J Autom Reasoning manuscript No.
(will be inserted by the editor)

Multi-Completion with Termination Tools

Sarah Winkler · Haruhiko Sato · Aart Middeldorp · Masahito Kurihara

Received: date / Accepted: date

Abstract Knuth-Bendix completion is a classical calculus in automated deduction for transforming a set of equations into a confluent and terminating set of directed equations which can be used to decide the induced equational theory. Multi-completion with termination tools constitutes an approach that differs from the classical method in two respects: (1) external termination tools replace the reduction order—a typically critical parameter—as proposed by Wehrman, Stump and Westbrook (2006), and (2) multi-completion as introduced by Kurihara and Kondo (1999) is used to keep track of multiple orientations in parallel while exploiting sharing to boost efficiency. In this paper we describe the inference system, give the full proof of its correctness and comment on completeness issues. Critical pair criteria and isomorphisms are presented as refinements together with all proofs. We furthermore describe the implementation of our approach in the tool mkbTT, present extensive experimental results and report on new completions.

1 Introduction

In a landmark paper, Knuth and Bendix [**?**] introduced a completion procedure which aims to transform a set of input equalities into a terminating and confluent rewrite system. If successful, the resulting system allows to decide the associated equational theory. However, success of a completion run critically depends on the reduction order that is required as additional input.

The first author is supported by a DOC-fFORTE grant of the Austrian Academy of Sciences.

S. Winkler · A. Middeldorp
Institute of Computer Science, University of Innsbruck,
Innsbruck, Austria
E-mail: sarah.winkler@uibk.ac.at, aart.middeldorp@uibk.ac.at

H. Sato · M. Kurihara
Graduate School of Information Science and Technology
Hokkaido University, Japan
E-mail: haru@complex.eng.hokudai.ac.jp, kurihara@ist.hokudai.ac.jp

Kurihara and Kondo [?] introduced the calculus of *multi-completion* which supports completion with multiple reduction orders at the same time. Basically, a deduction simulates parallel completion runs with the orders under consideration but gains efficiency by sharing inference steps among the parallel deductions. Since multi-completion succeeds as soon as one of the mimicked runs achieves a result, this approach partially tackles the problem of choosing an appropriate reduction order. Still, concrete reduction orders have to be provided as input.

Wehrman, Stump and Westbrook [?] proposed a different approach. Instead of relying on a reduction order supplied by the user, rewrite rules are oriented by a termination prover internally. In this way an appropriate reduction order is implicitly developed along the deduction. Since modern termination provers employ many more sophisticated techniques than plain reduction orders, this approach allows to construct convergent systems that cannot be obtained with classical completion procedures. One such system is the theory of two commuting group endomorphisms CGE_2 [?], which can be completed by the tool Slothrop described in [?] without user interaction.

Multi-completion with termination tools [?,?,?] constitutes a combination of these two approaches. While the use of termination tools allows for automatic completion without user interaction, multi-completion enables to keep track of multiple combinations of orientations in parallel, thereby exploiting sharing for efficiency reasons. The implementation of our technique thus yields a powerful tool for automatic completion with a high flexibility concerning orientations. In this paper we describe the underlying inference system, and present simulation and correctness results along with the proofs that were omitted in [?,?] due to reasons of space. After detecting a flaw in the fairness definition of [?], we newly contribute a corrected and explicit definition for MKBtt. We also present a novel completeness result for a variant of the MKBtt calculus that (to our knowledge) has not been achieved by previous completion approaches. Refinements such as critical pair criteria and isomorphisms that were already outlined in [?] are described in more detail along with proofs showing their correctness. In a section on implementation details, besides the basic control loop we present optimizations such as term indexing techniques and selection strategies and explain how various options can be controlled by the user. Recent optimizations led to the completion of novel systems such as CGE_5. By conducting thorough benchmark tests on a considerably extended database we assessed the different enhancements. Extending the experiments presented in previous papers, we also compare with *maximal completion* as developed by Klein and Hirokawa [?]. In short, this article constitutes a comprehensive report of the MKBtt approach and subsumes earlier contributions.

The paper is structured as follows. We start by summarizing preliminaries in Section 2. In Section 3 we present the inference system underlying MKBtt, simulation and correctness results as well as a completeness result concerning a modified version of the calculus. As optimizations, critical pair criteria and isomorphisms are presented in Sections 4 and 5. Section 6 comments on some implementation details of our completion tool before experimental results are described in Section 7. Finally, Section 8 adds some concluding remarks and lists issues for future work.

2 Preliminaries

We consider terms $\mathcal{T}(\mathcal{F},\mathcal{V})$ over a finite signature $\mathcal{F}$ and a set of variables $\mathcal{V}$. For some term t we denote by $\mathcal{P}\mathrm{os}(t)$ its set of *positions*, which is partitioned into function

symbol positions $\mathcal{P}\mathrm{os}_{\mathcal{F}}(t)$ and variable positions $\mathcal{P}\mathrm{os}_{\mathcal{V}}(t)$. If $p \in \mathcal{P}\mathrm{os}(t)$ then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ is the term obtained from t when replacing $t|_p$ by s. A term t *encompasses* a term s, denoted by $t \trianglerighteq s$, if $t = t[s\sigma]_p$ holds for some substitution σ and position $p \in \mathcal{P}\mathrm{os}(t)$. The strict part $\trianglerighteq \setminus \trianglelefteq$ of this relation is denoted by $\triangleright$. We call two terms s and t *variants* and write $s \doteq t$ if there exists a variable renaming σ such that $s\sigma = t\sigma$.

Sets of equations between terms will be denoted by $\mathcal{E}$ and are assumed to be symmetric. The associated *equational theory* is denoted by $\approx_{\mathcal{E}}$. As usual a set of directed equations $\ell \to r$ is called a rewrite system and denoted by $\mathcal{R}$, where $\to_{\mathcal{R}}$ is the associated *rewrite relation*. We write $s \xrightarrow{\ell \to r}_p t$ to express that $s \to_{\mathcal{R}} t$ was achieved by applying the rule $\ell \to r \in \mathcal{R}$ at position p. The relations $\to_{\mathcal{R}}^+$, $\to_{\mathcal{R}}^*$ and $\leftrightarrow_{\mathcal{R}}$ denote the transitive, transitive-reflexive and symmetric closure of $\to_{\mathcal{R}}$. The smallest equivalence relation containing $\to_{\mathcal{R}}$, which coincides with the equational theory $\approx_{\mathcal{R}}$ if $\mathcal{R}$ is considered as a set of equations, is denoted by $\leftrightarrow_{\mathcal{R}}^*$. Subscripts are omitted if the rewrite system or the set of equations is clear from the context.

A rewrite system $\mathcal{R}$ is *terminating* if it does not admit infinite rewrite sequences. It is *confluent* if for every peak $t \,{}^*\!\leftarrow s \to^* u$ there exists a term v such that $t \to^* v \,{}^*\!\leftarrow u$. An *overlap* is a triple $\langle u \to v, p, \ell \to r\rangle$ where $u \to v$ and $\ell \to r$ are rewrite rules without common variables such that $p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(u)$, $u|_p$ and ℓ are unifiable with most general unifier σ, and if $\ell \to r$ and $u \to v$ are variants then $p \neq \epsilon$. The term $u\sigma = u\sigma[\ell\sigma]_p$ can be rewritten in two different ways, resulting in the *critical pair* $v\sigma \approx u\sigma[r\sigma]_p$. The set of critical pairs among rules in $\mathcal{R}$ is denoted by $\mathrm{CP}(\mathcal{R})$. A rewrite system $\mathcal{R}$ which is both terminating and confluent is called *convergent*. If $\mathcal{R}$ has the property that for every rewrite rule $\ell \to r$ the right-hand side r is in normal form and the left-hand side ℓ is in normal form with respect to $\mathcal{R} \setminus \{\ell \to r\}$ then $\mathcal{R}$ is called *reduced*. We call $\mathcal{R}$ *convergent for* a set of equations $\mathcal{E}$ if $\mathcal{R}$ is convergent and $\leftrightarrow_{\mathcal{R}}^*$ coincides with $\approx_{\mathcal{E}}$. A convergent and reduced rewrite system is called *canonical*.

A proper order $\succ$ on terms is a *rewrite order* if it is closed under contexts and substitutions. A well-founded rewrite order is called a *reduction order*. The relation $\to_{\mathcal{R}}^+$ is a reduction order for every terminating rewrite system $\mathcal{R}$.

In the context of completion, we often consider a pair $(\mathcal{E}, \mathcal{R})$ of equations $\mathcal{E}$ and rewrite rules $\mathcal{R}$. An *equational proof step* $s \leftrightarrow_e^p t$ in $(\mathcal{E}, \mathcal{R})$ is an *equality step* if e is an equation $\ell \approx r$ in $\mathcal{E}$ or a *rewrite step* if e is a rule $\ell \to r$ in $\mathcal{R}$, and either $s = u[\ell\sigma]_p$ and $t = u[r\sigma]_p$ or $s = u[r\sigma]_p$ and $t = u[\ell\sigma]_p$ hold for some substitution σ and term u with position p. We sometimes write $s \leftrightarrow t$ to express the existence of some proof step, omitting the position p and equation or rule e. An *equational proof* P of an equation $t_0 \approx t_n$ is a finite sequence

$$t_0 \underset{e_0}{\overset{p_0}{\longleftrightarrow}} t_1 \underset{e_1}{\overset{p_1}{\longleftrightarrow}} \cdots \underset{e_{n-1}}{\overset{p_{n-1}}{\longleftrightarrow}} t_n \tag{1}$$

of equational proof steps. Note that $(\mathcal{E}, \mathcal{R})$ admits an equational proof of $s \approx t$ if and only if $s \leftrightarrow_{\mathcal{E}\cup\mathcal{R}}^* t$ holds. A sequence Q of the form $t_i \leftrightarrow \cdots \leftrightarrow t_j$ with $0 \leqslant i \leqslant j \leqslant n$ is a *subproof* of P. We write $P[Q]$ to express that P contains Q as a subproof. If P is an equational proof and σ a substitution then $P\sigma$ denotes the instantiated proof

$$t_0\sigma \underset{e_0}{\overset{p_0}{\longleftrightarrow}} t_1\sigma \underset{e_1}{\overset{p_1}{\longleftrightarrow}} \cdots \underset{e_{n-1}}{\overset{p_{n-1}}{\longleftrightarrow}} t_n\sigma$$

deduce	$\dfrac{\mathcal{E},\mathcal{R}}{\mathcal{E}\cup\{s\approx t\},\mathcal{R}}$	if $s\approx t\in \mathrm{CP}(\mathcal{R})$
orient	$\dfrac{\mathcal{E}\cup\{s\approx t\},\mathcal{R}}{\mathcal{E},\mathcal{R}\cup\{s\to t\}}$	if $s\succ t$
delete	$\dfrac{\mathcal{E}\cup\{s\approx s\},\mathcal{R}}{\mathcal{E},\mathcal{R}}$	
simplify	$\dfrac{\mathcal{E}\cup\{s\approx t\},\mathcal{R}}{\mathcal{E}\cup\{s\approx u\},\mathcal{R}}$	if $t\to_{\mathcal{R}} u$
compose	$\dfrac{\mathcal{E},\mathcal{R}\cup\{s\to t\}}{\mathcal{E},\mathcal{R}\cup\{s\to u\}}$	if $t\to_{\mathcal{R}} u$
collapse	$\dfrac{\mathcal{E},\mathcal{R}\cup\{t\to s\}}{\mathcal{E}\cup\{u\approx s\},\mathcal{R}}$	if $t\to_{\mathcal{R}} u$ using $\ell\to r$ such that $t\rhd\ell$

Fig. 1 System KB of standard completion.

For a term u with position q and a proof P of the shape (1) we write $u[P]_q$ to denote the sequence

$$u[t_0]_q \overset{qp_0}{\underset{e_0}{\longleftrightarrow}} u[t_1]_q \overset{qp_1}{\underset{e_1}{\longleftrightarrow}} \cdots \overset{qp_{n-1}}{\underset{e_{n-1}}{\longleftrightarrow}} u[t_n]_q$$

which is again an equational proof. Proofs of the shape $t_0 \to \cdots \to t_i \leftarrow \cdots \leftarrow t_n$ are called *rewrite proofs*. They play a special role in the context of completion. A *proof order* $\succcurlyeq$ is a well-founded order on equational proofs such that

i. $P \succcurlyeq Q$ implies $u[P\sigma]_p \succcurlyeq u[Q\sigma]_p$ for all terms u, positions $p \in \mathcal{P}\mathrm{os}(u)$ and substitutions σ,
ii. if P and P' prove the same equation then $P \succcurlyeq P'$ implies $Q[P] \succcurlyeq Q[P']$ for all proofs Q.

A *proof reduction relation* $\Rightarrow$ additionally satisfies

iii. $P \Rightarrow Q$ holds only if P and Q prove the same equation.

2.1 Standard Completion

The classical completion procedure proposed by Knuth and Bendix [**?**] was reformulated as an inference system by Bachmair [**?**], as depicted in Figure 1. The inference system (in the sequel referred to as KB) works on pairs $(\mathcal{E},\mathcal{R})$ consisting of a set of equations $\mathcal{E}$ and a set of rewrite rules $\mathcal{R}$, and is parameterized by a reduction order $\succ$. The inference rules of KB induce a proof transformation relation on the level of equational proofs. For example, if deduce adds a critical pair between rules $\ell\to r$ and $u\to v$ that overlap on a term w, this allows to replace the peak $s \overset{\ell\leftarrow r}{\longleftarrow} w \overset{u\to v}{\longrightarrow} t$ by the proof $s \overset{s\approx t}{\longleftrightarrow} t$. Similarly, the other inference rules allow to replace patterns in equational proofs. In the sequel, we denote by $\Rightarrow_{\mathsf{KB}}^{\succ}$ the (transitive) proof transformation relation induced by KB using reduction order $\succ$. This relation terminates and constitutes a proof reduction relation. [**?**]

A KB inference sequence of the form $(\mathcal{E}_0, \mathcal{R}_0) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash (\mathcal{E}_2, \mathcal{R}_2) \vdash \cdots$ is in the sequel referred to as a *run* with *persistent* equations $\mathcal{E}_\omega = \bigcup_i \bigcap_{j>i} \mathcal{E}_j$ and rules $\mathcal{R}_\omega = \bigcup_i \bigcap_{j>i} \mathcal{R}_j$. A run *fails* if $\mathcal{E}_\omega$ is not empty, it *succeeds* if $\mathcal{E}_\omega$ is empty and $\mathcal{R}_\omega$ is confluent and terminating. Moreover, a run is called *simplifying* if $\mathcal{R}_\omega$ is reduced. Since every inference step is reflected by one or more steps in the proof reduction relation $\Rightarrow^{\succ}_{\mathsf{KB}}$ and this relation terminates, in non-failing runs every identity is eventually connected by a rewrite proof, provided that required inference steps are not indefinitely ignored. This property is captured by the notion of *fairness*.

Definition 1 A run $(\mathcal{E}_0, \mathcal{R}_0) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash (\mathcal{E}_2, \mathcal{R}_2) \vdash \cdots$ is *fair* with respect to a proof reduction relation $\Rightarrow$ if for every non-rewrite proof P in $(\mathcal{E}_\omega, \mathcal{R}_\omega)$, for which there exists an inference step $(\mathcal{E}_\omega, \mathcal{R}_\omega) \vdash (\mathcal{E}'_\omega, \mathcal{R}'_\omega)$ and a proof P' in $(\mathcal{E}'_\omega, \mathcal{R}'_\omega)$ satisfying $P \Rightarrow P'$, there also exists a proof Q in $(\mathcal{E}_i, \mathcal{R}_i)$ for some $i \geqslant 0$ such that $P \Rightarrow Q$ holds.

A simpler and sufficient condition states that any run satisfying $\mathrm{CP}(\mathcal{R}_\omega) \subseteq \bigcup_i \mathcal{E}_i$ is fair. Finally, we recall the main theorems stating correctness and completeness of the inference system KB [**?**].

