KBCV–Knuth-Bendix Completion Visualizer[⋆]

Thomas Sternagel[1] and Harald Zankl[2]

[1] Master Program in Computer Science, University of Innsbruck, Austria
[2] Institute of Computer Science, University of Innsbruck, Austria

Abstract This paper describes a tool for Knuth-Bendix completion. In its interactive mode the user only has to select the orientation of equations into rewrite rules; all other computations (including necessary termination checks) are performed internally. Apart from the interactive mode, the tool also provides a fully automatic mode. Moreover, the generation of (dis)proofs in equational logic is supported. Finally, the tool outputs proofs in a certifiable format.

Keywords: term rewriting, completion, equational logic, automation.

1 Introduction

The *Knuth-Bendix Completion Visualizer* (KBCV) is an interactive/automatic tool for Knuth-Bendix completion and equational logic proofs. This paper describes KBCV version 1.7, which features a command-line and a graphical user interface as well as a Java-applet version. The tool is available under the *GNU Lesser General Public License 3* at

`http://cl-informatik.uibk.ac.at/software/kbcv`

Completion is a procedure which takes as input a finite set of equations E (and nowadays optionally a reduction order $>$) and attempts to construct a terminating and confluent term rewrite system (TRS) R which is equivalent to E, i.e., their equational theories coincide. In case the completion procedure succeeds, R represents a decision procedure for the word problem of E. Now two terms are equivalent with respect to E if and only if they reduce to the same normal form with respect to R.

The computation is done by generating a finite sequence of intermediate TRSs which constitute approximations of the equational theory of E. Following Bachmair and Dershowitz [2] the completion procedure can be modeled as an inference system like system $\mathcal{C}$ in Figure 1. The inference rules work on pairs (E, R) where E is a finite set of equations and R is a finite set of rewrite rules. The goal is to transform an initial pair $(E, \varnothing)$ into a pair $(\varnothing, R)$ such that R is terminating, confluent and equivalent to E. In our setting a completion procedure based on these rules may succeed (find R after finitely many steps), loop, or fail. In Figure 1 a reduction order $>$ is provided as part of the input. We use $s \overset{\sqsupset}{\to}_R u$

⋆ This research is supported by FWF P22467 and a grant of the Hypo Tirol Bank.

$$
\begin{array}{llll}
\textsc{Deduce} \ \dfrac{(E,R)}{(E \cup \{s \approx t\}, R)} & \text{if } s \ {}_R{\leftarrow} \ u \to_R t & \textsc{Orient} \ \dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E, R \cup \{s \to t\})} & \text{if } s > t \\[2ex]
\textsc{Compose} \ \dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})} & \text{if } t \to_R u & \textsc{Delete} \ \dfrac{(E \cup \{s \approx s\}, R)}{(E,R)} & \\[2ex]
\textsc{Collapse} \ \dfrac{(E, R \cup \{s \to t\})}{(E \cup \{u = t\}, R)} & \text{if } s \xrightarrow{\sqsupset}_R u & \textsc{Simplify} \ \dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R)} & \text{if } s \to_R u
\end{array}
$$

Figure 1: Inference rules for completion with a fixed reduction order ($\mathcal{C}$).

to express that s is reduced by a rule $\ell \to r \in R$ such that ℓ cannot be reduced by another rule $s \to t \in R$. The notation $s \mathrel{\dot{\approx}} t$ denotes either of $s \approx t$ and $t \approx s$.

Writing $(E, R) \vdash_{\mathcal{C}} (E', R')$ to indicate that (E', R') is obtained from (E, R) by one of the inference rules of system $\mathcal{C}$ we define a *completion procedure*:

Definition 1. *A* completion procedure *is a program that accepts as input a finite set of equations E_0 (together with a reduction order $>$) and uses the inference rules of Figure 1 to construct a sequence*

$$(E_0, \varnothing) \vdash_{\mathcal{C}} (E_1, R_1) \vdash_{\mathcal{C}} (E_2, R_2) \vdash_{\mathcal{C}} (E_3, R_3) \vdash_{\mathcal{C}} \cdots$$

Such a sequence is called a run *of the completion procedure on input E_0 and $>$. A finite run $(E_0, \varnothing) \vdash_{\mathcal{C}}^{n} (\varnothing, R_n)$ is* successful *if R_n is locally confluent.*

The following result follows from [1, Theorem 7.2.8] specialized to finite runs.

