Submitted to the International Conference on Rewriting Techniques and Applications
http://rewriting.org/rta/

OPTIMIZING MKB$_{TT}$ (SYSTEM DESCRIPTION) *

SARAH WINKLER [1] AND HARUHIKO SATO [2] AND
AART MIDDELDORP [1] AND MASAHITO KURIHARA [2]

[1] Institute of Computer Science
University of Innsbruck, Austria

[2] Graduate School of Information Science and Technology
Hokkaido University, Japan

Abstract. We describe performance enhancements that have been added to mkb$_{TT}$, a modern completion tool combining multi-completion with the use of termination tools.

1. Introduction

The Knuth-Bendix completion tool mkb$_{TT}$ combines a multi-completion approach as introduced by Kurihara and Kondo [Kur99] with the use of automatic termination tools proposed by Wehrman, Stump and Westbrook [Weh06]. In this paper we present several performance enhancements which improved the first version described in [Sat08].

(1) Checking for and joining isomorphic processes results in considerable speedups for some input systems.
(2) Critical pair criteria were carried over to the context of multi-completion to reduce the number of nodes.
(3) Several indexing techniques were implemented to allow for a higher inference rate, namely path indexing, discrimination trees and code trees.
(4) Different selection strategies to choose the next process and node were compared.

The potential of these optimizations, which are described in the next four sections, is witnessed by the first automatic completion of the CGE$_4$ system (an axiomatization of group theory with four commuting endomorphisms) obtained with the new version of mkb$_{TT}$. The optimizations can be conveniently configured through the web interface, which is described in Section 6. The experimental results reported in Section 7 show their usefulness.

We start by recalling some basic definitions. A Knuth-Bendix completion (KB) procedure takes a set of equations together with a reduction order as input and aims to produce a terminating and confluent rewrite system with the same equational theory. We call a KB

Key words and phrases: Knuth-Bendix completion, termination prover, automated deduction.
*This research is supported by FWF (Austrian Science Fund) project P18763. The first author is supported by a DOC-fFORTE fellowship of the Austrian Academy of Sciences.

run *fair* if all critical pairs among persistent rewrite rules are eventually considered. In Knuth-Bendix completion with termination tools (KBtt), a reduction order is no longer required. Instead, the inference system of KB is extended to work on tuples (E, R, C) of a set of equations E and rewrite systems R and C, which we refer to as KBtt *states*. The inference system $\mathsf{mkb}_{\mathrm{TT}}$ simulates multiple KBtt runs. Each run is identified by a bit sequence called a *process*. The inference rules of $\mathsf{mkb}_{\mathrm{TT}}$ operate on objects called *nodes*. A node has the form $\langle s : t, R_0, R_1, E, C_0, C_1\rangle$ where s, t are terms and the remaining components (called *labels*) are sets of processes. In the sequel we assume familiarity with [Sat08, Sat09].

2. Isomorphisms on Processes

The number of parallel processes simulated by $\mathsf{mkb}_{\mathrm{TT}}$ is critical for overall performance. However, some systems exhibit process pairs which are very similar and actually equally likely to succeed. This is illustrated in the following example.

Example 2.1. When running $\mathsf{mkb}_{\mathrm{TT}}$ on CGE_2 [Stu06], a process p with state

$$E_p = \left\{ \begin{array}{c} (x * y) * z \approx x * (y * z) \\ \mathsf{f}(\mathsf{e}) \approx \mathsf{e} \\ \mathsf{g}(\mathsf{e}) \approx \mathsf{e} \\ \mathsf{g}(x) * \mathsf{f}(y) \approx \mathsf{f}(y) * \mathsf{g}(x) \end{array} \right\} \qquad R_p = C_p = \left\{ \begin{array}{c} \mathsf{e} * x \to x \\ \mathsf{g}(x) * x \to \mathsf{e} \\ \mathsf{f}(x * y) \to \mathsf{f}(x) * \mathsf{f}(y) \\ \mathsf{g}(x * y) \to \mathsf{g}(x) * \mathsf{g}(y) \end{array} \right\}$$

has to orient the equation $\mathsf{g}(x) * \mathsf{f}(y) \approx \mathsf{f}(y) * \mathsf{g}(x)$. As both orientations are possible the process will be split, but the states (E_{p0}, R_{p0}, C_{p0}) and (E_{p1}, R_{p1}, C_{p1}) of the resulting child processes are the same up to interchanging f and g. Hence the deductions corresponding to these processes will be symmetric if the choice function is reasonably fair.