Theorem 1 *Any non-failing KB run using a reduction order $\succ$ that is fair with respect to $\Rightarrow^{\succ}_{KB}$ succeeds.* □

Theorem 2 *Assume there exists a finite convergent system $\mathcal{R}$ which has the same equational theory as a set of equations $\mathcal{E}$ and is contained in $\succ$. Then any non-failing run from $\mathcal{E}$ using $\succ$ which is fair with respect to $\Rightarrow^{\succ}_{KB}$ will produce a convergent system in finitely many steps.* □

With a suitable reduction order a run is thus guaranteed to produce a convergent system, provided that no persistent unorientable equations are encountered. However, a different reduction order might induce an infinite run, or even lead to failure. The choice of the order is thus highly critical for success, but hard to determine in advance. Different approaches have been proposed to tackle this problem. In the following sections we outline two of them. *Multi-completion* increases the chance for success by keeping track of multiple runs using different orders whereas *completion with termination tools* attempts to develop a suitable order in the course of the deduction by using modern termination provers, thereby considerably widening the class of applicable orders.

2.2 Multi-Completion

Completion with multiple reduction orders—referred to as multi-completion in the sequel—was proposed by Kondo and Kurihara [**?**]. For a set $\mathcal{O} = \{\succ_1, \ldots, \succ_n\}$ of orders, it simulates the parallel execution of corresponding completion runs, but shares common inference steps to gain efficiency. The key idea to sharing is a data structure called *node*.

Definition 2 A *node* is a tuple $\langle s : t, R_0, R_1, E \rangle$ where the *data* $s : t$ consist of terms s, t and the *labels* R_0, R_1, E are subsets of $\mathcal{O}$. The *node condition* requires that R_0, R_1 and E are mutually disjoint, $s \succ_i t$ holds for all $\succ_i \in R_0$, and $t \succ_i s$ for all $\succ_i \in R_1$.

orient	$\dfrac{N \cup \{\langle s : t, R_0, R_1, E \uplus R\rangle\}}{N \cup \{\langle s : t, R_0 \cup R, R_1, E\rangle\}}$	if $R \neq \varnothing$ and $s \succ_i t$ for all $\succ_i \in R$

Fig. 2 The orient rule in MKB.

orient	$\dfrac{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}, \mathcal{C}}{\mathcal{E}, \mathcal{R} \cup \{s \to t\}, \mathcal{C} \cup \{s \to t\}}$	if $\mathcal{C} \cup \{s \to t\}$ terminates

Fig. 3 The orient rule in KBtt.

Intuitively, a node $\langle s : t, R_0, R_1, E\rangle$ captures the state of the term pair $s : t$ in all simulated completion processes. All orders in the *equation label* E regard the data as an equation $s \approx t$ while orders in the *rewrite labels* R_0 and R_1 consider it as rewrite rules $s \to t$ and $t \to s$, respectively. Hence the node $\langle s : t, R_0, R_1, E\rangle$ is identified with $\langle t : s, R_1, R_0, E\rangle$.

Multi-completion can be described by an inference system MKB which operates on sets of nodes and consists of five rules. Figure 2 shows the orient inference rule. An MKB run γ of the form $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ can be projected to a valid KB run γ_i for every order $\succ_i \in \mathcal{O}$, and conversely every KB run using $\succ_i$ can be modelled by an MKB run. Due to these simulation properties also correctness and completeness results are obtained for MKB. For this purpose, a run γ is called *fair* if it is either finite and γ_i is a fair and nonfailing[1] KB run for some i, or if it is infinite and all γ_i are either fair or failing.

2.3 Completion with Termination Tools

Standard completion procedures depend critically on the choice of the reduction order supplied as input, thus requiring a careful decision by the user. The evolution of powerful modern termination provers exploiting a variety of sophisticated methods thus suggests to guarantee termination by employing respective tools instead of a fixed order. Such an approach was proposed by Wehrman, Stump and Westbrook [**?**] and implemented in the tool Slothrop. Some care has to be taken because it is known [**?**] that changing the reduction order during a completion run may result in a non-confluent rewrite system. The inference system KBtt underlying Slothrop thus operates on triples $(\mathcal{E}, \mathcal{R}, \mathcal{C})$ consisting of a set of equations $\mathcal{E}$, a rewrite system $\mathcal{R}$ and an additional rewrite system $\mathcal{C}$. This extra *constraint system* ensures that orientations are never reversed throughout a run, thereby guaranteeing confluence of the derived system.

The system KBtt consists of the orient inference rule depicted in Figure 3 together with the remaining KB rules where the constraint component is not modified. Again, $\vdash$ denotes the inference relation and $\vdash^=$ its reflexive closure. An empty step $(\mathcal{E}, \mathcal{R}, \mathcal{C}) \vdash^= (\mathcal{E}, \mathcal{R}, \mathcal{C})$ is also called an *equality step*. Since constraint rules are only added if termination is preserved, all constraint systems $\mathcal{C}_0 \subseteq \mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \cdots$ developed during a deduction terminate. Thus the relations $\to^+_{\mathcal{C}_i}$ constitute a sequence of

[1] Note that our definition differs from the original definition in [**?**] in that we require γ_i to be nonfailing; otherwise, a finite nonfailing and fair MKB run need not generate a convergent system if it is only fair for γ_i.

subsequently refined reduction orders with respect to which completion is performed, naturally exploiting the incrementality of reduction orders defined by a rewrite relation. In this respect the method resembles the approach adopted by the tool REVE [**?**] where the advantages of an incremental order are emphasized. Any KB run using $\succ$ can obviously be simulated in KBtt since the required termination checks of the constraint systems succeed when employing $\succ$. Conversely, finite KBtt runs deriving the final constraint system $\mathcal{C}$ are reflected by KB runs that use the reduction order $\rightarrow_{\mathcal{C}}^{+}$. Hence a KBtt run is called *fair*, *successful*, *failing* and *simplifying* whenever the respective definition applies to the simulated KB run. This entails finite correctness of KBtt [**?**], although this result does not extend to infinite runs as the infinite union of terminating rewrite systems need not terminate.

Theorem 3 *Any finite non-failing and fair KBtt run succeeds.* □

3 Multi-Completion with Termination Tools

In an orient step of KBtt, termination of a new rule $s \to t$ together with the set $\mathcal{C}$ of all previously oriented rules is checked. If both orientations $s \to t$ and $t \to s$ terminate together with $\mathcal{C}$, an implementation encounters the challenge how to deal with this choice. Slothrop uses a best-first strategy to decide which branch to explore further. In contrast, MKBtt keeps track of both orientations but avoids an explosion of the search space by integrating the concept of multi-completion to share common inferences.

Since every simulated KBtt branch corresponds to a sequence of decisions on how to orient nodes, a *process* p is modeled by a bit string in $(0+1)^*$. A set of processes P is called *well-encoded* if there are no pairs of processes p and p' in P such that p' is a proper prefix of p. The initial process is represented by the empty string ϵ.

MKBtt is described by an inference system operating on a set of nodes. In contrast to MKB, labels are now sets of processes instead of reduction orders, and in order to account for the constraint systems required in KBtt, nodes are extended with two additional constraint labels.

Definition 3 An MKBtt node $\langle s : t, R_0, R_1, E, C_0, C_1\rangle$ contains as *data* two terms s and t and as *labels* sets of processes R_0, R_1, E, C_0, C_1, where the *node condition* requires that $R_0 \cup C_0$, $R_1 \cup C_1$ and E are mutually disjoint.

The process sets R_0, R_1 are called *rewrite labels*, $\mathcal{E}$ is the *equation label* and C_0, C_1 are the *constraint labels*. As in the case of MKB, the node $\langle s : t, R_0, R_1, E, C_0, C_1\rangle$ is identified with $\langle t : s, R_1, R_0, E, C_1, C_0\rangle$. The sets of all processes occurring in a node n or a node set N are denoted by $\mathcal{P}(n)$ and $\mathcal{P}(N)$, respectively. To relate a node set N to the corresponding states of the simulated KBtt processes, *projections* are used.

Definition 4 For a node $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ and a process p, the equation and rule projection of n to p are defined as

$$E[n,p] = \begin{cases} \{s \approx t\} & \text{if } p \in E \\ \varnothing & \text{otherwise} \end{cases} \qquad R[n,p] = \begin{cases} \{s \to t\} & \text{if } p \in R_0 \\ \{t \to s\} & \text{if } p \in R_1 \\ \varnothing & \text{otherwise} \end{cases}$$

The constraint projection $C[n,p]$ is defined analogous to $R[n,p]$. These projections are naturally extended to node sets by defining $E[N,p] = \bigcup_{n\in N} E[n,p]$, $R[N,p] = \bigcup_{n\in N} R[n,p]$ and $C[N,p] = \bigcup_{n\in N} C[n,p]$.

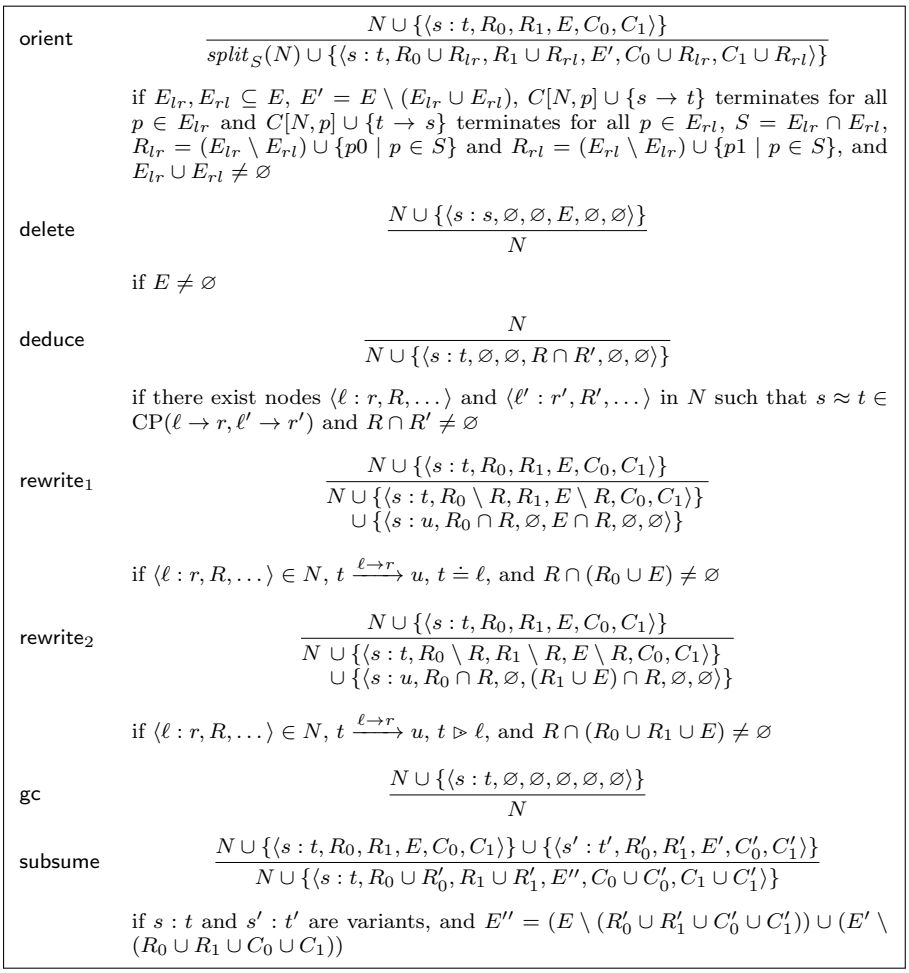

Fig. 4 Inference rules of MKBtt.

The inference rules of MKBtt are depicted in Figure 4. Note that all rules preserve well-encodedness of labels and the disjointness condition on nodes. The following paragraphs add some clarifying remarks on the inference rules.

– The orient rule applied to a node $\langle s : t, R_0, R_1, E, C_0, C_1\rangle$ attempts to turn the equation $s \approx t$ into a rule for as many processes as possible. This is modelled in the node structure by moving processes $p \in E$ to rewrite labels. More precisely, the respective inference rule in KBtt is modelled by checking for every process $p \in E$ whether its current constraint system $C[N,p]$ terminates when extended with $s \to t$ or $t \to s$. If $C[N,p] \cup \{s \to t\}$ terminates then p is added to the set E_{lr}, and if $C[N,p] \cup \{t \to s\}$ terminates then p is added to the set E_{rl}. The set $E_{lr} \setminus E_{rl}$ ($E_{rl} \setminus E_{lr}$) thus collects processes which can only perform the orientation $s \to t$ ($t \to s$). These processes are added to R_0 and C_0 (R_1 and C_1). The set $S = E_{lr} \cap E_{rl}$ collects processes that allow both orientations. Thus every $p \in S$ is

split into two child processes $p0$ and $p1$, and pi is added to R_i and C_i, for $i \in \{0, 1\}$. Finally, $split_S(N)$ replaces every occurrence of a process in S by its descendants: the operation $split_S(P) = (P \setminus S) \cup \{p0, p1 \mid p \in P \cap S\}$ is applied to every process set P occurring in N.
- If the current node set N contains nodes with data $\ell : r$ and $\ell' : r'$ such that the rules $\ell \to r$ and $\ell' \to r'$ give rise to a critical pair $s \approx t$, deduce adds a respective node for all processes p that have both rules present in their current rewrite system $R[N, p]$.
- In standard completion, given a term pair $s : t$ and a rewrite step $t \xrightarrow{\ell \to r} u$, the rules compose, simplify and collapse create a term pair $s : u$. If t and ℓ are variants, the MKBtt rule $\mathsf{rewrite}_1$ allows to combine respective compose and simplify steps. If $t \rhd \ell$ holds, $\mathsf{rewrite}_2$ simulates all three rules at once.
- To increase efficiency, the optional gc rule deletes nodes with empty labels.
- The rule subsume is optional as well, it merges pairs of nodes which have the same data up to renaming.

As usual, a sequence of MKBtt inference steps $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ is referred to as a *run*. Given a set of equations $\mathcal{E}$, the *initial node set* $N_0 = N_{\mathcal{E}}$ consists of all nodes $\langle s : t, \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle$ such that $s \approx t$ is in $\mathcal{E}$.

3.1 An Example

In this section we illustrate multi-completion with termination tools on the example system CGE_2, which consists of the following equations:

$$\begin{aligned} \mathsf{e} \cdot x &\approx x & \mathsf{f}(x \cdot y) &\approx \mathsf{f}(x) \cdot \mathsf{f}(y) \\ \mathsf{i}(x) \cdot x &\approx \mathsf{e} & \mathsf{g}(x \cdot y) &\approx \mathsf{g}(x) \cdot \mathsf{g}(y) \\ x \cdot (y \cdot z) &\approx (x \cdot y) \cdot z & \mathsf{f}(x) \cdot \mathsf{g}(y) &\approx \mathsf{g}(y) \cdot \mathsf{f}(x) \end{aligned}$$

An MKBtt run starts with the initial node set

$$\langle \mathsf{e} \cdot x : x, \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle \tag{1}$$
$$\langle \mathsf{i}(x) \cdot x : \mathsf{e}, \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle \tag{2}$$
$$\langle x \cdot (y \cdot z) : (x \cdot y) \cdot z, \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle \tag{3}$$
$$\langle \mathsf{f}(x \cdot y) : \mathsf{f}(x) \cdot \mathsf{f}(y), \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle \tag{4}$$
$$\langle \mathsf{g}(x \cdot y) : \mathsf{g}(x) \cdot \mathsf{g}(y), \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle \tag{5}$$
$$\langle \mathsf{f}(x) \cdot \mathsf{g}(y) : \mathsf{g}(y) \cdot \mathsf{f}(x), \varnothing, \varnothing, \{\epsilon\}, \varnothing, \varnothing\rangle \tag{6}$$

When applying orient to nodes (1) and (2), only the direction from left to right yields valid and terminating rewrite rules. For node (3), both orientations are possible such that process ϵ is split into 0 and 1. These three nodes are thus modified as follows:

$$\langle \mathsf{e} \cdot x : x, \{0, 1\}, \varnothing, \varnothing, \{0, 1\}, \varnothing\rangle \tag{1}$$
$$\langle \mathsf{i}(x) \cdot x : \mathsf{e}, \{0, 1\}, \varnothing, \varnothing, \{0, 1\}, \varnothing\rangle \tag{2}$$
$$\langle x \cdot (y \cdot z) : (x \cdot y) \cdot z, \{0\}, \{1\}, \varnothing, \{0\}, \{1\}\rangle \tag{3}$$

