N = \{u_1, \ldots, u_n\}$.

Based on Lemmas 1, 2, and 3, we can define an encoding function for the recursive path order by $enc(\succ^{\tau}_{rpo}) = enc_{prec}(\succsim) \wedge enc_{stat}(\tau)$ where $\succsim$ and τ are the quasi-precedence and status function of $\succ^{\tau}_{rpo}$, respectively. In order to execute the ORIENT rule of MKBOOL, however, we need to compute $RPO^{\mathcal{F}}_{t,u}$, which is the set of all RPOs $\succ^{\tau}_{rpo}$ such that $t \succ^{\tau}_{rpo} u$. We also need to compute $enc(RPO^{\mathcal{F}})$, for starting MKBOOL. The following theorem gives the results. The proof is omitted.

Theorem 4.1 *$enc(RPO^{\mathcal{F}}_{t,u}) = RPO_{t,u}$ and $enc(RPO^{\mathcal{F}}) = enc_{stat}(Stat^{\mathcal{F}})$*

Table 1: Computation time of MKB and MKBOOL

Problem	3-1	3-3	3-4	3-5	3-6	3-7	3-20	3-28	3-29
# of orders	675	675	225	39	14607	34083	1722204423	4683	545835
MKB(sec)	20.7	16.7	435.0	3.9	88.0	188.6	>3600	47.9	359.3
MKBOOL(sec)	8.1	6.7	331.3	3.6	3.7	3.2	2.8	6.8	1.4

5 Implementation and Experiments

We have implemented MKBOOL and experimented on a set of the standard benchmark problems [9]. For example, the problem 3-1 is the most well-known problem from the group theory. For efficiency, we have used binary decision diagrams (BDDs) [3] as a representation for boolean constraints. The results are summarized in Table 1. The problems selected are all the problems that could be solved by the systems. We have considered all the RPOs with total quasi-precedence and status. The table gives the numbers of such orders and CPU times (in seconds) run on a PC with Pentium 4 CPU and 512MB main memory. Clearly, MKBOOL is more efficient than MKB in all the problems examined.

6 Conclusion

We have presented a new multi-completion system MKBOOL, based on the abstract framework for boolean constraints on reduction orders. In particular, we have presented an encoding for the recursive path orders with quasi-precedence and status. The experiments show that MKBOOL is more efficient than MKB in all the problems examined. As future work, we plan to incorporate the dependency pair method into multi-completion. We also plan to support more classes of reduction orders in addition to RPO.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[2] L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, Vol.C-35, No.8, pages 677-691, 1986.

[4] M. Codish, V. Lagoon and P. Stuckey. Solving partial order constraints for LPO termination. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 4-18, 2006.

[5] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. in J. Leech(ed.), *Computational Problems in Abstract Algebra*, pages 263-297, Pergamon Press, 1970.

[6] M. Kurihara and H. Kondo. Completion for multiple reduction orderings. *Journal of Automated Reasoning*, Vol.23, No.1, pages 25-42, 1999.

[7] M. Kurihara and H. Kondo. BDD encoding for partial order constraints and its application to expert systems in software verification domains. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, pages 2062-2067, 2000.

[8] J. Steinbach. Extensions and comparison of simplification orderings. in N. Dershowitz(ed.), *Proc. 3rd RTA*, volume 355 of *LNCS*, pages 434-448, 1989.

[9] J. Steinbach and U. Kühler. Check your ordering - termination proofs and problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.

[10] Terese. *Term rewriting systems*. Cambridge University Press, 2003.

[11] H. Zankl and A. Middeldorp. KBO as a satisaction problem. Technical Report, University of Innsbruck, 2006.