The following definition formally captures such similarities in general.

Definition 2.2. A mapping $\theta\colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ induces an *isomorphism* between two rewrite systems R and R' if $R' = \{\theta(l) \to \theta(r) \mid l \to r \in R\}$ and for all terms s and t, $s \to_R t$ if and only if $\theta(s) \to_{R'} \theta(t)$. Similarly, θ induces an isomorphism between two sets of equations E and E' if $E' = \{\theta(u) \approx \theta(v) \mid u \approx v \in E\}$ and for all terms s and t, $s \approx_E t$ if and only if $\theta(s) \approx_{E'} \theta(t)$.

Two rewrite systems R and R' are *isomorphic* if there exists an isomorphism θ between them, which is expressed by writing $R \cong_\theta R'$. Two KBtt states (E, R, C) and (E', R', C') are isomorphic if there is an isomorphism θ such that $E \cong_\theta E'$, $R \cong_\theta R'$, and $C \cong_\theta C'$. Two $\mathsf{mkb}_{\mathrm{TT}}$ processes p and q are isomorphic in a node set N if their projected states $(E(N,p), R(N,p), C(N,p))$ and $(E(N,q), R(N,q), C(N,q))$ are isomorphic.

Isomorphic processes are equally likely to succeed, which is easy to prove as the rewrite relations coincide.

Lemma 2.3. *Assume N is a node set with isomorphic processes p and q. If there exists a fair mkb_{TT} completion run $N \vdash^* N'$ such that $E(N', p) = \varnothing$ then there is another fair deduction $N \vdash^* N''$ such that $E(N'', q) = \varnothing$.*

Due to Lemma 2.3, completeness is not compromised if one of two isomorphic processes is removed. $\mathsf{mkb}_{\mathrm{TT}}$ exploits such symmetries by checking for two concrete shapes of isomorphisms.

– *Renamings* swap function symbols as in Example 2.1 where $p0$ and $p1$ are isomorphic processes under the mapping θ that exchanges f and g.
– *Argument permutations* associate with every function symbol f of arity n a permutation $\pi_f \in \mathcal{S}_n$. Then the mapping on terms defined by $\theta(x) = x$ and $\theta(f(t_1, \ldots, t_n) = f(\theta(t_{\pi_f(1)}), \ldots, \theta(t_{\pi_f(n)}))$ may also induce an isomorphism. For example, during a completion run of SK3.02 [Ste90] a process p with state

$$E_p = \{(x+y)+z \approx x+(y+z)\} \qquad R_p = C_p = \left\{ \begin{array}{l} \mathsf{f}(\mathsf{f}(x)) \to x \\ \mathsf{f}(x+y) \to \mathsf{f}(x)+\mathsf{f}(y) \end{array} \right\}$$

has to orient the associativity axiom. Again both orientations are possible, but the two child processes emerging from a process split are isomorphic under an argument permutation satisfying $\pi_+ = (1\,2)$.

$\mathsf{mkb}_{\mathrm{TT}}$ can check for both kinds of isomorphisms, either only in orient inferences to avoid unnecessary splittings or by comparing the states of all process pairs repeatedly.

3. Indexing Techniques

For rewrite inferences $\mathsf{mkb}_{\mathrm{TT}}$ needs to filter out nodes from the current node set that can be used to rewrite a given node. In the case of deduce inferences one needs to find nodes that overlap with the current node. Especially rewrite inferences are frequently executed. Performing these operations efficiently is crucial for overall performance.

In automated reasoning this problem is referred to as the *indexing problem*: Given a large set L of terms (the *index*), a binary relation on terms R (the *retrieval condition*) and a term t (the *query term*), find all $s \in L$ such that $(s, t) \in R$. For this purpose, a number of sophisticated term indexing techniques have been developed [Sek01].

In the case of $\mathsf{mkb}_{\mathrm{TT}}$, for $\mathsf{rewrite}_1$ the retrieval of variants, for $\mathsf{rewrite}_2$ finding encompassments, and for deduce the search of unifiable terms is required. Since encompassment retrieval can be implemented as multi-term retrieval of subsumed terms, indexing structures need to support variance, subsumption and unifiability as retrieval conditions. Variant and encompassment retrieval is required particularly often and consumes about 25% of the total computation time if performed naively.