Nodes (4) and (5) can be oriented in both directions, independent of the orientation of associativity. Now the current node set contains eight processes (constraint labels

are omitted for the sake of readability; at this point they coincide with the respective rewrite labels):

$$\langle \mathsf{e} \cdot x : x, \{000, \dots, 111\}, \varnothing, \varnothing, \dots \rangle \tag{1}$$
$$\langle \mathsf{i}(x) \cdot x : \mathsf{e}, \{000, \dots, 111\}, \varnothing, \varnothing, \dots \rangle \tag{2}$$
$$\langle x \cdot (y \cdot z) : (x \cdot y) \cdot z, \{000, 001, 010, 011\}, \{100, 101, 110, 111\}, \varnothing, \dots \rangle \tag{3}$$
$$\langle \mathsf{f}(x \cdot y) : \mathsf{f}(x) \cdot \mathsf{f}(y), \{000, 001, 100, 101\}, \{010, 011, 110, 111\}, \varnothing, \dots \rangle \tag{4}$$
$$\langle \mathsf{g}(x \cdot y) : \mathsf{g}(x) \cdot \mathsf{g}(y), \{000, 010, 100, 110\}, \{001, 011, 101, 111\}, \varnothing, \dots \rangle \tag{5}$$
$$\langle \mathsf{f}(x) \cdot \mathsf{g}(y) : \mathsf{g}(y) \cdot \mathsf{f}(x), \varnothing, \varnothing, \{000, \dots, 111\}, \dots \rangle \tag{6}$$

We abbreviate $\{000, 001, 010, 011\}$ to P_0 and $\{100, 101, 110, 111\}$ to P_1. The overlap $\mathsf{i}(x) \cdot (x \cdot y) \leftarrow (\mathsf{i}(x) \cdot x) \cdot y \rightarrow \mathsf{e} \cdot y$ between nodes (3) and (2) allows to deduce the additional node

$$\langle \mathsf{i}(x) \cdot (x \cdot y) : \mathsf{e} \cdot y, \varnothing, \varnothing, P_1, \varnothing, \varnothing \rangle \tag{7}$$

A $\mathsf{rewrite}_1$ step with node (1) simplifies this node to

$$\langle \mathsf{i}(x) \cdot (x \cdot y) : \mathsf{e} \cdot y, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle \tag{7}$$

and adds

$$\langle \mathsf{i}(x) \cdot (x \cdot y) : y, \varnothing, \varnothing, P_1, \varnothing, \varnothing \rangle \tag{8}$$

The former is removed by gc and the latter is oriented to

$$\langle \mathsf{i}(x) \cdot (x \cdot y) : y, P_1, \varnothing, \varnothing, P_1, \varnothing \rangle \tag{8}$$

In a similar way, for processes in P_0 the overlap $(x \cdot \mathsf{i}(y)) \cdot y \leftarrow x \cdot (\mathsf{i}(y) \cdot y) \rightarrow x \cdot \mathsf{e}$ between (3) and (2) yields a node

$$\langle (x \cdot \mathsf{i}(y)) \cdot y : x \cdot \mathsf{e}, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{9}$$

Additionally, there are critical peaks $(x \cdot \mathsf{e}) \cdot y \leftarrow x \cdot (\mathsf{e} \cdot y) \rightarrow x \cdot y$ between nodes (3) and (1), $\mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e} \leftarrow (\mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{i}(x)) \cdot x \rightarrow \mathsf{e} \cdot x$ between nodes (9) and (2), and $x \leftarrow \mathsf{i}(\mathsf{i}(x)) \cdot (\mathsf{i}(x) \cdot x) \rightarrow \mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e}$ between nodes (8) and (2). Orienting the ensuing nodes yields

$$\langle (x \cdot \mathsf{e}) \cdot y : x \cdot y, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{10}$$
$$\langle \mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e} : \mathsf{e} \cdot x, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{11}$$
$$\langle \mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e} : x, P_1, \varnothing, \varnothing, P_1, \varnothing \rangle \tag{12}$$

Applying $\mathsf{rewrite}_1$ with (1) to node (11) creates a node with the same data as (12) also for processes in P_0, such that a $\mathsf{subsume}$ step results in the updated node

$$\langle \mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e} : x, P_1 \cup P_0, \varnothing, \varnothing, P_1, \varnothing \rangle \tag{12}$$

Now the peak $\mathsf{i}(\mathsf{i}(x)) \cdot y \leftarrow (\mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e}) \cdot y \rightarrow x \cdot y$ between (10) and (12) adds

$$\langle \mathsf{i}(\mathsf{i}(x)) \cdot y : x \cdot y, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{13}$$

after a subsequent orient step. At this point overlaps between (13) and (12) and (13) and (2) trigger the creation of nodes that are oriented as

$$\langle x \cdot \mathsf{e} : x, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{14}$$
$$\langle x \cdot \mathsf{i}(x) : \mathsf{e}, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{15}$$

We obtain the modified node

$$\langle (x \cdot \mathsf{i}(y)) \cdot y : x \cdot \mathsf{e}, \varnothing, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{9}$$

when using node (14) in a $\mathsf{rewrite}_1$ step, together with a new node with data $(x \cdot \mathsf{i}(y)) \cdot y : x$, which is oriented as

$$\langle (x \cdot \mathsf{i}(y)) \cdot y : x, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{16}$$

Node (14) can also be used in $\mathsf{rewrite}_2$ steps to modify (10) and (12) to

$$\langle (x \cdot \mathsf{e}) \cdot y : x \cdot y, \varnothing, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{10}$$

and

$$\langle \mathsf{i}(\mathsf{i}(x)) \cdot \mathsf{e} : x, P_1, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{12}$$

while adding

$$\langle x \cdot y : x \cdot y, \varnothing, \varnothing, P_0, \varnothing, \varnothing \rangle \tag{17}$$

and

$$\langle \mathsf{i}(\mathsf{i}(x)) : x, \varnothing, \varnothing, P_0, \varnothing, \varnothing \rangle \tag{18}$$

to the current node set. The latter is oriented into

$$\langle \mathsf{i}(\mathsf{i}(x)) : x, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{18}$$

while node (17) is subject to a delete inference. The overlaps $\mathsf{i}(\mathsf{e}) \leftarrow \mathsf{i}(\mathsf{e}) \cdot \mathsf{e} \rightarrow \mathsf{e}$ between (14) and (2) and $(x \cdot y) \cdot \mathsf{i}(y) \leftarrow x \cdot (y \cdot \mathsf{i}(y)) \rightarrow x \cdot \mathsf{e}$ between (3) and (15) add

$$\langle \mathsf{i}(\mathsf{e}) : \mathsf{e}, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{19}$$

and

$$\langle (x \cdot y) \cdot \mathsf{i}(y) : x, P_0, \varnothing, \varnothing, P_0, \varnothing \rangle \tag{20}$$

to the node set (in the latter case, after $\mathsf{rewrite}_1$ using (14) simplifies $x \cdot \mathsf{e}$ to x).

To make a long story short, we will only sketch the remainder of the run. After some additional deduce steps, the last node concerning plain group theory

$$\langle \mathsf{i}(x \cdot y) : \mathsf{i}(y) \cdot \mathsf{i}(x), \varnothing, \varnothing, P_1 \cup P_0, \varnothing, \varnothing \rangle$$

is derived, and can again be oriented in both directions, resulting in a split of all current processes. To complete the theory of homomorphisms, nodes with data $\mathsf{f}(x) \cdot (\mathsf{f}(y) \cdot z) : \mathsf{f}(x \cdot y) \cdot z$, $\mathsf{f}(\mathsf{e}) : \mathsf{e}$, and $\mathsf{f}(\mathsf{i}(x)) : \mathsf{i}(\mathsf{f}(x))$ and similar ones for g are derived. The last kind of nodes gives again rise to process splits. It remains to orient node (6) and consider

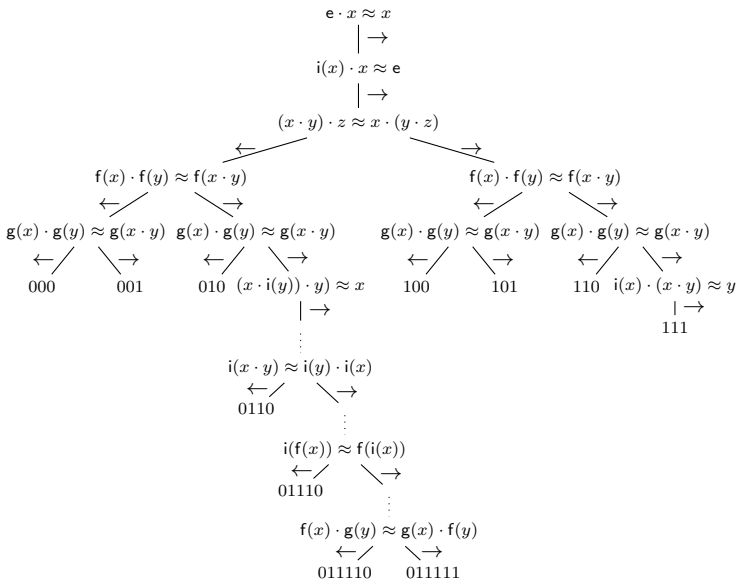

Fig. 5 Part of a CGE_2 process tree with all branching points leading to process 011110.

the critical pair $\mathsf{f}(x) \cdot (\mathsf{g}(y) \cdot z) : \mathsf{g}(y) \cdot (\mathsf{f}(x) \cdot z)$ before e.g. process 011110 succeeds with a convergent system after joining all remaining critical pairs:

$$
\begin{array}{rcc}
\mathsf{e} \cdot x \to x & \mathsf{f}(x) \cdot \mathsf{f}(y) \to \mathsf{f}(x \cdot y) & x \cdot (y \cdot z) \to (x \cdot y) \cdot z \\
x \cdot \mathsf{e} \to x & \mathsf{f}(\mathsf{e}) \to \mathsf{e} & (x \cdot y) \cdot \mathsf{i}(y) \to x \\
\mathsf{i}(x) \cdot x \to \mathsf{e} & \mathsf{i}(\mathsf{f}(x)) \to \mathsf{f}(\mathsf{i}(x)) & (x \cdot \mathsf{i}(y)) \cdot y \to x \\
x \cdot \mathsf{i}(x) \to \mathsf{e} & \mathsf{g}(x) \cdot \mathsf{g}(y) \to \mathsf{g}(x \cdot y) & \mathsf{f}(x) \cdot (\mathsf{f}(y) \cdot z) \to \mathsf{f}(x \cdot y) \cdot z \\
\mathsf{i}(\mathsf{e}) \to \mathsf{e} & \mathsf{g}(\mathsf{e}) \to \mathsf{e} & \mathsf{g}(x) \cdot (\mathsf{g}(y) \cdot z) \to \mathsf{g}(x \cdot y) \cdot z \\
\mathsf{i}(\mathsf{i}(x)) \to x & \mathsf{i}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{i}(x)) & \mathsf{g}(x) \cdot (\mathsf{f}(y) \cdot z) \to \mathsf{f}(x) \cdot (\mathsf{g}(y) \cdot z) \\
\mathsf{i}(x \cdot y) \to \mathsf{i}(y) \cdot \mathsf{i}(x) & \mathsf{g}(x) \cdot \mathsf{f}(y) \to \mathsf{f}(y) \cdot \mathsf{g}(x) &
\end{array}
$$

The sequence of orientations gives rise to a process tree, where every branching point corresponds to a process split in an orient step. Part of the process tree developed during the described completion run is sketched in Figure 5.

3.2 Correctness

Before we can state properties of MKBtt runs, notions to track process splits in the course of a deduction are required.

Definition 5 Consider an MKBtt inference step $N \vdash N'$. If orient was applied the set of processes S which was split into two child processes is called the step's *split set*. For all other inference rules the split set is empty. For a step with split set S and

$p' \in \mathcal{P}(N')$, we define the *predecessor* of p' as

$$pred_S(p') = \begin{cases} p & \text{if } p' = p0 \text{ or } p' = p1 \text{ for some } p \in S \\ p' & \text{otherwise} \end{cases}$$

In Lemmata 1 and 2 we prove that an MKBtt step corresponds to a (possibly non-proper) KBtt step for every process occurring in some node, and every KBtt step can be modelled by MKBtt. Here, $\vdash^=$ denotes the reflexive closure of the KBtt inference relation $\vdash$.

Lemma 1 *For an MKBtt step $N \vdash N'$ with split set S the KBtt step*

$$(E[N,p], R[N,p], C[N,p]) \vdash^= (E[N',p'], R[N',p'], C[N',p']) \tag{1}$$

is valid for all $p' \in \mathcal{P}(N')$ such that $p = pred_S(p')$. Moreover, there exists at least one process $p' \in \mathcal{P}(N')$ for which the step is not an equality step if the rule applied in $N \vdash N'$ is not gc or subsume.

Proof By case analysis on the applied MKBtt rule in (1).

- Assume orient with split set S replaced the node $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ by $n' = \langle s : t, R_0 \cup R_{lr}, R_1 \cup R_{rl}, E', C_0 \cup R_{lr}, C_1 \cup R_{rl}\rangle$. Let p' be a process in $\mathcal{P}(N')$ and $p = pred_S(p')$ be its predecessor with respect to S. We have $E[N \setminus \{n\}, p] = E[N' \setminus \{n'\}, p']$, $R[N \setminus \{n\}, p] = R[N' \setminus \{n'\}, p']$ and $C[N \setminus \{n\}, p] = C[N' \setminus \{n'\}, p']$. These sets will in the sequel be denoted by E_i, R_i and C_i, respectively. A further case distinction reveals three possibilities:
  i. If $p' \in R_{lr}$, by definition of orient $R[n', p'] = C[n', p'] = \{s \to t\}$ and $E[n', p'] = \varnothing$. Inference (1) is thus a valid orient step in KBtt if p happens to be in E. Since p' occurs in R_{lr}, either $p' \in E_{lr} \setminus E_{rl}$ or $p' = p0$ for some $p \in S$. If $p' \in E_{lr} \setminus E_{rl}$ then $p \in E$ follows from $p = pred_S(p') = p'$ and $E_{lr} \subseteq E$. Otherwise $p = pred_S(p')$ entails $p' = p0$ such that $p \in S$ and because of $S \subseteq E$ also $p \in E$ holds. As p occurs in E one has $E[n,p] = \{s \approx t\}$ and—because of the node condition—$R[n,p] = C[n,p] = \varnothing$. Hence the KBtt inference step

  $$(E_i \cup \{s \approx t\}, R_i, C_i) \vdash_{\mathsf{KBtt}} (E_i, R_i \cup \{s \to t\}, C_i \cup \{s \to t\})$$

  is valid since $C_i \cup \{s \to t\} = C[N,p] \cup \{s \to t\}$ terminates according to the side condition of orient in MKBtt.
  ii. If $p' \in R_{rl}$, similar reasoning as in the previous case shows that the simulated inference step is

  $$(E_i \cup \{s \approx t\}, R_i, C_i) \vdash_{\mathsf{KBtt}} (E_i, R_i \cup \{t \to s\}, C_i \cup \{t \to s\})$$

  iii. Finally, if $p' \notin R_{lr} \cup R_{rl}$ then process p' was not affected in this inference step, so $p = p'$ and we have $E[n,p] = E[n',p']$, $R[n,p] = R[n',p']$ and $C[n,p] = C[n',p']$. The projection of the considered MKBtt inference to process p' is thus an identity step.

In all remaining cases $p = p'$ holds as no process splitting occurs.

- Whenever delete removes some node $\langle s : s, \varnothing, \varnothing, E, \varnothing, \varnothing\rangle$ then $s \approx s \in E[N,p]$ for all $p \in E$, and hence delete also applies in KBtt. For all $p \notin E$ an identity step is obtained.