Lemma 2. *Let $(E_0, \varnothing) \vdash_{\mathcal{C}}^{n} (\varnothing, R_n)$ be a successful run of completion. Then R_n is terminating, confluent, and equivalent to E_0.* □

In the sequel we assume familiarity with term rewriting, equational logic, and completion [1]. The remainder of this paper is organized as follows. In the next section the main features of KBCV are presented before Section 3 addresses implementation issues and experimental results. Section 4 concludes.

2 Features

KBCV offers two modes for completion, namely the Normal Mode (Section 2.1) and the Expert Mode (Section 2.2). In the GUI the user can change the mode via the menu entry *View* at any time. Irregardless of the chosen mode, termination checks are performed automatically, following the recent approach from [11]. By default, an incremental LPO is constructed and maintained by the tool but also external termination tools are supported (this option is not available in the applet version). For convenience KBCV stores a history that allows to step backwards (and forwards again) in interactive completion proofs. Apart from completion proofs, the tool can generate proofs in equational logic (Section 2.3) and produces output in a certifiable format.

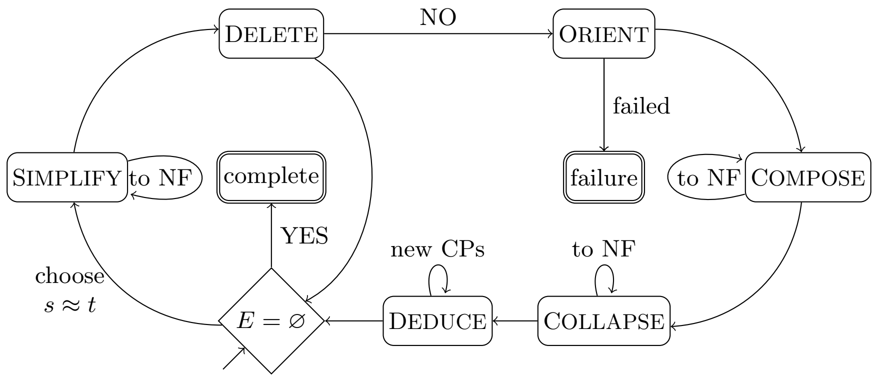

Figure 2: Flow chart for the efficient completion procedure.

2.1 Normal Mode

In *normal mode* the user can switch between *efficient* and *simple* completion. The efficient procedure executes all inference rules from Figure 1 in a fixed order, while the simple procedure considers a subset only.

Efficient Completion The *efficient completion procedure* (following Huet [4], see Figure 2) takes a set of equations E as input and has three possible outcomes: It may terminate successfully, it may loop indefinitely, or it may fail because an equation could not be oriented into a rewrite rule.

While $E \neq \varnothing$ the user chooses an equation $s \approx t$ from E. The terms in this equation are simplified to normal form by using SIMPLIFY exhaustively. In the next step the equation is deleted if it was trivial and if so the next iteration of the loop starts. Otherwise (following the transition labeled NO) the user suggests the orientation of the equation into a rule and ORIENT performs the necessary termination check. Here the procedure might fail if the equation cannot be oriented (in either direction) with the used termination technique. But if the orientation succeeds the inferred rule is used to reduce the right-hand sides of (other) rules to normal form (COMPOSE) while COLLAPSE rewrites the left-hand sides of rules, which transforms rules into equations that go back to E. In this way the set of rules in R is kept as small as possible at all times. Afterwards DEDUCE is used to compute (all) critical pairs (between the new rule and the old rules and between the new rule and itself). If still $E \neq \varnothing$ the next iteration of the loop begins and otherwise the procedure terminates successfully yielding the terminating and confluent (complete) TRS R equivalent to the input system E.

Simple Completion The simple procedure (following the *basic* completion procedure [1, Figure 7.1]) makes no use of COMPOSE and COLLAPSE, which means that the inference rule DEDUCE immediately follows ORIENT. Hence although correct, this procedure is not particularly efficient.