$\mathsf{mkb}_{\mathrm{TT}}$ currently supports *path indexing* [McC92, Gra96] to retrieve unifiable terms. For variant and generalization retrieval, also *discrimination trees* [McC92, Gra96] and *code trees* [Vor95] are implemented.

In line with earlier observations in automated reasoning, the use of indexing techniques in $\mathsf{mkb}_{\mathrm{TT}}$ increases performance considerably.

4. Critical Pair Criteria

Maintenance and treatment of equational consequences derived in a deduction is a critical factor in many automated reasoning tools. Since it keeps track of multiple processes this is an even more serious issue for $\mathsf{mkb}_{\mathrm{TT}}$. For standard completion several *critical pair criteria* were proposed as a means to filter out equational consequences that can be ignored without losing completeness.

In the setting of $\mathsf{mkb}_{\mathrm{TT}}$, also the computation of critical pair criteria can be shared among multiple processes. If a node with datum $s : t$ is deduced from an overlap o for a

process set E then $\mathrm{CPC}(o, E, N)$ returns all processes for which $s \approx t$ is not superfluous. Thus the deduce rule is modified as follows.

Definition 4.1. The inference rule

$$\mathsf{deduce} \quad \frac{N}{N \cup \{\langle s : t, \varnothing, \varnothing, E, \varnothing, \varnothing\rangle\}}$$

is applicable if there exist nodes $\langle l : r, R, \dots\rangle, \langle l' : r', R', \dots\rangle \in N$ such that $s \approx t$ is a critical pair originating from an overlap o involving the rules $l_1 \to r_1$ and $l_2 \to r_2$, and $E = \mathrm{CPC}(o, R \cap R', N) \neq \varnothing$.

Given a critical pair $s \approx t$ originating from an overlap $(l_1 \to r_1, p, l_2 \to r_2)_\sigma$, all implemented criteria have in common that the term $l_1\sigma = l_1\sigma[l_2\sigma]$ is checked for being reducible in a different way than by $l_1 \to r_1$ and $l_2 \to r_2$. Consequently, all criteria need to filter the current node set N for nodes that allow to rewrite $l_1\sigma$ in an appropriate way (depending on the actual criterion). The execution of a CPC function can thus be shared among multiple processes. Moreover, CPC functions can take advantage from term indexing techniques as they require to find encompassments for a given term.

The criteria implemented in $\mathsf{mkb}_{\mathrm{TT}}$ are multi-completion variants of the criteria developed for standard completion: the primality criterion PCP [Kap88], the blocked criterion BCP [Bac88], and the connectedness criterion CCP [Küc85]. The projection of an $\mathsf{mkb}_{\mathrm{TT}}$ deduction using a critical pair criterion to a process p yields a KBtt deduction which is fair with respect to the criterion. Being specializations of the more general compositeness criterion [Bac94], the implemented criteria are *correct*. This means that a nonfailing deduction which is fair with respect to a critical pair criterion is also fair in the more general sense. Hence, according to the definition of fairness in $\mathsf{mkb}_{\mathrm{TT}}$ [Sat09, Section 4], deductions using critical pair criteria are also fair, so correctness is preserved. Experimental results evaluating the usefulness of the critical pair criteria are given in Section 7.

5. Selection Strategies

At the beginning of $\mathsf{mkb}_{\mathrm{TT}}$'s main control loop, a *choice* function selects a node to process next by evaluating a cost heuristic. The measure applied in this selection has significant impact on performance. To allow for a variety of strategies, we defined a language that is general enough to cover selection strategies that proved to be useful, but also allows to capture some concepts used in selection strategies of other automated reasoning tools. A selection strategy is thus specified by the following grammar:

strategy	::=	`?` | `(`*node_p*`,`*strategy*`)` | *float*`(`*strategy*`:`*strategy*`)`
node_p	::=	`data(`*termpair_p*`)` | `el(`*processset_p*`)` | `-` *node_p* | *node_p* `+` *node_p* | `*`
processset_p	::=	`min(`*process_p*`)` | `sum(`*process_p*`)` | `#`
process_p	::=	*process_p* `+` *process_p* | `e(`*eqs_p*`)` | `r(`*trs_p*`)` | `c(`*trs_p*`)`
eqs_p	::=	`sum(`*termpair_p*`)` | `#`
trs_p	::=	`sum(`*termpair_p*`)` | `cp(`*eqs_p*`)` | `#`
termpair_p	::=	`smax` | `ssum`

The following paragraphs shortly comment on the elements of selection strategies.