– If deduce adds a node $\langle s : t, \varnothing, \varnothing, R \cap R', \varnothing, \varnothing\rangle$ then for all $p \in R \cap R'$ both $\ell \to r$ and $\ell' \to r'$ occur in $R[N, p]$. Hence deduce can also be applied in KBtt, yielding $s \approx t$ which is also contained in $E[N', p]$.
– Next, assume $\mathsf{rewrite}_1$ was used. For every process $p \notin (R_0 \cup E) \cap R$ an identity step is obtained. Otherwise, two cases can be distinguished which are distinct due to the node condition.
  i. If $p \in R_0 \cap R$ then $R[N, p]$ contains rules $s \to t$ and $\ell \to r$ such that $t \xrightarrow{\ell \to r} u$. Hence compose can be applied to replace $s \to t$ by $s \to u$, which is modelled in MKBtt by moving p from the rewrite label of a node with data $s : t$ to a node with data $s : u$.
  ii. If $p \in E \cap R$ there is an equation $s \approx t$ in $E[N, p]$ and a rule $\ell \to r$ in $R[N, p]$ such that $t \xrightarrow{\ell \to r} u$. Thus simplify can turn $s \approx t$ into $s \approx u$, and indeed $s \approx u$ instead of $s \approx t$ occurs in $E[N', p]$.
– In the case where $\mathsf{rewrite}_2$ was applied, the inference is an identity step for every process $p \notin (R_0 \cup R_1 \cup E) \cap R$. Otherwise, three distinct possibilities can be distinguished. If $p \in R_0 \cap R$ or $p \in E \cap R$ then compose or simplify can be applied, as argued in the case for $\mathsf{rewrite}_1$.
  iii. If $p \in R_1 \cap R$ then there are rules $\ell \to r$ and $t \to s$ in $R[N, p]$ such that the latter can be collapsed into an equation $s \approx u$ because $t \rhd \ell$. Hence $s \approx u$ belongs to $E[N', p]$ and $t \to s$ is not in $R[N', p]$.
– If gc was applied the step obviously corresponds to an identity step on the level of KBtt for every process $p \in \mathcal{P}(N')$, and the same holds for subsume.

Finally, for every inference rule the non-emptiness requirement for the set of affected labels ensures that the strict part $\vdash$ holds for at least one $p' \in \mathcal{P}(N')$. □

Lemma 2 *Assume for a KBtt inference step $(\mathcal{E}, \mathcal{R}, \mathcal{C}) \vdash (\mathcal{E}', \mathcal{R}', \mathcal{C}')$ there exist a node set N and a process p such that $\mathcal{E} = E[N, p]$, $\mathcal{R} = R[N, p]$ and $\mathcal{C} = C[N, p]$. Then there is some inference step $N \vdash N'$ with split set S and a process $p' \in \mathcal{P}(N')$ such that $p = pred_S(p')$, $\mathcal{E}' = E[N', p']$, $\mathcal{R}' = R[N', p']$ and $\mathcal{C}' = C[N', p']$.*

Proof In the following case analysis on the applied KBtt rule, $(*)$ refers to the proof obligations

$$\mathcal{E}' = E[N', p'],\ \mathcal{R}' = R[N', p'],\ \mathcal{C}' = C[N', p']$$

– Assume orient was applied to replace some equation $s \approx t \in \mathcal{E}$ by the rule $s \to t \in \mathcal{R}'$. Then there must be an node $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ in N such that $p \in E$ and $\mathcal{C} \cup \{s \to t\}$ terminates. We distinguish two further cases. If $\mathcal{C} \cup \{t \to s\}$ terminates as well, we set $S = \{p\}$. For $R_{lr} = \{p0\}$ and $R_{rl} = \{p1\}$ an application of orient yields

  $$N' = split_{\{p\}}(N \setminus \{n\}) \cup \{\langle s : t, R_0 \cup \{p0\}, R_1 \cup \{p1\}, E \setminus \{p\}, C_0 \cup \{p0\}, C_1 \cup \{p1\}\rangle\}$$

  For $p' = p0$ we have $p = pred_S(p')$, and $(*)$ is satisfied. If $\mathcal{C}[N, p] \cup \{t \to s\}$ does not terminate, we apply orient with $S = \varnothing$ and $R_{lr} = \{p\}$, which yields

  $$N' = (N \setminus \{n\}) \cup \{\langle s : t, R_0 \cup \{p\}, R_1, E \setminus \{p\}, C_0 \cup \{p\}, C_1\rangle\}$$

  Thus we have $p' = p$ which trivially satisfies $p = pred_S(p')$, and again $(*)$ holds.

In all remaining cases we can set $p' = p$ since no splitting occurs.

– If compose rewrites $s \to t$ to $s \to u$ using a rule $\ell \to r$, N contains nodes $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ and $\langle \ell : r, R, \dots\rangle$ such that $p \in R_0 \cap R$. Thus $\mathsf{rewrite}_1$ or $\mathsf{rewrite}_2$ applies, depending on whether $t \doteq \ell$ or $t \rhd \ell$. We obtain

$$N' = (N \setminus \{n\}) \cup \{\langle s : t, R_0 \setminus R, R_1', E \setminus R, C_0, C_1\rangle, \langle s : u, R_0 \cap R, \varnothing, E', \varnothing, \varnothing\rangle\}$$

where R_1' is R_1 or $R_1 \setminus R$, and E' is $E \cap R$ or $(E \cup R_1) \cap R$, determined by whether $\mathsf{rewrite}_1$ or $\mathsf{rewrite}_2$ is applied, respectively. Note that $(*)$ is satisfied.

– If simplify reduces an equation $s \approx t$ to $s \approx u$ using a rule $\ell \to r$, there are nodes $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ and $\langle \ell : r, R, \dots\rangle$ in N such that $p \in E \cap R$. If t is a variant of ℓ we can therefore use $\mathsf{rewrite}_1$ and otherwise $\mathsf{rewrite}_2$ to infer

$$\begin{aligned} N' = (N \setminus \{n\}) &\cup \{\langle s : t, R_0 \setminus R, R_1', E \setminus R, C_0, C_1\rangle\} \\ &\cup \{\langle s : u, R_0 \cap R, \varnothing, E', \varnothing, \varnothing\rangle\} \end{aligned}$$

where R_1' and E' depend on which inference rule applies. Since $p \in E \cap R$, $(*)$ holds.

– Assume collapse is applied to turn a rule $t \to s$ into an equation $u \approx s$ using $\ell \to r$. Then $t \rhd \ell$ must hold, and N contains nodes $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ and $\langle \ell : r, R, \dots\rangle$ such that p occurs in $R_1 \cap R$. To satisfy $(*)$ we can thus apply $\mathsf{rewrite}_2$ to obtain

$$\begin{aligned} N' = (N \setminus \{n\}) &\cup \{\langle s : t, R_0 \setminus R, R_1 \setminus R, E \setminus R, C_0, C_1\rangle\} \\ &\cup \{\langle s : u, R_0 \cap R, \varnothing, (E \cup R_1) \cap R, \varnothing, \varnothing\rangle\} \end{aligned}$$

– If delete removes some equation $s \approx s$ from $\mathcal{E}$ then N must contain a node $n = \langle s : s, R_0, R_1, E \uplus \{p\}, C_0, C_1\rangle$. Since the equation $s \approx s$ cannot be oriented into a terminating rule, the sets R_0, R_1, C_0 and C_1 must be empty. Thus n can be removed by delete in MKBtt.

– Finally, in the case where deduce generates $s \approx t$ from an overlap involving rules $\ell \to r$ and $\ell' \to r'$, there are nodes $\langle \ell : r, R, \dots\rangle$ and $\langle \ell' : r', R', \dots\rangle$ in N such that $p \in R \cap R'$. Applying deduce in MKBtt thus yields

$$N' = N \cup \{\langle s : t, \varnothing, \varnothing, R \cap R', \varnothing, \varnothing\rangle\}$$

such that $(*)$ is satisfied. □

Since MKBtt steps are reflected in KBtt, an MKBtt run γ of the form $N_0 \vdash^* N$ corresponds to a valid KBtt run γ_p for every process $p \in \mathcal{P}(N)$.

Definition 6 Consider an MKBtt run γ of the form $N_0 \vdash N_1 \vdash \cdots \vdash N_k$ and some process $p \in \mathcal{P}(N_k)$. We inductively define the sequence $p_0, \dots, p_k$ of *ancestors* of p by setting $p_k = p$ and $p_i = \mathit{pred}_{S_i}(p_{i+1})$ for $0 \leqslant i < k$, where S_i is the split set of the step $N_i \vdash N_{i+1}$. Let $\mathcal{E}_i$, $\mathcal{R}_i$ and $\mathcal{C}_i$ denote $E[N_i, p_i]$, $R[N_i, p_i]$ and $C[N_i, p_i]$, respectively. Then the *projected run* γ_p is the sequence

$$(\mathcal{E}_0, \mathcal{R}_0, \mathcal{C}_0) \vdash^= (\mathcal{E}_1, \mathcal{R}_1, \mathcal{C}_1) \vdash^= \cdots \vdash^= (\mathcal{E}_k, \mathcal{R}_k, \mathcal{C}_k)$$

According to Lemma 1, γ_p is a valid KBtt run for every process p.

Using projections, the definitions of success, failure and fairness given for KBtt can be naturally extended to MKBtt.

Definition 7 A finite MKBtt run γ of the form $N_0 \vdash^* N$

- is *fair* if γ_p is fair and nonfailing for some process $p \in \mathcal{P}(N)$,
- *succeeds* if $E[N,p] = \varnothing$ for some process $p \in \mathcal{P}(N)$, and
- *fails* if γ_p fails for all processes $p \in \mathcal{P}(N)$.

It is easy to see that MKBtt is sound in the sense that the equational theory is preserved.

Lemma 3 *Consider an MKBtt step $N \vdash N'$ with split set S and a process $q \in \mathcal{P}(N')$ with $p = pred_S(q)$. The relations $\leftrightarrow^*_{E[N,p] \cup R[N,p]}$ and $\leftrightarrow^*_{E[N',q] \cup R[N',q]}$ coincide.* □

As the simulation of KBtt with MKBtt is sound (Lemma 1) and complete (Lemma 2), it is straightforward to establish correctness and completeness using the corresponding results for KBtt. We call an MKBtt run $\gamma\colon N_0 \vdash^* N$ *simplifying* if the resulting system $R[N,p]$ is reduced whenever γ succeeds for some process p.

Theorem 4 *Let $N_{\mathcal{E}}$ be the initial node set for a set of equations $\mathcal{E}$ and let γ be a finite non-failing MKBtt run of the form $N_{\mathcal{E}} \vdash^* N$ which is fair for some $p \in \mathcal{P}(N)$. Then $R[N,p]$ is convergent.*

Proof According to Lemma 1 there is a corresponding KBtt run γ_p which is non-failing and fair. Since finite runs of KBtt are correct (Theorem 3), $R[N,p]$ is convergent. □

3.3 Completeness

Theorem 2 states the completeness of KB in the following sense: If a set of equations $\mathcal{E}$ admits an equivalent finite convergent rewrite system $\mathcal{R}$, any fair KB run will produce an equivalent finite convergent system if a reduction order compatible with $\mathcal{R}$ is used, provided the run does not fail. The following example shows that MKBtt might even fail if one uses a termination tool $\mathbb{T}$ that can prove the termination of $\mathcal{R}$.

Example 1 The convergent rewrite system $\mathcal{R}$ consisting of the rules

$$\begin{aligned} \mathsf{f}(\mathsf{h}(x,y)) &\to \mathsf{f}(\mathsf{i}(x,x)) & \mathsf{h}(\mathsf{a},\mathsf{a}) &\to \mathsf{c} \\ \mathsf{g}(\mathsf{i}(x,y)) &\to \mathsf{g}(\mathsf{h}(x,x)) & \mathsf{i}(\mathsf{a},\mathsf{a}) &\to \mathsf{c} \end{aligned}$$

is derived from the input equalities $\mathcal{E}$

$$\begin{aligned} \mathsf{f}(\mathsf{h}(x,y)) &\approx \mathsf{f}(\mathsf{i}(x,x)) & \mathsf{h}(\mathsf{a},\mathsf{a}) &\approx \mathsf{c} & \mathsf{h}(\mathsf{a},\mathsf{a}) &\approx \mathsf{i}(\mathsf{a},\mathsf{a}) \\ \mathsf{g}(\mathsf{i}(x,y)) &\approx \mathsf{g}(\mathsf{h}(x,x)) & \mathsf{i}(\mathsf{a},\mathsf{a}) &\approx \mathsf{c} \end{aligned}$$

in any fair run of standard completion that uses the reduction order $\to^+_{\mathcal{R}}$. The system $\mathcal{R}$ is easily shown to be terminating with a matrix interpretation of dimension 2; e.g. the termination tool $\mathsf{T_T T_2}$ using the strategy `matrix -ib 2 -d 2 -direct` immediately outputs a termination proof. However, if a KBtt run uses $\mathsf{T_T T_2}$ with this strategy and starts by orienting $\mathsf{h}(\mathsf{a},\mathsf{a}) \approx \mathsf{i}(\mathsf{a},\mathsf{a})$ then no matter which orientation is chosen, one of the equations in the leftmost column remains unorientable. Similarly, if MKBtt starts by applying orient to $\mathsf{h}(\mathsf{a},\mathsf{a}) \approx \mathsf{i}(\mathsf{a},\mathsf{a})$ then process ϵ gets split into 0 and 1. But in subsequent steps neither process can orient both of the equations in the leftmost column, so the run fails.

$$\textsf{orient}\quad \frac{N \cup \{\langle s : t, R_0, R_1, E, C_0, C_1\rangle\}}{\mathit{split}'(E_{lr}, E_{rl}, N) \cup \{\langle s : t, R_0 \cup R_{lr}, R_1 \cup R_{rl}, E', C_0 \cup R_{lr}, C_1 \cup R_{rl}\rangle\}}$$

if $E_{lr}, E_{rl} \subseteq E$, $E' = E \setminus (E_{lr} \cup E_{rl}) \cup \{p- \mid p \in E_{lr} \cup E_{rl}\}$, E_{lr} is the set of all processes $p \in E$ such that $\mathbb{T} \vDash C_p(N) \cup \{s \to t\}$, E_{rl} is the set of all processes p such that $\mathbb{T} \vDash C_p(N) \cup \{t \to s\}$, $R_{lr} = \{p0 \mid p \in E_{lr}\}$ and $R_{rl} = \{p1 \mid p \in E_{rl}\}$, and $E_{lr} \cup E_{rl} \neq \varnothing$

Fig. 6 The orient rule in $\mathsf{MKBtt_c}$.

This example shows that the order in which nodes are processed has considerable influence: orienting nodes too early can prevent KBtt and MKBtt from producing a convergent system even if a successful run exists. Nevertheless, completeness in this sense can be partially obtained in a slightly modified version of MKBtt which we will refer to as $\mathsf{MKBtt_c}$. In contrast to the previous version, a process can now also keep an equation unoriented. For this purpose, processes are now viewed as strings in $(0 + 1 + -)^*$. We write $\mathbb{T} \vDash \mathcal{R}$ if the *termination tool* $\mathbb{T}$ can verify termination of the rewrite system $\mathcal{R}$. The orient rule in $\mathsf{MKBtt_c}$ is given in Figure 6. Here, $\mathit{split}'(E_{lr}, E_{rl}, N)$ replaces every occurrence of a process $p \in E_{lr} \cap E_{rl}$ in a node of N by $\{p-, p0, p1\}$, every occurrence of $p \in E_{lr} \setminus E_{rl}$ by $\{p-, p0\}$ and every occurrence of $p \in E_{rl} \setminus E_{lr}$ by $\{p-, p1\}$. The notion of a split set in MKBtt is replaced by *split tuple*, which refers to the pair of process sets (E_{lr}, E_{rl}). For all inference steps that use a different rule than orient, the split tuple is $(\varnothing, \varnothing)$.

Example 2 If an $\mathsf{MKBtt_c}$ run on the input equalities from Example 1 starts by orienting $\mathsf{h}(\mathsf{a},\mathsf{a}) \approx \mathsf{i}(\mathsf{a},\mathsf{a})$, the resulting node is $\langle \mathsf{h}(\mathsf{a},\mathsf{a}) : \mathsf{i}(\mathsf{a},\mathsf{a}), \{0\}, \{1\}, \{-\}, \{0\}, \{1\}\rangle$. In contrast to MKBtt, a descendant of process $-$ can deliver a convergent system.

To obtain a completeness result for $\mathsf{MKBtt_c}$, we require a stronger notion of fairness which requires to equally advance all processes at some point.

Definition 8 Consider an equational proof P, a run $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ with $p \in \mathcal{P}(N_k)$, and let $\succ$ denote $\to^+_{C[N_k,p]}$. Then p *eventually simplifies* P *starting from* N_k if

- there is a proof Q in $(E[N_k, p], R[N_k, p])$ such that $P \Rightarrow^{\succ}_{\mathsf{KB}} Q$, or
- all direct successors $q \in \mathcal{P}(N_{k+1})$ of p eventually simplify P starting from N_{k+1}.

Thus a run γ with process $p \in \mathcal{P}(N_k)$ eventually simplifies a proof P if all successors of p in γ allow for a smaller proof at some point.