- The simplest strategy `?` chooses a node randomly. Otherwise, a strategy can be a tuple `(`*np*`,`*s*`)` consisting of a node property *np* and another strategy *s*. By using this rule multiple times, a node property tuple of the form `(`np_1`,` $\dots$ `(`np_k`,?)` $\dots$ `)`

is obtained. A selection strategy is implemented by mapping each node to the corresponding cost tuple of integers and choosing the (lexicographic) minimum. To mix strategies, a strategy can also be of the shape $r(s_1 : s_2)$. Here r is assumed to be a rational value in $[0, 1]$, with the intention that in every selection step the strategy s_1 is applied with probability r, and s_2 is used in the remaining cases.

- Node properties capture features of nodes with integers. A node property of a node $n = \langle s : t, \ldots, E, \ldots \rangle$ can be its creation time (denoted by `*`), a property of the node's datum $s : t$, or a process set property pp of its equation labels E, which is written as `el(`pp`)`. Moreover, node properties can be added or inverted.
- A process set property may be the number of processes it contains (denoted by `#`) or the sum or the minimum over a property of the processes it contains.
- A process property pp of a process p may be an equation set property of its equation projection $E(N, p)$ or a TRS property of either its rule projection $R(N, p)$ or its constraint projection $C(N, p)$, which is expressed by writing `e(`pp`)`, `r(`pp`)` and `c(`pp`)`, respectively. In addition, process properties can be added.
- An equation set property of a set of equations E can be its cardinality (`#`), or the sum over a term pair property of all its elements. A TRS property of a rewrite system R can additionally be a property of its critical pairs $\mathrm{CP}(R)$.
- Finally, a term pair property of terms s and t can be the sum $|s| + |t|$ or maximum $\max\{|s|, |t|\}$ of their sizes.

Many automated reasoning tools [Vor01] employ a size-age ratio when selecting a fact to be processed next. For example, if this ratio is 2 : 3 then out of 5 selections 2 will pick the oldest and 3 the smallest node, i.e., the node where the sum of its term sizes $|s| + |t|$ is minimal. In $\mathsf{mkb}_{\mathrm{TT}}$ a parameter $r \in [0, 1]$ controls the ratio of weight-determined selections, i.e., an age-weight ratio of 2 : 3 would correspond to $r = 0.6$. This is described with the following strategy:

$$s_{\texttt{size/age}}(r) = r\texttt{((data(ssum),?):(*,?))}$$

In the first version of $\mathsf{mkb}_{\mathrm{TT}}$ [Sat08] we first chose a process p for which $|E(N, p)| + |R(N, p)|$ was minimal and then a node for this process by considering the term size and timestamp, the latter to ensure fairness of the derivation. By again using a size-age ratio in the second step, this is captured by the strategy

$$s_{\texttt{mkbtt1}}(r) = \texttt{(el(min(e(\#)+r(\#))),}s_{\texttt{size/age}}(r)\texttt{)}$$

The selection approach used in Slothrop [Weh06] corresponds to choosing a process for which $|E(N, p)| + |\mathrm{CP}(R(N, p))| + |C(N, p)|$ is minimal, which is expressed as follows:

$$s_{\texttt{slothrop}}(r) = \texttt{(el(min(e(\#)+r(cp(\#))+c(\#))),}s_{\texttt{size/age}}(r)\texttt{)}$$

We recently experimented with an approach which first restricts attention to those processes where the number of symbols in $E(N, p)$ and $C(N, p)$ is minimal, then selects nodes with minimal data and finally goes for a node which has the greatest number of processes in its equation label:

$$s_{\texttt{sum}} = \texttt{(el(min(e(sum(ssum))+c(sum(ssum)))),(data(ssum),(-el(\#),?))))}$$

The strategy where `ssum` is replaced by `smax` is referred to as $s_{\texttt{max}}$. To use other heuristics than those just described, the strategy can also be specified via a command line option.

6. Web Interface

In addition to a command line interface, mkb$_{\mathrm{TT}}$ is now also available via a web interface which allows to configure various options.