Definition 9 (Strong Fairness) A run $\gamma\colon N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ is *strongly fair* if for every $k \geqslant 0$, $p \in N_k$, and equational proof P in $(E[N_k, p], R[N_k, p])$ which is not in normal form, the following conditions hold:

- If N_k admits a step $N_k \vdash N$ such that $p \in \mathcal{P}(N)$ and there is an equational proof Q in $(E[N, p], R[N, p])$ satisfying $P \Rightarrow^{\succ}_{\mathsf{KB}} Q$, then p eventually simplifies P starting from N_k.
- If there exists an orient step $N_k \vdash N$ applied to node n such that N contains a successor p' of p and there is an equational proof Q in $(E[N, p'], R[N, p'])$ satisfying $P \Rightarrow^{\succ}_{\mathsf{KB}} Q$, then every successor q of p either performed an orient step on n and got extended by $-$ in this step, or eventually simplifies P from N_k.

Here $\succ$ denotes the reduction order $\rightarrow^{+}_{C[N_k,p]}$.

Intuitively, a strongly fair run requires all processes to simplify an equational proof if this simplification can be done without process splits (case i). Moreover, if an orient step on, say, a node with data $s : t$ allows for a simplification then all processes except the one that does not orient $s : t$ are required to perform this step (case ii). A sufficient condition for a run to be strongly fair is that all processes are advanced using a breadth-first strategy.

A termination tool $\mathbb{T}$ *covers* some reduction order $\succ$ if for any rewrite system $\mathcal{R}$ that is compatible with $\succ$, $\mathbb{T} \vDash \mathcal{R}$ holds.

Lemma 4 *Consider an* $\mathsf{MKBtt_c}$ *run* $\gamma\colon N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ *which employs a termination tool* $\mathbb{T}$ *covering some reduction order* $\succ$.

i. For every node set N_k *there exists a process* p_k *such that* $C[N_k, p_k] \subseteq \succ$ *and the sequence* $(E[N_k, p_k], R[N_k, p_k])_{k\geqslant 0}$ *is a valid* KB *run* γ_p *using* $\succ$.
ii. If γ *is strongly fair then* γ_p *is fair.*

Proof

i. We construct the process sequence $(p_k)_{k\geqslant 0}$ inductively such that

$$\begin{aligned}(E[N_k, p_k], R[N_k, p_k], C[N_k, p_k])\\ \vdash^{=} (E[N_{k+1}, p_{k+1}], R[N_{k+1}, p_{k+1}], C[N_{k+1}, p_{k+1}])\end{aligned} \qquad (*)$$

is a valid KBtt inference step and $C[N_k, p_k] \subseteq \succ$.
We start by setting $p_0 = \epsilon$. Now consider an inference step $N_k \vdash N_{k+1}$ with split tuple (S_0, S_1). If $p_k \notin S_0 \cup S_1$ then we take $p_{k+1} = p_k$. By a straightforward adaptation of Lemma 1 to $\mathsf{MKBtt_c}$ a corresponding KBtt (or empty) step $(*)$ is possible, and $C[N_k, p_k] \subseteq \succ$ follows from the induction hypothesis. Otherwise, we must have $p_k \in E$ for an inference step orienting a term pair $s : t$ (adopting the notation used in Figure 6). If $s \succ t$ then $\mathbb{T} \vDash C[N_k, p_k] \cup \{s \to t\}$ as $\mathbb{T}$ covers $\succ$. In this case we set $p_{k+1} = p_k 0$. Due to the side condition of orient, $p_k \in S_0$ and hence $p_{k+1} \in \mathcal{P}(N_{k+1})$. Again $(*)$ is a KBtt step and by the choice of p_{k+1} also $C[N_{k+1}, p_{k+1}] \subseteq \succ$ holds. The argument for the case $t \succ s$ is symmetric. If s and t are incomparable in $\succ$, we may choose $p_{k+1} = p_k -$. Then $(*)$ is an equality step and $C[N_{k+1}, p_{k+1}] \subseteq \succ$ follows from the induction hypothesis.
As the constructed sequence $(E[N_k, p_k], R[N_k, p_k], C[N_k, p_k])_{k\geqslant 0}$ constitutes a KBtt run which satisfies $C[N_k, p_k] \subseteq \succ$ for all $k \geqslant 0$, there is also a valid KB run $(E[N_k, p_k], R[N_k, p_k])_{k\geqslant 0}$ which uses $\succ$ as reduction order.

ii. Let $\mathcal{E}_\omega$ and $\mathcal{R}_\omega$ denote the persistent sets of γ_p. Suppose P is a proof in $(\mathcal{E}_\omega, \mathcal{R}_\omega)$ which is not a rewrite proof and there exists an inference $(\mathcal{E}_\omega, \mathcal{R}_\omega) \vdash (\mathcal{E}, \mathcal{R})$ such that $(\mathcal{E}, \mathcal{R})$ admits a proof Q satisfying $P \Rightarrow^{\succ}_{\mathsf{KB}} Q$. Then there must be a node set N_j in γ such that $(E[N_j, p_j], R[N_j, p_j])$ contains all equations and rules that are used in P together with those used when simplifying P to Q. By adapting Lemma 2 to $\mathsf{MKBtt_c}$, it follows that there is an inference step $N_j \vdash N$ such that $\mathcal{E}' = E[N, p']$, $\mathcal{R}' = R[N, p']$, and $C[N, p'] \subseteq \succ$ holds for some successor p' of p_j, and $(\mathcal{E}', \mathcal{R}')$ admits proof Q.
We distinguish two cases. If $(E[N_j, p_j], R[N_j, p_j]) \vdash (\mathcal{E}', \mathcal{R}')$ and thus $N_j \vdash N$ does not apply orient then no process splitting occurs and $p_j \in \mathcal{P}(N)$. By strong fairness, p_j eventually simplifies P. In particular, some successor p_m in the process sequence

$(p_k)_{k\geqslant 0}$ with $m \geqslant j$ has to provide a proof Q' in $(E[N_m, p_m], R[N_m, p_m])$ such that $P \Rightarrow^{\succ}_{\mathsf{KB}} Q'$. Therefore also γ_p allows for this simplified proof.
Now suppose $(E[N_j, p_j], R[N_j, p_j]) \vdash (\mathcal{E}', \mathcal{R}')$ applied orient to some equation $s \approx t$ and $s \succ t$ holds. By construction of the sequence $(p_k)_{k\geqslant 0}$ no successor of p_j can have obtained $-$ as part of its label when orienting a node with data $s : t$. Hence, according to strong fairness all successors of p_j have to eventually simplify P. So some p_m in $(p_k)_{k\geqslant 0}$ with $m \geqslant j$ has to provide a proof Q' in $(E[N_m, p_m], R[N_m, p_m])$ with $P \Rightarrow^{\succ}_{\mathsf{KB}} Q'$. Again this proof is reflected in γ_p, which proves fairness of this KB run. □

The following completeness result shows that an $\mathsf{MKBtt_c}$ run employing a sufficiently powerful termination prover can produce any convergent system which is derivable in a KB run.

Theorem 5 *Consider a finite canonical rewrite system $\mathcal{R}$ which can be constructed from $\mathcal{E}$ in a fair KB run using $\succ$. If $\mathbb{T}$ covers $\succ$ then any strongly fair and simplifying $\mathsf{MKBtt_c}$ run $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ which uses $\mathbb{T}$ and does not have a failing process develops some process $p \in \mathcal{P}(N_n)$ which satisfies $E[N_n, p] = \varnothing$ and $R[N_n, p] = \mathcal{R}$ (up to renaming variables).*

Proof According to Lemma 4 there is a sequence of processes $(p_k)_{k\geqslant 0}$ such that $(E[N_k, p_k], R[N_k, p_k])_{k\geqslant 0}$ is a fair KB run using $\succ$. By repeating the following argument of [**?**, Theorem 3.9], we will see that this run succeeds with system $\mathcal{R}$. Each rule $\ell \to r$ in $\mathcal{R}$ is a theorem in $\mathcal{E}$ and therefore will have a persisting rewrite proof after a finite number of steps in every fair and unfailing run. Let $\mathcal{R}' \subseteq \bigcup_i R[N_i, p_i]$ be the set of rules required for proofs of all rules in $\mathcal{R}$. Both $\mathcal{R}$ and $\mathcal{R}'$ are contained in $\succ$. Hence all these proofs must be of the form $\ell \to^*_{\mathcal{R}'} r$: Suppose r was reducible in $\mathcal{R}'$ to a term r' such that $r \succ r'$. Then there must also be a proof $r \leftrightarrow^*_{\mathcal{R}} r'$ as $\mathcal{R}$ is a convergent presentation of the theory. But $r \succ r'$ implies that r is reducible in $\mathcal{R}$, contradicting the assumption that $\mathcal{R}$ is canonical.

Thus $\to_{\mathcal{R}} \subseteq \to^+_{\mathcal{R}'}$ holds. As $\mathcal{R}$ and $\mathcal{R}'$ have the same equational theory, $\mathcal{R}'$ must be convergent and hence canonical since it was constructed by a simplifying run. Thus $\mathcal{R}$ and $\mathcal{R}'$ have to be equal, because the canonical rewrite system compatible with a given reduction order is unique (up to variable renaming) [**?**]. □

Note that even if $\mathbb{T}$ covers $\succ$, an $\mathsf{MKBtt_c}$ run might still fail if the wrong strategy is chosen. For example, a run on the equations $\mathcal{E}$

$$\mathsf{a} \approx \mathsf{b} \qquad \mathsf{a} \approx \mathsf{c} \qquad \mathsf{f(b)} \approx \mathsf{b} \qquad \mathsf{f(a)} \approx \mathsf{d}$$

where $\mathbb{T}$ covers the lexicographic path order (LPO) with a precedence satisfying $\mathsf{a} > \mathsf{b}$, $\mathsf{a} > \mathsf{c}$, $\mathsf{b} > \mathsf{d}$ and $\mathsf{c} > \mathsf{d}$ may succeed with the convergent rewrite system $\mathcal{R}$

$$\mathsf{a} \to \mathsf{d} \qquad \mathsf{b} \to \mathsf{d} \qquad \mathsf{c} \to \mathsf{d} \qquad \mathsf{f(d)} \to \mathsf{d}$$

if a suitable strategy is adopted, but fail with the unorientable equation $\mathsf{b} \approx \mathsf{c}$ if the equations are processed in an unfortunate order [**?**]. This example also illustrates that for completeness it is not sufficient to require that $\mathbb{T}$ can prove termination of $\mathcal{R}$, or covers $\to^+_{\mathcal{R}}$; any run on $\mathcal{E}$ where $\mathbb{T}$ only supports the reduction order $\to^+_{\mathcal{R}}$ fails immediately.

4 Critical Pair Criteria

In order to limit the number of deduced equations during a completion run, several *critical pair criteria* were proposed as a means to filter out critical pairs that can be ignored without compromising completeness [**?**,**?**,**?**,**?**]. In a later work, Bachmair and Dershowitz [**?**] showed that these criteria match the more general pattern of *compositeness*. Before describing the use of critical pair criteria in MKBtt, the relevant definitions and some concrete criteria are recalled. We consider a fixed reduction order $\succ$, a proof order $\succcurlyeq$ and a proof reduction relation $\Rightarrow$. For details the reader is referred to [**?**].

4.1 Critical Pair Criteria in Standard Completion

A critical pair criterion *CPC* is a mapping from sets of equations to sets of equations such that $CPC(\mathcal{E})$ is a subset of $\mathrm{CP}(\mathcal{E})$. Intuitively, $CPC(\mathcal{E})$ contains those critical pairs that are considered redundant. A run $(\mathcal{E}_0, \mathcal{R}_0) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash (\mathcal{E}_2, \mathcal{R}_2) \vdash \cdots$ using reduction order $\succ$ is *fair with respect to CPC* if for every peak P associated with a critical pair in $\mathrm{CP}(\mathcal{R}_\omega) \setminus \bigcup_i CPC(\mathcal{R}_i \cup \mathcal{E}_i)$ there exists a proof Q in $(\mathcal{R}_i, \mathcal{E}_i)$ for some $i > 0$ such that $P \Rightarrow Q$. A critical pair criterion *CPC* is *correct* if a nonfailing run is fair in the general sense whenever it is fair with respect to *CPC*. Clearly, correct critical pair criteria allow to filter out unnecessary critical pairs without compromising completeness.

An equational proof P that has the form of a peak $s \leftarrow u \rightarrow t$ is *composite* if there exist terms $u_0, \ldots, u_{n+1}$ where $s = u_0$ and $t = u_{n+1}$ and proofs $P_0, \ldots, P_n$ such that P_i proves $u_i \approx u_{i+1}$ and $P \succcurlyeq P_i$ holds for all $1 \leqslant i \leqslant n$. The *compositeness criterion* returns all critical pairs among equations in $\mathcal{E}$ for which the associated overlaps are composite, which was proven to be correct [**?**]. This very general criterion is hard to apply in practice. However, some of the earlier proposals to filter out superfluous critical pairs in completion procedures actually capture special cases of compositeness.

Kapur *et al.* [**?**] introduced the notion of primality for critical pairs. An overlap $t\sigma \leftarrow s\sigma = s\sigma[u\sigma]_p \rightarrow s\sigma[v\sigma]_p$ between rules $s \rightarrow t$ and $u \rightarrow v$ in $\mathcal{R}$ is *prime* if $s\sigma$ is not reducible at some position strictly below p. The primality criterion $\mathrm{PCP}(\mathcal{R})$ returns all critical pairs among rules in $\mathcal{R}$ for which the associated overlaps are not prime. A special case of PCP is captured by the *unblockedness criterion* BCP [**?**]. A critical pair originating from an overlap $t\sigma \leftarrow s\sigma = s\sigma[u\sigma]_p \rightarrow s\sigma[v\sigma]_p$ is *blocked* if $x\sigma$ is irreducible in $\mathcal{R}$ for all variables $x \in \mathcal{V}\mathrm{ar}(s) \cup \mathcal{V}\mathrm{ar}(u)$. The set $\mathrm{BCP}(\mathcal{R})$ contains all unblocked critical pairs among rules in $\mathcal{R}$.

Küchlin [**?**] introduced the notion of *connectedness* to limit equational consequences deduced in a completion procedure. A critical pair $s \approx t$ originating from an overlap $s \leftarrow u \rightarrow t$ is *connected below* u if there exists an equational proof $s = u_0 \leftrightarrow u_1 \leftrightarrow \cdots \leftrightarrow u_n = t$ such that $u \succ u_i$ for all $1 \leqslant i < n$. Clearly, if a critical pair $s \approx t$ is connected below u such that $n > 1$ then it is also composite. For a practical criterion, Küchlin assumes $\rightarrow \subseteq \succ$ and concentrates on finding connecting sequences $u_1, \ldots, u_{n-1}$ such that $u \rightarrow^+ u_i$. As a special case the following *weak connectivity test* is proposed. Given an overlap $t\sigma \leftarrow s\sigma = s\sigma[u\sigma]_p \rightarrow s\sigma[v\sigma]_p$ between rules $s \rightarrow t$ and $u \rightarrow v$, the associated critical pair is connected if there exists a reduction step $s\sigma \rightarrow w$ using a rule $\ell \rightarrow r$ at position q fulfilling the (non-exclusive) properties: (i) if $q \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(s)$ then the critical overlap $\langle s \rightarrow t, q, \ell \rightarrow r \rangle$ is already considered, (ii) if $q = pq'$ and $q' \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(u)$ then $\langle \ell \rightarrow r, q', u \rightarrow v \rangle$ is already considered, and (iii) if $p = qp'$ and

deduce
$$\frac{N}{N \cup \{\langle s : t, \varnothing, \varnothing, E, \varnothing, \varnothing\rangle\}}$$
if there exist nodes $\langle \ell : r, R, \dots\rangle, \langle \ell' : r', R', \dots\rangle \in N$ and an overlap o involving rules $\ell \to r$ and $\ell' \to r'$ that gives rise to a critical pair $s \approx t$ such that $E = CPC_m(o, R \cap R', N) \neq \varnothing$

Fig. 7 The deduce inference rule using a critical pair criterion.

$p' \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(l)$ then $\langle u \to v, p', \ell \to r\rangle$ is already considered. This criterion is generalized to a full connectivity test where the critical pair is connected via an arbitrary sequence $w_0, \dots, w_n$ instead of a single intermediate term w. In the sequel the connectedness criterion returning connected critical pairs among rules in $\mathcal{R}$ will be referred to as CCP($\mathcal{R}$).