- The user can set both a global timeout and a timeout for each termination check.
- Concerning termination checks, users can either apply TTT2 [Kor09] or incorporate an external tool. If TTT2 is used internally, different predefined termination strategies can be selected. This includes basic reduction orderings as well as `ttt2micro` and `ttt2fast`, two powerful and fast strategies. Alternatively, a user-defined strategy may be supplied in the strategy language of TTT2. Another option allows to check termination externally with AProVE [Gie06] or MuTerm [Ala07].
- For the retrieval of encompassments and variants, one of the implemented indexing techniques can be selected (path indexing, discrimination trees, code trees or naive search in the node set).
- To filter deduced equations, one of the three implemented critical pair criteria PCP, BCP or CCP can be selected, as well as certain combinations.
- Users can choose if isomorphism checks employing symbol renamings or term permutations are to be used, and whether these checks are performed repeatedly or only when processes are split.

The screenshot in Figure 1 gives an impression of the web interface, which is accessible at

`http://colo6-c703.uibk.ac.at/mkbtt/interface/`

7. Experimental Results

All of the following experiments were performed on a single core of a server equipped with eight dual-core AMD Opteron® processors 885 running at a clock rate of 2.6GHz and 64GB of main memory. As a test set we used 101 problems collected from various papers. These include theories underlying unit equality problems in TPTP 3.6.0 [Sut09], all examples coming with the Slothrop [Weh06] distribution, all systems in [Ste90, Section 3], instances of parametric systems given in [Bün94, Chr89], and systems describing commuting group endomorphisms [Stu06]. The whole testbench as well as the full experimental data can be obtained from the website.

In its fastest configuration it took the previous version [Sat08] of mkb$_{\mathrm{TT}}$ 175 seconds to complete CGE$_2$, and more than 3000 seconds to complete CGE$_3$. The implemented optimizations, in particular isomorphisms and different selection strategies, allowed for significant speedups: Using $s_{\mathtt{max}}$ as selection strategy, `ttt2micro` for termination checks and code trees for indexing operations, completing CGE$_2$ and CGE$_3$ requires 8.4 and 184 seconds, respectively. Using periodical checks for renaming isomorphisms further reduces these numbers to 4.7 and 33 seconds. The implemented optimizations even allow mkb$_{\mathrm{TT}}$ to complete CGE$_4$, which was neither achieved with the previous version nor with any other approach we know of. With the same settings as described above, a complete system is obtained in 622 seconds.

The current version of mkb$_{\mathrm{TT}}$ can also produce a canonical presentation for the proof reduction system presented in [Weh05]. Using `ttt2micro` as termination strategy and $s_{\mathtt{sum}}$ to control node selections, a complete system is obtained in 115 seconds. According to [Weh05] Waldmeister [Löc02] produces a ground-complete system for this theory. Since