Since both CCP and PCP are special cases of compositeness, these criteria can also be combined. This *mixed* criterion that filters out critical pairs that are redundant according to one of the criteria will in the sequel be referred to as MCP.

4.2 Critical Pair Criteria in MKBtt

In the following paragraphs we describe how critical pair criteria can be integrated into MKBtt.

Definition 10 Given a KB critical pair criterion CPC, the corresponding MKBtt *critical pair criterion* CPC_m maps an overlap o with associated critical pair $s \approx t$, a set of processes E and a node set N to a process set $CPC_m(o, E, N) = E'$ such that $E' \subseteq E$ and $s \approx t \in E[N, p] \setminus CPC(E[N, p])$ for all $p \in E'$.

Intuitively, the set E' contains all processes in E for which the critical pair derived from o is not superfluous. Thus, in the deduce rule for MKBtt the equation label of the new node is filtered by the criterion as shown in Figure 7.

Consider a finite MKBtt run γ of the form $N_0 \vdash^* N_k$ and a process $p \in N_k$. Let p_i denote the ancestor of p in N_i and let $\succ$ denote the reduction order $\to^+_{C[N_k,p]}$. Then we call γ *fair with respect to* CPC_m *and* p if the following condition holds: Whenever a node set N_i gives rise to an overlap o with critical pair $s \approx t$ as described in Figure 7 and $p_i \in E \setminus CPC_m(o, E, N_i)$ then there exists a proof Q in some $(E[N_j, p_j], R[N_j, p_j])$ for $j > 0$ such that $s \approx t \Rightarrow^{\succ}_{\mathsf{KB}} Q$ holds. The run γ is *fair with respect to* CPC_m if it is fair with respect to CPC_m and some process $p \in \mathcal{P}(N_k)$. An MKBtt *critical pair criterion* CPC_m is *correct* if every finite non-failing run γ that is fair with respect to CPC_m is also fair in the sense of Definition 7.

Lemma 5 *Every* MKBtt critical pair criterion CPC_m *obtained from a correct criterion* CPC *is correct.*

Proof Let γ be a finite non-failing run of the form $N_0 \vdash^* N_k$ which is fair with respect to CPC_m and some process $p \in \mathcal{P}(N_k)$, and let p_i denote the ancestor of p in N_i. Assume a critical overlap o where $p_i \in E \setminus CPC_m(o, E, N_i)$ for some ancestor p_i of p. By definition there exists a proof Q in some $(E[N_j, p_j], R[N_j, p_j])$ such that $s \approx t \Rightarrow^{\succ}_{\mathsf{KB}} Q$. Hence the projected run γ_p is fair with respect to CPC and by correctness of CPC it is also fair. Thus γ is fair as well. □

Thus the use of MKBtt criteria obtained from correct standard criteria does not compromise completeness, and the chance of a run to succeed is not influenced. The following example illustrates the use of critical pair criteria in MKBtt.

Example 3 An MKBtt run on CGE_2 encounters the nodes

$$\langle \mathsf{e} \cdot x : x, P_0 \cup P_1, \ldots \rangle \quad (1)$$
$$\langle \mathsf{i}(x) \cdot x : \mathsf{e}, P_0 \cup P_1, \ldots \rangle \quad (2)$$
$$\langle x \cdot \mathsf{e} : x, P_0, \ldots \rangle \quad (14)$$
$$\langle \mathsf{i}(\mathsf{e}) : \mathsf{e}, P_0, \ldots \rangle \quad (19)$$
$$\langle y \cdot \mathsf{i}(x \cdot y) : \mathsf{i}(x), P_0 \cup P_1, \ldots \rangle \quad (21)$$

The overlap $\langle y \cdot \mathsf{i}(x \cdot y) \to \mathsf{i}(x), \epsilon, \mathsf{e} \cdot x \to x \rangle$ produces the critical pair $\mathsf{i}(x) \approx \mathsf{i}(x \cdot \mathsf{e})$ for the set of processes $P_0 \cup P_1$. When PCP_m is applied, it is checked whether there exists a node which allows to reduce the term $u = \mathsf{e} \cdot \mathsf{i}(x \cdot \mathsf{e})$ at some position below the root. Since node (14) can reduce u at position 21, the critical pair is recognized as redundant for all processes in P_0 such that the deduced node is $\langle \mathsf{i}(x) : \mathsf{i}(x \cdot \mathsf{e}), \varnothing, \varnothing, P_1, \varnothing, \varnothing \rangle$.

Furthermore, the overlap $\langle \mathsf{i}(x) \cdot x \to \mathsf{e}, 1, \mathsf{i}(\mathsf{e}) \to \mathsf{e} \rangle$ between nodes (2) and (19) gives rise to the critical pair $\mathsf{e} \approx \mathsf{e} \cdot \mathsf{e}$ for the process set P_0. To reduce the term $\mathsf{i}(\mathsf{e}) \cdot \mathsf{e}$ also node (14) can be applied at the root position. While PCP_m is not applicable since the overlap position is below ϵ, CCP_m requires to check the critical pairs involved in the decomposition. Indeed, the critical pair $\mathsf{e} \leftarrow \mathsf{i}(\mathsf{e}) \cdot \mathsf{e} \to \mathsf{i}(\mathsf{e})$ between nodes (2) and (14) is already covered by node (19) and the peak involving nodes (14) and (19) can be ignored since it is just a variable overlap. Hence the deduce step is superfluous.

The critical pair criteria PCP, BCP, and CCP require to check whether an overlapped term can be reduced in a certain way other than indicated by the overlap itself. Since MKBtt allows to check reducibility for multiple processes at once, the redundancy checks required for the respective multi-completion criteria can be shared among multiple processes.

5 Isomorphisms

The performance of our tool mkbTT is significantly affected by the number of simulated processes. On some input problems, runs exhibit similar process pairs which have the same probability of success.

Example 4 A run on CGE_2 may generate a node set N with process p where $E[N, p]$ consists of the equations

$$(x \cdot y) \cdot z \approx x \cdot (y \cdot z) \qquad \mathsf{f}(\mathsf{e}) \approx \mathsf{e}$$
$$\mathsf{g}(x) \cdot \mathsf{f}(y) \approx \mathsf{f}(y) \cdot \mathsf{g}(x) \qquad \mathsf{g}(\mathsf{e}) \approx \mathsf{e}$$

and $R[N, p] = C[N, p]$ consists of the rewrite rules

$$\mathsf{e} \cdot x \to x \qquad \mathsf{f}(x \cdot y) \to \mathsf{f}(x) \cdot \mathsf{f}(y)$$
$$\mathsf{i}(x) \cdot x \to \mathsf{e} \qquad \mathsf{g}(x \cdot y) \to \mathsf{g}(x) \cdot \mathsf{g}(y)$$

If an inference step $N \vdash N'$ applies orient to the equation $\mathsf{g}(x) \cdot \mathsf{f}(y) \approx \mathsf{f}(y) \cdot \mathsf{g}(x)$, the process p is split as both orientations are possible. But the states of the emerging child processes $p0$ and $p1$ are the same up to interchanging f and g. Hence further deductions of these processes will be symmetric.

Such similarities between processes are generally captured by *isomorphisms*.

Definition 11 A bijection $\theta\colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F},\mathcal{V})$ extends to an *isomorphism* between rewrite systems $\mathcal{R}$ and $\mathcal{R}'$ if $\mathcal{R}' = \{\theta(\ell) \to \theta(r) \mid \ell \to r \in \mathcal{R}\}$ such that $s \to_{\mathcal{R}} t$ if and only if $\theta(s) \to_{\mathcal{R}'} \theta(t)$ for all terms s and t. Two sets of equations $\mathcal{E}$ and $\mathcal{E}'$ are isomorphic with respect to θ if $\mathcal{E}' = \{\theta(u) \approx \theta(v) \mid u \approx v \in \mathcal{E}\}$ and for all terms s and t, $s \approx_{\mathcal{E}} t$ if and only if $\theta(s) \approx_{\mathcal{E}'} \theta(t)$. These concepts are expressed by writing $\mathcal{R} \cong_\theta \mathcal{R}'$ and $\mathcal{E} \cong_\theta \mathcal{E}'$, respectively. Two MKBtt processes p and q are isomorphic in a node set N if there exists some isomorphism θ such that $E[N,p] \cong_\theta E[N,q]$, $R[N,p] \cong_\theta R[N,q]$ and $C[N,p] \cong_\theta C[N,q]$.

Lemma 6 *Let N_p and N_q be node sets containing processes p and q such that*

$$(E[N_p,p], R[N_p,p], C[N_p,p]) \cong_\theta (E[N_q,q], R[N_q,q], C[N_q,q])$$

If there is a step $N_p \vdash N_p'$ such that p is the predecessor of $p' \in \mathcal{P}(N_p')$ then there is also an inference step $N_q \vdash^{=} N_q'$ and a process $q' \in \mathcal{P}(N_q')$ such that q is the predecessor of q' and

$$(E[N_p',p'], R[N_p',p'], C[N_p',p']) \cong_\theta (E[N_q',q'], R[N_q',q'], C[N_q',q'])$$

Proof If p was not affected by the step $N_p \vdash N_p'$, that is $(E[N_p,p], R[N_p,p], C[N_p,p])$ coincides with $(E[N_p',p], R[N_p',p], C[N_p',p])$ and $p \in \mathcal{P}(N_p')$, then we can set $N_q' = N_q$. Otherwise a *mirror step* $N_q \vdash N_q'$ using the same inference rule can model the step for q. More precisely, the mirror step is defined by case distinction on the rule applied in $N_p \vdash N_p'$.

- Assume orient turned a node $n = \langle s : t, R_0, R_1, E, C_0, C_1\rangle$ into $\langle s : t, R_0 \cup R_{lr}, R_1 \cup R_{rl}, E', C_0 \cup R_{lr}, C_1 \cup R_{rl}\rangle$ using split set S. Then a node n' with data $\theta(s) : \theta(t)$ has to occur in N_q since $E[N_q,q] \cong E[N_p,p]$ holds by assumption. Three further possibilities can be distinguished:
  Assume $p = p'$ and $p \in E_{lr} \setminus S$ because $C[N_p,p] \cup \{s \to t\}$ terminates, but $C[N_p,p] \cup \{t \to s\}$ does not. Thus also $C[N_q,q] \cup \{\theta(s) \to \theta(t)\}$ terminates, but $C[N_q,q] \cup \{\theta(t) \to \theta(s)\}$ does not. So the mirror step $N_q \vdash N_q'$ can apply orient to node n' with $E_{lr} = R_{rl} = \{q\}$ and $E_{rl} = R_{rl} = \varnothing$. In the second case where $p = p'$ and $p \in E_{rl} \setminus S$, we reason symmetrically to the preceding case. For the final case, let $p \in S$. We orient n' to obtain the mirror step $N_q \vdash N_q'$. Because $C[N_p,p] \cong C[N_q,q]$ holds, both orientations terminate, so q gets split into $q0$ and $q1$ which are then isomorphic to $p0$ and $p1$, respectively.

All remaining inference rules do not split processes so $p = p'$ and thus $q = q'$.

- Assume delete was applied to a node $\langle s : s, \varnothing, \varnothing, E, \varnothing, \varnothing\rangle$ where $p \in E$. Then there must be a node n' of the form $\langle \theta(s) : \theta(s), \varnothing, \varnothing, E', \varnothing, \varnothing\rangle$ in N_q such that $q \in E'$, and $N_q \vdash N_q'$ will be a delete step removing n'.

– If deduce created a node with data $\langle s : t, \varnothing, \varnothing, R \cap R', \varnothing, \varnothing\rangle$ originating from a critical pair involving nodes with terms $\ell : r$ and $\ell' : r'$, due to $R[N_p, p] \cong R[N_q, q]$, there must be nodes with data $\langle \theta(\ell) : \theta(r), R, \ldots\rangle$ and $\langle \theta(\ell') : \theta(r'), R', \ldots\rangle$ in N_q which allow in a mirror step $N_q \vdash N'_q$ to deduce a node $\langle \theta(s) : \theta(t), \varnothing, \varnothing, R \cap R', \varnothing, \varnothing\rangle$.
– If $\mathsf{rewrite}_1$ or $\mathsf{rewrite}_2$ was applied to a node $\langle s : t, R_0, R_1, E, C_0, C_1\rangle$ using a rule node $\langle \ell : r, R, \ldots\rangle$ to create $\langle s : u, R'_0, \varnothing, E', \varnothing, \varnothing\rangle$, then the mirror step $N_q \vdash N'_q$ can apply the same rule to nodes with data $\theta(s) : \theta(t)$ and $\theta(\ell) : \theta(r)$, which exist by assumption. Then q is removed from the rewrite or equation label of the node with data $\theta(s) : \theta(t)$, and occurs now in a node with data $\theta(s) : \theta(u)$. □

Theorem 6 *Let N_i be a set of nodes containing isomorphic processes $p_i, q_i \in \mathcal{P}(N_i)$. Assume there exists an MKBtt completion run γ of the form $N_i \vdash^* N_k$ and a process $p_k \in \mathcal{P}(N_k)$ such that p_i is the ancestor of p_k in N_i, and the projected run γ_{p_k} is fair and successful. Then there is also a fair deduction γ' of the form $N_i \vdash^* N'_k$ producing a process $q_k \in \mathcal{P}(N'_k)$ such that q_i is an ancestor of q_k, and also γ'_{q_k} is fair and successful.*

Proof We show by induction on the length of $\gamma\colon N_i \vdash^* N_k$ that there exists a sequence $\gamma'\colon N_i = N'_i \vdash^* N'_k$ with processes $q_j \in N'_j$ such that $(E[N_k, p_k], R[N_k, p_k], C[N_k, p_k])$ is isomorphic to $(E[N'_k, q_k], R[N'_k, q_k], C[N'_k, q_k])$. For the case where $k = i$ the processes p_k and q_k are by assumption isomorphic via some mapping θ. If $k > i$ we consider a sequence $N_i \vdash^* N_{k-1} \vdash N_k$ where p_i is the ancestor of p_k in N_i. By the induction hypothesis there is a sequence $N_i = N'_i \vdash^* N'_{k-1}$ with processes $q_{k-1} \in N'_{k-1}$ such that $(E[N_{k-1}, p_{k-1}], R[N_{k-1}, p_{k-1}], C[N_{k-1}, p_{k-1}])$ is isomorphic with respect to θ to $(E[N'_{k-1}, q_{k-1}], R[N'_{k-1}, q_{k-1}], C[N'_{k-1}, q_{k-1}])$. By Lemma 6, the last step $N_{k-1} \vdash N_k$ can be mirrored by $N'_{k-1} \vdash^= N'_k$ such that N'_k contains a process q_k for which $(E[N_k, p_k], R[N_k, p_k], C[N_k, p_k])$ and $(E[N'_k, q_k], R[N'_k, q_k], C[N'_k, q_k])$ are again isomorphic via θ. Note that $N'_{k-1} = N'_k$ if $N_{k-1} \vdash N_k$ did not affect p_{k-1}.