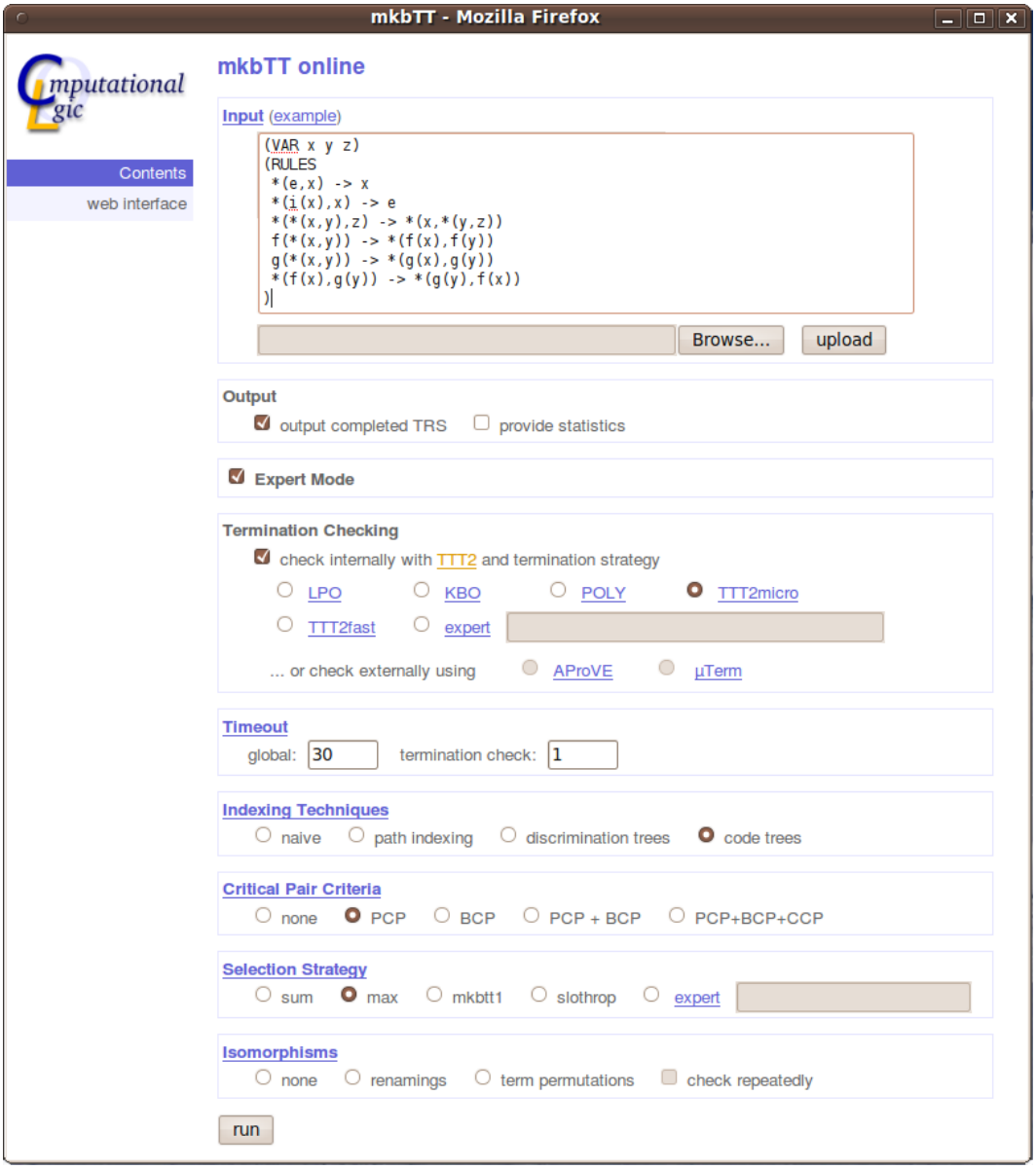

Figure 1: Web interface of $\mathsf{mkb}_{\mathrm{TT}}$.

the resulting system is not terminating with one of the reduction orders implemented in Waldmeister, it is not clear whether it can produce a complete system for this theory.

The following paragraphs present experimental results for each of the optimizations presented in the previous sections. In all of the following examples, $\mathsf{mkb}_{\mathrm{TT}}$ performs termination checks by interfacing TTT2 internally, using `ttt2micro` as termination strategy. The global timeout and the timeout for each termination check were set to 600 and 10 seconds, respectively. If not stated otherwise, we used $s_{\mathtt{max}}$ to control node selections, code trees for indexing operations and neither critical pair criteria nor isomorphisms.

none		rename		rename+		permutations		permutations+	
(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
188	25	188	25	186	21	188	25	188	25

Table 1: Isomorphisms.

naive			path indexing			discrimination trees			code trees		
(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
19.9	387	91	18.9	345	18	16.5	150	5	15.8	106	5

Table 2: Indexing techniques.

Isomorphisms. The results obtained with $\mathsf{mkb}_{\mathrm{TT}}$ using isomorphism checks are given in Table 1. The columns list (1) the average time in seconds and (2) the average number of processes. As witnessed by the CGE examples, renaming isomorphisms can greatly improve performance on systems with symmetries. Also when tested on the whole database, repeatedly checking for renamings (rename+) pays off slightly. Apparently the advantage gained on systems with isomorphic processes outweighs the overhead required for the remaining input problems. Argument permutations, on the other hand, have little effect. For example, when completing the systems LS94_G0 and GRP012-4th coming from group theory, the number of processes drops from 8 to 4 and 4 to 2 using argument permutations. That also affects the time required for their completion. But since these examples are small and easily completable in any case, the overall performance is not improved.