Hence given $\gamma\colon N_i \vdash N_{i+1} \vdash \cdots \vdash N_k$ with $p_k \in \mathcal{P}(N_k)$ there is a run $\gamma'\colon N_i \vdash^= N'_{i+1} \vdash^= \cdots \vdash^= N'_k$ with $q_k \in \mathcal{P}(N'_k)$ such that $(E[N_k, p_k], R[N_k, p_k], C[N_k, p_k])$ is isomorphic to $(E[N'_k, q_k], R[N'_k, q_k], C[N'_k, q_k])$. As p_k succeeds in N_k we must have $E[N_k, p_k] = \varnothing$ and thus also $E[N'_k, q_k] = \varnothing$. Since all intermediate states of p_k and q_k are isomorphic, γ' is fair for q_k. □

If our tool mkbTT detects two isomorphic processes in the current node set N then one process is deleted from all nodes in N. We exploit two concrete shapes of symmetries. *Renaming isomorphisms* swap function symbols as in Example 4, where $p0$ and $p1$ are isomorphic under the mapping θ that exchanges f and g. *Argument permutations* associate with every function symbol f of arity n a permutation π_f of the set $\{1, \ldots, n\}$. Then the mapping on terms defined by $\theta(x) = x$ and $\theta(f(t_1, \ldots, t_n)) = f(\theta(t_{\pi_f(1)}), \ldots, \theta(t_{\pi_f(n)}))$ also induces an isomorphism. For example, when completing SK90-3.02 [**?**] a process with state

$$(x + y) + z \approx x + (y + z) \qquad \begin{array}{rcl} \mathsf{f}(\mathsf{f}(x)) & \to & x \\ \mathsf{f}(x + y) & \to & \mathsf{f}(x) + \mathsf{f}(y) \end{array}$$

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

```
procedure mkbTT(No, Nc)
if ∃ successful process p then return p
else if No = ∅ then fail
else n := choose(No) ;
     No := (No \ {n}) ⊎ rewrite({n}, Nc) ;
     if n ≠ ⟨..., ∅, ∅, ∅, ∅, ∅⟩ then
        (n, No, Nc) := orient(n, No, Nc) ;
        if n ≠ ⟨..., ∅, ∅, ..., ..., ...⟩ then
           No := No ⊎ delete(rewrite(Nc, {n})) ;
           Nc := gc(Nc) ;
           Nd := deduce(n, Nc) ;
           No := No ⊎ gc(delete(Nd ⊎ rewrite(Nd, Nc))) ;
           Nc := Nc ⊎ {n} ;
mkbTT(No, Nc) ;
```

Fig. 8 Procedure implementing MKBtt.

6 Implementation

The inference system described in the previous sections is implemented in our tool mkbTT. The tool is written in the programming language OCaml. Binaries and sources are available from the tool's website

`http://cl-informatik.uibk.ac.at/mkbtt/`

where also a web interface can be found. In the following sections we provide implementation details which were found to be of special importance.

6.1 Control

The basic control of mkbTT is a multi-completion variant of a DISCOUNT loop, very similar to the one originally proposed for completion with multiple reduction orders [**?**]. Pseudo-code describing the control loop is given in Figure 8. The procedure advances two node sets containing *open* nodes *No* and *closed* nodes *Nc*, corresponding to passive and active facts. Intuitively, closed nodes have been fully exploited with respect to the orient and $\mathsf{rewrite}_{1,2}$ inference rules, and to every pair of closed nodes deduce has been applied exhaustively. Therefore, at the beginning of a run *Nc* is empty whereas *No* contains the set of initial nodes.

We briefly describe the components occurring in the main control loop. At the beginning of each recursive call it is checked whether some process succeeded. For this purpose, a process p is considered *successful* if it does not occur in an open node or in a closed equation label, i.e., all of $E[Nc, p]$, $R[No, p]$ and $E[No, p]$ are empty. If no successful process exists but there are open nodes left, *choose* selects a node n from the set of open nodes. Different selection strategies are considered for this purpose (see Section 6.4). The function $rewrite(N, N')$ applies $\mathsf{rewrite}_{1,2}$ to nodes in N by employing nodes in N' as rules. Nodes are implemented as mutable structures, so the original objects are modified and only newly created nodes are returned. The function call $orient(n, No, Nc)$ is used to apply the orient inference to node n. It returns the modified node along with the node sets *No* and *Nc* adapted by the *split* operation. Immediately after *rewrite* and *deduce* calls, *delete* is invoked to filter out nodes with

equal terms. After having been subjected to *rewrite*, all labels in a node might become empty. The purpose of *gc* is to filter out such nodes. The call $deduce(n, Nc)$ returns all equational consequences originating from deduce inference steps involving node n together with some node from Nc. To avoid redundant nodes, the union operation $\uplus$ exploits the subsume inference.

6.2 Termination Checking

Our tool supports two possibilities for termination checks in orient inference steps. They can either be performed internally by interfacing T$_{\mathsf{T}}$T$_2$ [**?**] with the user's favourite termination strategy supplied in T$_{\mathsf{T}}$T$_2$'s strategy language. Alternatively, any external termination checker can be used which complies to a minimal interface: It must take as argument the name of a file specifying the termination problem in TPDB format[2] and print `YES` on the first line of the output if termination could be established.

6.3 Term Indexing

Automated deduction tools typically spend a major part of computation time on deducing equational consequences and rewriting. A variety of sophisticated term indexing techniques [**?**] have been developed in order to speed up filtering out matching and unifiable terms. Also mkbTT relies on indexing techniques to quickly sift through nodes that are applicable for inferences. For instance, for deduce inferences the retrieval of unifiable (sub)terms is needed. For rewrite$_1$ steps, variants have to be retrieved and rewrite$_2$ requires encompassment retrieval, where the latter can be achieved by repeatedly retrieving subsumptions (also referred to as generalizations in the literature). To select unifiable terms, mkbTT implements *path indexing* and *discrimination trees* [**?**,**?**], for variant and subsumption retrieval also *code trees* [**?**] are supported.

6.4 Selection Strategies

An iteration of mkbTT's main control loop starts by selecting a node to process next. For this purpose a *choice* function picks the node where a given cost heuristic evaluates to a minimal value. The measure applied in this selection has significant impact on performance. To allow for a variety of possibilities, a strategy language is defined that is general enough to cover selection strategies that proved to be useful, but also captures some concepts used in choice strategies of other tools. A strategy is specified by the grammar rule

$$strategy \ ::= \ \texttt{?} \mid \texttt{(}node\texttt{,}strategy\texttt{)} \mid float\texttt{(}strategy\ \texttt{:}\ strategy\texttt{)}$$

[2] `http://www.lri.fr/~marche/tpdb/`

Here *node* refers to a node property, which is in turn defined via properties of process sets, processes, sets of equations, rewrite systems and term pairs:

$$\begin{aligned}
\mathit{node} &::= \texttt{data(}\mathit{term_pair}\texttt{)} \mid \texttt{el(}\mathit{process_set}\texttt{)} \mid \texttt{-}\,\mathit{node} \mid \mathit{node}\ \texttt{+}\ \mathit{node} \mid \texttt{*} \\
\mathit{process_set} &::= \texttt{min(}\mathit{process}\texttt{)} \mid \texttt{sum(}\mathit{process}\texttt{)} \mid \texttt{\#} \\
\mathit{process} &::= \mathit{process}\ \texttt{+}\ \mathit{process} \mid \texttt{e(}\mathit{eqs}\texttt{)} \mid \texttt{r(}\mathit{trs}\texttt{)} \mid \texttt{c(}\mathit{trs}\texttt{)} \\
\mathit{eqs} &::= \texttt{sum(}\mathit{term_pair}\texttt{)} \mid \texttt{\#} \\
\mathit{trs} &::= \texttt{sum(}\mathit{term_pair}\texttt{)} \mid \texttt{cp(}\mathit{eqs}\texttt{)} \mid \texttt{\#} \\
\mathit{term_pair} &::= \texttt{smax} \mid \texttt{ssum}
\end{aligned}$$

The properties forming the basic elements of strategies are captured by integer values. The following paragraphs explain the different components.

- 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 data $s : t$, or a process set property *pp* of its equation labels E, which is written as `el(`*pp*`)`. Node properties can also be added or inverted.
- A *process set property* takes either the sum or the minimum over a property of the involved processes, or may simply be the number of processes it contains, which is denoted by `#`.
- Given a current node set N, a *process property* of a process p may be an equation set property *ep* of its equation projection $E[N, p]$ or a rewrite system property *tp* of either its rule projection $R[N, p]$ or its constraint projection $C[N, p]$, which is expressed by writing `e(`*ep*`)`, `r(`*tp*`)` and `c(`*tp*`)`, respectively. Process properties can also be added.
- An *equation set property* of a set of equations $\mathcal{E}$ can be its cardinality (`#`) or the sum over a term pair property of the contained equations. A *rewrite system property* of a rewrite system $\mathcal{R}$ can additionally be a property of its set of critical pairs $\mathrm{CP}(\mathcal{R})$.
- A *term pair property* of $s : t$ can be the sum $|s| + |t|$ or maximum $\max\{|s|, |t|\}$ of the sizes of the involved terms, which is specified by writing `ssum` and `smax`.
- Finally, properties are combined to obtain selection strategies. The simplest possible strategy is `?`, which is implemented by picking a node randomly. Alternatively, a strategy may combine a node property *np* with another strategy *s* to a tuple `(`*np*`,`*s*`)`. By using this rule multiple times, a node property list of the form `(`np_1`,` $\ldots$ `(`np_k`,?)` $\ldots$ `)` can be constructed. To mix strategies, a strategy can also be of the shape r`(`s_1`:`s_2`)`, where r is assumed to be a rational value in the closed interval $[0, 1]$. Node property lists are evaluated by mapping each node to the corresponding tuple of integer values, its *cost*, and choosing the (lexicographic) minimum. In case of a mixed strategy r`(`s_1`:`s_2`)`, the strategy s_1 is applied with probability r, and s_2 is used in the remaining cases.

As selection measures have considerable impact, many different strategies for automated reasoning tools have been reported in the literature. For instance, Vampire [**?**] employs a size/age ratio when deciding on a fact to be processed next. If this ratio is (e.g.) $2 : 3$ 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 mkbTT this approach can be achieved with the strategy

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

where the parameter $r \in [0, 1]$ controls the ratio of size-determined selections, e.g., a size/age ratio of $2 : 3$ corresponds to $r = 0.6$.

The "best-first" selection approach applied in Slothrop [?] corresponds to advancing a process for which $|E[N,p]| + |\mathrm{CP}(R[N,p])| + |C[N,p]|$ is minimal. When combined with, for example, a size/age ratio, this is expressed as follows:

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

In mkbTT, the strategies $s_{\texttt{max}}$ and $s_{\texttt{sum}}$ turned out to be beneficial. These strategies first restrict attention to processes where the number of symbols in $E[N,p]$ and $C[N,p]$ is minimal, then select nodes with minimal data and finally go 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 $s_{\texttt{max}}$ differs from $s_{\texttt{sum}}$ only in that `ssum` is replaced by `smax`. To use mkbTT with other heuristics than those just described, a user-defined strategy can be specified via a command line option.

6.5 Command Line Interface

The tool mkbTT is equipped with a simple command line interface. It expects an input problem in TPTP3 [?] or TPDB format, where in the latter case both the old textual and the newer XML format[3] are supported.

A number of options allow to configure the tool. Both a global and a local timeout in seconds can be specified using `-t` and `-T`. The termination prover is given as argument to the `-tp` option. Alternatively, if TT_2 is used internally a termination strategy can be supplied with the `-s` option. A selection strategy can be given with the option `-ss`, using the grammar described in Section 6.4.

The critical pair criteria BCP, PCP, CCP and MCP can be applied by supplying `-cp` with arguments `blocked`, `prime`, `connected`, and `all`, respectively. Isomorphism checks are to be specified via the option `-is` with optional arguments `rename`, `rename+`, `perm`, or `perm+`. With the suffix `+` we compare processes pairs in every iteration, otherwise checks are only performed when a process is split.

By default mkbTT applies a heuristic to determine which isomorphism is potentially applicable. Term indexing techniques used for rewriting and unification may be selected with the options `-ix` and `-ui` together with one of `nv`, `pi`, `dt`, or—in the case of rewriting—`ct`, referring to naive search, path indexing, discrimination trees, and code trees respectively.

The option `-kp` expects a floating point value larger than 1 and allows to give a process filtering rate. For example, `mkbtt -kp 1.2` deletes all processes that exceed the cost of the best process by 20%.

To control output, the flags `-ct`, `-st` and `-p` require mkbTT to print the completed system, statistics and a proof in case of success. Furthermore, the tool offers a checking mode where a file containing a rewrite system supplied via the option `-ch` is tested for termination, confluence and for allowing rewrite proofs for the the input equalities.

As an example, the call

```
mkbtt -t 600 -T 5 -tp aprove -cp prime WSW06_CGE2.trs
```

runs the tool on CGE_2 for at most 600 seconds using a script calling AProVE [?] for termination checks with a timeout of 5 seconds, and employs the critical pair criterion PCP.

[3] `http://www.termination-portal.org/`

6.6 Web Interface

Besides a command line interface, mkbTT can also be executed via a web interface. The screenshot in Figure 9 provides an impression. Various options may be configured by the user. A global timeout and a timeout for each termination check can be specified. Termination may be either checked inside mkbTT using $\mathsf{T_TT_2}$ functions or by applying an external tool. If $\mathsf{T_TT_2}$ is used internally, different predefined termination strategies can be selected. This includes basic reduction orders as well as predefined strategies combining dependency pairs, a dependency graph approximation, the subterm criterion, and reduction pairs, which proved to be powerful and fast. In addition, also a user-defined strategy may be supplied in the strategy language of $\mathsf{T_TT_2}$. Alternatively, termination checks can be performed externally with AProVE or MuTerm [**?**]. For the retrieval of encompassments and variants, one of the implemented term indexing techniques can be selected (path indexing, discrimination trees, code trees or naive search in the node set). To restrict the deduced equational consequences, one of the implemented critical pair criteria PCP, BCP, CCP or MCP can be selected. In case of CCP, the weak connectivity test as described in Section 2 is performed. Users can control isomorphism checks by selecting symbol renamings or term permutations, and specify whether these checks are performed repeatedly or only when processes are split.

7 Experimental Results

We ran experiments on a server equipped with eight dual-core AMD Opteron® processors 885 running at a clock rate of 2.6GHz with 64GB of main memory. Our test set comprises 101 problems collected from various papers. In the following paragraphs we summarize results obtained for the whole test set, and illustrate our conclusions with selected examples from that database. For this purpose, systems with prefix TPTP refer to theories underlying unit equality problems in TPTP 3.6.0 [**?**], prefix SK90 refers to [**?**, Section 3], WSW06 refers to the Slothrop [**?**] distribution, and BGK94 and C89 indicate systems stemming from [**?**] and [**?**], respectively. The whole testbench as well as the full experimental data can be obtained from the website. All experiments described in the following tables featured a timeout of 600 seconds. If a successful completion could not be achieved within that period this is marked by ∞, whereas $\perp$ indicates failure. If not stated otherwise, in all of the following experiments the following default settings of mkbTT were used: We interface $\mathsf{T_TT_2}$ internally with termination strategy `dp-lpo-kbo` and a termination timeout of two seconds, apply selection strategy $s_{\texttt{max}}$, and use the critical pair criterion MCP. We use only renaming isomorphisms, controlled by the `auto` heuristic. As term indexing techniques code trees and discrimination trees allow to retrieve encompassments and unifiable terms, respectively.

Table 1 compares mkbTT with Slothrop, listing the time required for a successful completion in seconds. The last two lines give the number of successes and the average time required to compute a convergent system, respectively. Overall, mkbTT solves about 15% more problems than Slothrop, and achieves completion on average about three times faster. Nevertheless, due to different selection strategies which are favourable for different problems there are also examples where Slothrop can produce a convergent system but our tool (with its default strategy) cannot, such as the system BGK94-M_8.

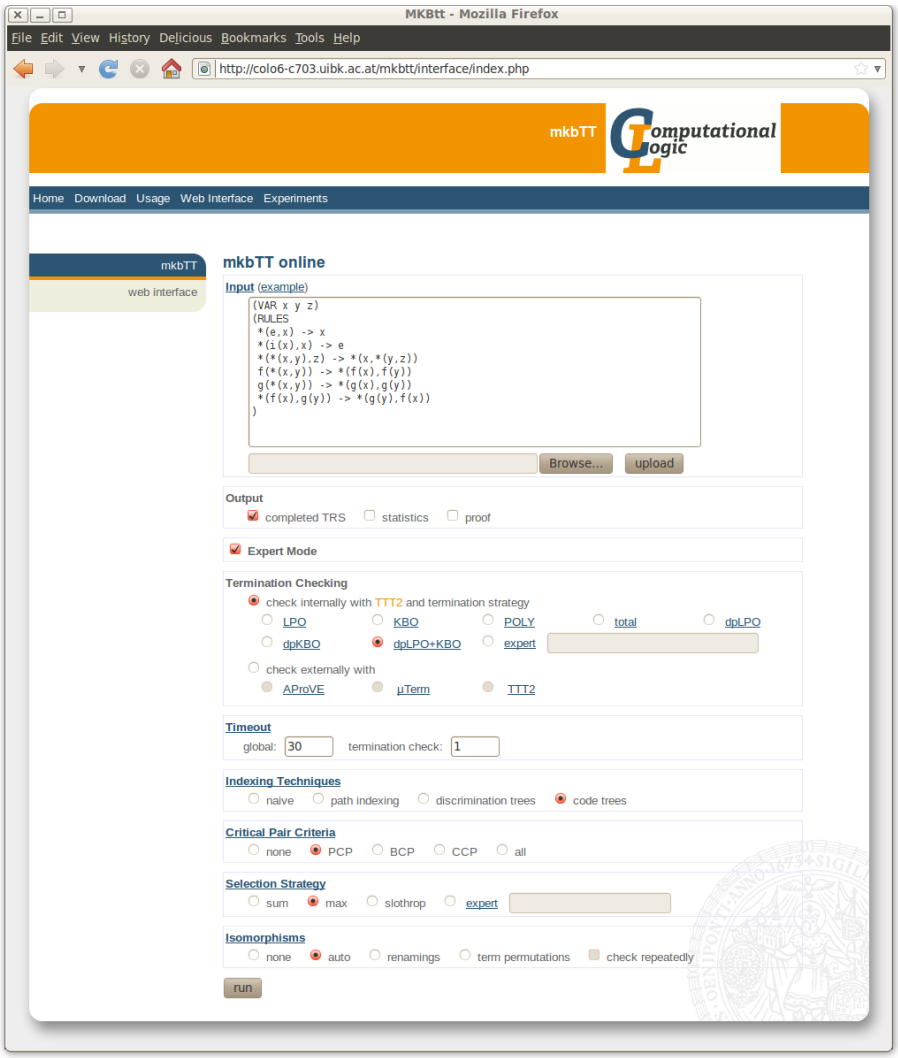

Fig. 9 Web interface of `mkbTT`.