Indexing Techniques. Table 2 presents results obtained with $\mathsf{mkb}_{\mathrm{TT}}$ using different indexing techniques for rewriting. The columns list (1) the average time to complete a system, (2) the time required for encompassment, and (3) the time for variant retrieval. Retrieving unifiable terms for the computation of overlaps requires only about 1% of the computation time (using naive search in node sets). While path indexing hardly adds anything, the use of discrimination trees allows to decrease this ratio to 0.3%.

Critical Pair Criteria. Table 3 summarizes the outcome of $\mathsf{mkb}_{\mathrm{TT}}$ applying the critical pair criteria mentioned in Section 4. For the column "all", the criteria PCP, BCP and CCP were combined such that every equation deemed superfluous by some criterion got filtered out. The columns list (1) the time required to complete a system, and (2) the number of critical pairs that were recognized as redundant, both for the successful process and for all processes. In the last two rows, the number of successful completions and the total time for all problems in the test series are given. The prefix BGK94 designates examples originating from [Bün94]. Problem Chr89-A_2 is taken from [Chr89], CGE_3 stems from [Stu06], WS06-1 describes the proof reduction system presented in [Weh05], and GRP463-1 is the theory associated with the respective problem in the unit equality division of TPTP [Sut09].

The use of PCP, BCP and the combination of all criteria increments the number of successes by one. Timewise, there are some input problems which exhibit a speedup with critical pair criteria, such as BGK-D_{16} or GRP463-1. For other problems such as Chr89-A_2, many critical pairs were filtered out but few of them were actually of use for the successful process. Comparing the implemented criteria, PCP proved to be both the fastest and the most powerful option. BCP recognizes slightly fewer redundancies, but is still feasible.

	none	PCP		BCP		CCP		all	
	(1)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
BGK94-D_8	∞	550.9	28/220	550.2	28/212	∞		549.7	28/224
BGK94-D_{16}	105.9	101.1	19/119	104.3	19/112	106.6	8/41	104.7	20/125
Chr89-A_2	126.8	133.7	70/904	134.5	51/771	168.7	25/199	137.5	75/923
CGE_3	198.7	197.7	16/23	198.3	16/20	197.5	8/11	200.4	18/29
GRP463-1	8.6	5.5	24/237	7.1	24/223	9.5	9/43	6.5	27/257
WS06-1	138.6	139.7	0/0	140.3	0/0	139.3	0/0	138.3	0/0
⋮	⋮	⋮		⋮		⋮		⋮	
successes	70	71		71		70		71	
total time	18867	18814		18815		18935		18822	

Table 3: Critical pair criteria.

	$s_{\texttt{sum}}$		$s_{\texttt{max}}$		$s_{\texttt{mkbtt1}}$		$s_{\texttt{mkbtt1max}}$		$s_{\texttt{slothrop}}$	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
Chr89-A_2	77.9	153	149.6	150	∞		∞		∞	
SK90-3.04	74.6	133	1.6	39	37.6	105	2.3	42	2.9	33
SK90-3.27	59.1	68	70.0	46	56.8	58	178.5	86	∞	
BGK94-D_8	303.7	217	90.4	134	∞		71.1	105	591.9	160
BGK94-D_{10}	39.8	126	31.7	102	∞		198.4	171	∞	
BGK94-M_{14}	1.48	34	∞		∞		∞		∞	
TPTP/GRP454-1	87.4	168	2.0	38	14.5	75	∞		8.8	40
TPTP/GRP484-1	252.2	324	∞		∞		∞		∞	
CGE_2	138.2	157	9.0	44	7.6	56	9.9	46	15.8	46
CGE_3	∞		189.6	56	∞		121.3	58	343.9	66
⋮	⋮		⋮		⋮		⋮		⋮	
successes	74		71		66		68		69	
average time	22.2		12.8		23.5		15.8		38.9	

Table 4: Selection strategies.

For CCP, the computational overhead clearly outweighs the advantage of the (rather few) superfluous critical pairs. Also using the combination of all criteria is hardly worth the effort, despite the fact that the largest number of critical pairs is filtered out. Overall, critical pair criteria allow for rather small improvements.

Selection Strategies. Table 4 illustrates the effect of different selection strategies. The strategies $s_{\texttt{sum}}$, $s_{\texttt{max}}$, $s_{\texttt{slothrop}}$ and $s_{\texttt{mkbtt1}}$ are described in Section 5, where the latter two use a size-age ratio of 0.65. The strategy $s_{\texttt{mkbtt1max}}$ differs from $s_{\texttt{mkbtt1}}$ in that `ssum` is replaced by `smax`. The columns list (1) the time required to complete a system and (2) the number of selected nodes, i.e., the number of iterations of the main control loop. In the last two rows the number of successful completions and the average time for these are given. The prefix SK90 designates examples originating from [Ste90].