Termination

Table 2 shows results obtained with different termination strategies when interfacing $\mathsf{T_TT_2}$ internally. The strategies `dp-kbo`, `dp-lpo`, and `dp-lpo-kbo` combine dependency pairs, a dependency graph approximation, the subterm criterion and some simple counting techniques with reduction pair processors using KBO, LPO, and both, respectively. The first three strategies apply plain LPO, KBO (with weights of two bits) and linear polynomial interpretations (with coefficients of two bits). Columns (1) show the time required for completion and columns (2) the percentage of time spent on termination.

	mkbTT	Slothrop
BGK94-D_8	75.9	∞
BGK94-M_8	∞	12.1
SK90-3.27	30.4	446.2
TPTP-GRP496-1	63.6	281.1
WS06-proofreduct	183.4	∞
WSW06-CGE_2	6.7	460.4
WSW06-CGE_3	45.2	∞
successes	80	67
average time	18.3	65.8

Table 1 Comparing mkbTT with Slothrop.

	`lpo`		`kbo`		`poly`		`dp-lpo`		`dp-kbo`		`dp-lpo-kbo`	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
C89-A_3	∞		234.0	17	∞		65.0	48	71.8	50	74.8	51
SK90-3.04	0.7	52	∞		∞		2.6	53	∞		2.7	54
SK90-3.27	5.6	21	23.4	7	446.8	31	34.8	29	27.9	21	29.7	27
TPTP-GRP493-1	∞		37.5	15	∞		∞		92.4	29	93.1	32
TPTP-GRP496-1	66.0	15	60.0	19	∞		∞		60.0	32	63.4	37
WS06-proofreduct	∞		∞		∞		174.9	93	179.5	92	182.4	92
WSW06-CGE_3	∞		∞		∞		43.7	80	42.5	80	44.3	81
successes	69		67		41		76		78		80	
average time	13.8		19.1		20.7		13.2		17.5		18.1	
termination %	18		23		60		55		49		51	

Table 2 Different termination strategies.

The use of plain reduction orders such as LPO or KBO often results in comparatively fast completions (as e.g. in the cases of SK90-3.04 and SK90-3.27 for LPO and TPTP-GRP493-1 for KBO) because little time is spent on termination checks as can be seen from the bottom line. On the other hand, plain reduction orders have comparatively limited power when it comes to orienting equations, which can prevent success as in case of WS06-proofreduct or the CGE systems. Overall, this results in fewer completions than obtained with more complex strategies that offer a higher flexibility. We could not find a system where polynomial interpretations are beneficial, and the overall success rate is considerably worse. The overall success rate turned out to be best with `dp-lpo-kbo`, supposedly since a combined strategy can cope best with problems where LPO is beneficial and problems where KBO is preferable. There are systems that can be completed with a plain reduction order but not with a strictly more powerful termination strategy employing dependency pairs, like TPTP-GRP496-1 using KBO. This illustrates that more termination power does not necessarily result in a higher chance for success because many possibilities for orienting equations also induce many processes, which can deteriorate performance significantly.

Selection Strategies

Table 3 demonstrates the crucial impact of selection strategies in mkbTT. Columns (1) give the time required for completion and columns (2) the number of control loop

	$s_{\texttt{max}}$		$s_{\texttt{sum}}$		$s_{\texttt{slothrop}}$		$s_{\texttt{size/age}}$		$s_{\texttt{old}}$	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
BGK94-D_8	195.0	112	∞		370.4	161	352.2	352	158.8	149
BGK94-D_{16}	45.2	102	117.8	132	∞		∞		152.1	151
BGK94-M_8	∞		14.6	22	∞		∞		∞	
C89-A_2	28.1	108	165.6	270	∞		∞		138.8	109
SK90-3.07	86.2	161	∞		∞		∞		∞	
SK90-3.22	∞		∞		∞		25.0	99	∞	
TPTP-GRP490-1	130.9	218	∞		564.0	182	∞		∞	
WSW06-CGE_3	46.8	49	439.1	325	232.0	87	138.2	136	132.0	85
successes	79		68		68		71		68	
average time	22.6		31.2		61.8		52.8		35.7	

Table 3 Different selection strategies.

iterations (i.e., selected nodes). In line with previous observations from the theorem proving literature, we found that the selection strategy is critical for the success of a run. While for some systems such as SK90-3.07, TPTP-GRP490-1 or the CGE examples it is beneficial to use $s_{\texttt{max}}$, there are also systems like BGK94-M_8 which can only be solved using $s_{\texttt{sum}}$, and problems like SK90-3.22 for which a mere size/age ratio works best. It thus seems impossible to determine a single best strategy. Since overall $s_{\texttt{max}}$ could complete most systems and is fastest on average, it is used by default in a specialized and faster implementation.

Critical Pair Criteria

Table 4 compares results obtained with mkbTT using the primality criterion PCP, the connectedness criterion CCP and the mixed criterion MCP. Columns (1) list the time required for completion, columns (2) the number of redundant critical pairs for the successful process and columns (3) the total number of created nodes.

We found a single system, TPTP-GRP490-1, which could only be completed when using PCP, CCP or MCP. For a number of systems the use of critical pair criteria results in a considerably smaller number of nodes and consequently some speedup, as in the cases of C89-A_3, TPTP-GRP457-1 or TPTP-GRP496-1. This is also reflected in the reduced average time for completion with CCP and MCP. However, there are also examples such as BGK94-D_{12} which can no longer be completed when using PCP (although CCP and MCP work), and examples such as TPTP-GRP484-1 where a less powerful criterion results in less control loop iterations and thus a shorter completion time. In these cases the selection strategy $s_{\texttt{max}}$ seems to be influenced by the critical pair criterion in an unfortunate way: the effect of critical pair criteria for a certain system was generally found to depend on the selection strategy. When comparing the three criteria, it turns out that PCP detects the least number of critical pairs, but performs redundancy checks very fast (see the bottom line). When summing up all critical pairs filtered out for successful processes, CCP is twice as effective as PCP. The criterion BCP is a little less effective than PCP, relevant results can be obtained from the website. Overall MCP turned out to be most beneficial.

	none		PCP			CCP			MCP		
	(1)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
BGK94-D_{12}	78.4	4884		∞		23.6	24	1946	23.1	38	1877
C89-A_3	96.3	2168	73.7	24	2098	93.5	24	2045	76.4	54	2009
TPTP-GRP457-1	7.9	777	2.8	17	460	4.8	29	590	4.0	33	512
TPTP-GRP484-1	343.6	15874	53.0	35	4225	111.2	76	5815	105.1	80	5571
TPTP-GRP490-1	∞		130.8	67	5371	80.0	155	3683	90.4	174	3600
TPTP-GRP496-1	80.3	5444	77.4	60	4702	65.3	100	3998	64.8	104	3927
WSW06-CGE_3	44.0	2240	45.2	14	2217	44.8	29	2141	44.8	29	2129
successes	79		78			80			80		
average time	21.6		23.5			18.7			18.4		
redundant CPs			834			1529			1605		
time to check			5.1			32.7			28.4		

Table 4 Different critical pair criteria.

Term Indexing

Table 5 compares the term indexing techniques implemented in mkbTT to retrieve variants and encompassments. Here `nv` abbreviates naive filtering of the node database, `pi` refers to path indexing, `dt` refers to discrimination trees and `ct` to code trees. Columns (1) list the time required for completion while columns (2) and (3) give the percentage of time spent on retrieval and rewrite operations, respectively. While all indexing techniques allow to complete the same number of systems, the time consumed by retrieval operations can be reduced significantly when using discrimination trees or code trees. Table 5 singles out some examples where the gain is especially significant. When comparing the time required for rewrite steps, discrimination trees fall back behind code trees since the retrieved candidate nodes still have to be checked for subsuming the query term. This is not required when using a technique achieving perfect filtering such as code trees.

The bottom lines sum up retrieval times over the whole database and show that although there is little difference concerning variant retrieval (which is negligible anyway) the time for encompassment retrieval can be reduced by more than 90% using discrimination trees or code trees. As expected, maintenance operations such as insertion and removal consume hardly any time.

Concerning the retrieval of unifiable terms in deduce operations, the use of term indexing techniques turned out to be less influential. Compared to naive filtering, discrimination trees decrease the average share of time spent on retrieval from 1% to 0.3%.

Isomorphisms

Isomorphism checks can be performed either only on process splits, or repeatedly for every process pair. Table 6 compares both possibilities for renamings (`ren`) and argument permutations (`ap`) with the setting where no isomorphism checks are used (a `+` indicates repeated checks). The setting `auto` refers to mkbTT's default strategy, which determines at the beginning of a completion run whether the initial equations allow for a nontrivial renaming or argument permutation. In this case repeated checks are per-

	nv			pi			dt			ct		
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
BGK94-D_8	173.9	6	19	180.8	3	21	160.1	1	15	151.8	<1	13
C89-A_2	29.3	6	17	29.6	4	16	27.5	1	12	25.5	<1	11
SK90-3.07	48.6	13	33	50.8	6	33	44.4	2	26	40.8	1	24
TPTP-GRP481-1	40.7	12	36	42.0	7	42	34.8	2	33	32.9	1	28
successes	79			79			79			79		
average time	21.1			21.1			19.0			17.9		
time/variants	4.2			3.9			3.8			4.1		
time/encompassment	104.8			51.5			11.5			6.8		
time/maintenance	0.8			1.2			0.8			0.5		

Table 5 Different term indexing techniques.

	none		ren		ren+		ap		ap+		auto	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
C89-A_3	64.8	174	65.6	174	77.1	161	65.4	174	86.4	174	77.4	161
SK90-3.02	0.1	7	0.1	7	0.1	7	0.1	3	0.1	3	0.1	7
SK90-3.28	101.7	499	102.8	499	157.6	385	102.4	499	116.2	499	156.3	385
SK90-3.29	3.5	128	2.6	97	3.0	73	3.5	128	4.5	128	3.0	73
TPTP-GRP011-4	3.0	13	1.8	6	1.8	6	3.0	13	3.1	13	1.8	6
WSW06-CGE_2	37.7	50	37.6	50	7.0	11	37.5	50	48.7	50	7.0	11
WSW06-CGE_3	175.1	163	176.6	163	46.0	35	176.8	163	204.9	163	46.0	35
successes	80		80		80		80		80		80	
average time	18.3		18.3		23.1		18.3		27.3		17.8	

Table 6 Different isomorphisms.

formed throughout the deduction. Columns (1) give the time required for completion in seconds, and columns (2) the number of processes emerging in the course of a run.

Renaming checks, especially when performed repeatedly, turned out to be useful for a number of problems, in particular the CGE systems. Also for some string systems like SK90-3.28 and SK90-3.29, and TPTP-GRP011-4 the number of processes could be reduced, although this does not always result in faster completions due to the time required for checking. Argument permutations were useful for just two small systems in the benchmark set, one of which is SK90-3.02 where the number of processes could be halved.

On the other hand, especially repeated checks for isomorphisms can be costly if no isomorphic process pairs appear. This is for example the case for SK90-3.28 and the CGE systems when used with argument permutations. Overall the `auto` setting prevails concerning number of successes and average time, although the heuristic does not always go for the best choice.

Novel Completions

While Slothrop was the first completion tool to handle CGE_2 (in more than 200 seconds), mkbTT can also complete the systems CGE_3, CGE_4 and CGE_5 describing the theory of 3, 4 and 5 commuting group endomorphisms within 18, 145 and 35796 seconds,

	mkbTT			Maxcomp	
	standard	LPO	KBO	LPO	KBO
BGK94-D_8	76.0	24.3	238.8	0.5	∞
C89-A_3	74.8	∞	234.0	2.3	407.6
OKW95-dt1	2.1	1.5	2.7	40.8	∞
SK90-3.22	∞	∞	∞	3.2	5.7
WS06-proofreduct	182.4	$\perp$	∞	$\perp$	∞
WSW06-CGE_2	6.7	∞	∞	∞	∞
WSW06-equiv-proofs	5.5	∞	∞	∞	∞
successes	80	69	67	77	61
average time	18.1	13.8	19.1	3.6	8.6

Table 7 Comparison of mkbTT and Maxcomp.

respectively. Our tool also produced the first convergent TRS for the proof reduction system presented in [**?**].

8 Conclusion

The present paper reports on multi-completion with termination tools, a completion approach that combines the use of automatic termination provers with completion using multiple reduction orders. The resulting method offers a novel degree of automation as users do not have to supply a suitable order, and provides increased flexibility concerning the orientation of rules. We described the inference system, illustrated the approach by means of an example run, and formally proved its correctness. We also commented on new insights into the method's completeness. Critical pair criteria, a classical means to filter equational consequences in standard completion procedures, were carried over to the present setting. As further improvement isomorphisms were described, and shown to not compromise completeness. We gave a detailed account of the implementation of our approach in the tool mkbTT, reporting on its control flow and implementation issues such as term indexing techniques and selection strategies. An outline of both the web and command line interface provides insight into the tool's usage. We concluded with detailed experimental results which prove the power of multi-completion when comparing to Slothrop and show how the improvements allowed mkbTT to achieve new completions of challenging problems such as systems of the CGE family.

Very recently, the tool Maxcomp took a novel approach to completion by encoding the whole process as a satisfiability problem [**?**]. Although currently restricted to reduction orders like LPO, KBO and MPO where orientability can be encoded as a satisfaction problem, the resulting tool is fully automatic in that users do not need to supply a concrete order. Moreover, Maxcomp circumvents the choice of a concrete selection strategy, which is a critical parameter for both mkbTT and Slothrop. It can thus also never fail due to premature orientation, in contrast to mkbTT as illustrated by Example 1. This is reflected in the results presented in Table 7, where Maxcomp solves more problems than our tool when restricting the termination strategy to LPO.

	mkbTT	Maxcomp with LPO	Maxcomp with KBO
successes	1109	821	812

Table 8 Comparison of mkbTT and Maxcomp on a subset of TPDB.

Even with its more complex standard termination strategy, mkbTT can complete only a few more problems, although Maxcomp of course fails on challenging problems like WSW06-CGE$_2$ and WS06-proofreduct which cannot be completed using plain LPO or KBO. However, the difference grows when a benchmark set requiring more sophisticated termination techniques is considered, as shown in Table 8. Here the 3061 problems in TPDB version 7 were considered which are not already confluent and could be completed by at least one of the tools within 600 seconds.

The approach underlying mkbTT has been extended to *ordered* completion [**?**] although a ground-confluent system is in this setting only obtained when restricting to termination techniques that entail total termination. Completion modulo theories also benefits from a multi-completion approach. Termination tools that support termination analysis modulo theories can be used here. For the important AC case, this approach is implemented in the tool mascott [**?**]. It is to be investigated whether termination tools can be used in other variants of completion, such as normalized completion [**?**]. The extension to other calculi of automated reasoning which classically depend on reduction orders, for example paramodulation [**?**], is subject to future work, too.

Acknowledgments.

We thank the anonymous reviewers for helpful comments.