For many problems, the results depend greatly on the selection strategy used. Some problems like CGE_3 perform better when the node size measure `smax` is used. Using `smax`

also improves the average time to complete a problem, although overall $s_{\texttt{sum}}$ could complete the most input problems. There are also some problems like SK90-3.04 where $s_{\texttt{slothrop}}$ needs the fewest node selections. However, this strategy tends to take more time as the critical pair computation in its node measure is costly to compute. Altogether, 78 examples could be completed with some strategy.

References

[Ala07] B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Proving termination of context-sensitive rewriting with MU-TERM. In *Proc. 6th PROLE*, *ENTCS*, vol. 188, pp. 105–115. 2007.

[Bac88] L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.

[Bac94] L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of the ACM*, 41(2):236–276, 1994.

[Bün94] R. Bündgen, M. Göbel, and W. Küchlin. A fine-grained parallel completion procedure. In *Proc. 7th ISSAC*, pp. 269–277. 1994.

[Chr89] J. Christian. Fast Knuth-Bendix completion. In *Proc. 3rd RTA*, *LNCS*, vol. 355, pp. 551–555. 1989.

[Gie06] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. 3rd IJCAR*, *LNAI*, vol. 4130, pp. 281–286. 2006.

[Gra96] P. Graf. *Term Indexing*, *LNAI*, vol. 1053. Springer-Verlag, 1996.

[Kap88] D. Kapur, D.R. Musser, and P. Narendran. Only prime superpositions need be considered in the Knuth-Bendix completion procedure. *Journal of Symbolic Computation*, 6(1):19–36, 1988.

[Kor09] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean termination tool 2. In *Proc. 20th RTA*, *LNCS*, vol. 5595, pp. 295–304. 2009.

[Küc85] W. Küchlin. A confluence criterion based on the generalised Newman lemma. In *Proc. 2nd EUROCAL*, *LNCS*, vol. 204, pp. 390–399. 1985.

[Kur99] M. Kurihara and H. Kondo. Completion for multiple reduction orderings. *Journal of Automated Reasoning*, 23(1):25–42, 1999.

[Löc02] B. Löchner and T. Hillenbrand. A phytography of WALDMEISTER. *AI Communications*, 15(2-3):127–133, 2002.

[McC92] W. McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.

[Sat08] H. Sato, S. Winkler, M. Kurihara, and A. Middeldorp. Multi-completion with termination tools (system description). In *Proc. 4th IJCAR*, *LNAI*, vol. 5195, pp. 306–312. 2008.

[Sat09] H. Sato, S. Winkler, M. Kurihara, and A. Middeldorp. Constraint-based multi-completion procedures for term rewriting systems. *IEICE Transactions on Information and Systems*, E92-D(2):220–234, 2009.

[Sek01] R. Sekar, I. V. Ramakrishnan, and A. Voronkov. Term indexing. In *Handbook of Automated Reasoning*, pp. 1853–1964. Elsevier Science Publishers, 2001.

[Ste90] J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Tech. Rep. SR-90-25, Universität Kaiserslautern, 1990.

[Stu06] A. Stump and B. Löchner. Knuth-Bendix completion of theories of commuting group endomorphisms. *Information Processing Letters*, 98(5):195–198, 2006.

[Sut09] G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[Vor95] A. Voronkov. The anatomy of Vampire. *Journal of Automated Reasoning*, 15(2):237–265, 1995.

[Vor01] A. Voronkov. Algorithms, datastructures, and other issues in efficient automated deduction. In *Proc. 1st IJCAR*, *LNCS*, vol. 2083, pp. 13–28. 2001.

[Weh05] I. Wehrman and A. Stump. Mining propositional simplification proofs for small validating clauses. In *Proc. 3rd PDPAR*, *ENTCS*, vol. 144, pp. 79–91. 2005.

[Weh06] I. Wehrman, A. Stump, and E.M. Westbrook. Slothrop: Knuth-Bendix completion with a modern termination checker. In *Proc. 17th RTA*, *LNCS*, vol. 4098, pp. 287–296. 2006.