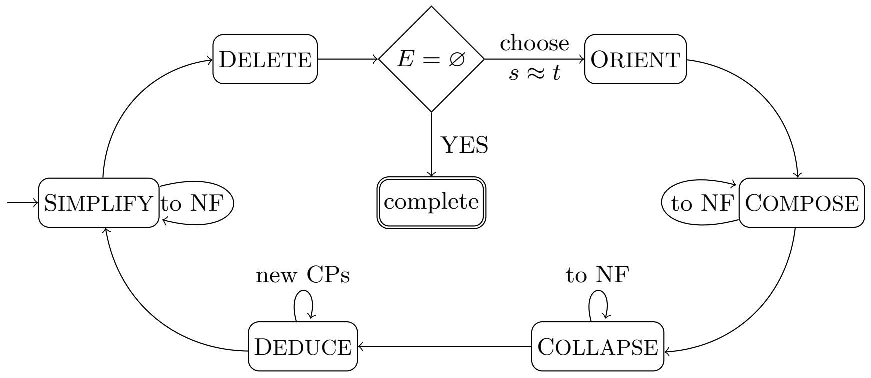

Figure 3: Flow chart for the automatic mode.

2.2 Expert Mode

Inference System In the *expert mode* the user can select the equations and rewrite rules on which the desired inference rules from Figure 1 should be applied on. If no equations/rules are selected explicitly then all equations/rules are considered. For efficiency reasons DEDUCE does only add critical pairs emerging from overlaps that have not yet been considered. KBCV notifies the user if a complete R equivalent to the input E is obtained.

Automatic Mode At any stage of the process the user can press the button $\boxed{\textit{Completion}}$ which triggers the automatic mode of KBCV where it applies the inference rules according to the loop in Figure 3. Pressing the button again (during the completion attempt) stops the automatic mode and shows the current state (of the selected thread, see below). It is also possible to specify an upper limit on the loops performed in Figure 3 (*Settings* → *Automatic Completion*). This is especially useful to step through a completion proof with limit 1.

In Figure 3 the rules SIMPLIFY and DELETE operate on all equations and are applied exhaustively. If $E = \varnothing$ then R is locally confluent (since the previous DEDUCE considered all remaining critical pairs) and the procedure successfully terminates. Note that in contrast to the completion procedure from Figure 2 the automatic mode postpones the choice of the equation $s \approx t$. Hence KBCV can choose an equation of minimal length *after* simplification (which is typically beneficial for the course of completion) for the rule ORIENT. To maximize power, KBCV executes two threads in parallel which have different behavior for ORIENT. The first thread prefers to orient equations from left-to-right and if this is not possible it tries a right-to-left orientation (the second thread behaves dually). If this also fails another equation is selected in the next turn. (Note that it is possible that some later equation can be oriented which then simplifies the problematic equation such that it can be oriented or deleted.) A thread fails if no equation in E can be oriented in the ORIENT step.

2.3 Equational Logic and Certification

Since KBCV stores how rules have been deduced from equations [9], in command-line mode the command `showh` lists how rules/equations have been derived and allows to trace back the completion steps that gave rise to a rule/equation. The same mechanism facilitates KBCV to automatically transform a join $s \rightarrow_R^* \cdot {}_R^*\!\!\leftarrow t$ with respect to the current system R (which need not be complete yet) into a conversion with respect to the input system E, i.e., $s \leftrightarrow_E^* t$, and further into equational proofs with respect to E (*File* → *Equational Proof*).

If E could be completed into a TRS R, the recent work in [9] allows KBCV to export proof certificates (*File* → *Export Equational Proof* and *File* → *Export Completion Proof*) in CPF, a certification proof format for rewriting.[3] These proof certificates can be certified by CeTA [10], i.e., checked by a trustable program generated from the theorem prover Isabelle. Apart from the input system E and the completed TRS R such certificate must also contain a proof that E and R are equivalent, e.g., by giving an explicit conversion $\ell \leftrightarrow_E^* r$ for each $\ell \rightarrow r \in R$.

3 Implementation and Experiments

KBCV is implemented in Scala,[4] an object-functional programming language which compiles to Java Byte Code. For this reason KBCV is portable and runs on Windows and Linux machines. We have developed a term library in Scala (`scala-termlib`, available from KBCV's homepage) of approximately 1700 lines of code. KBCV builds upon this library and has an additional 4500 lines of code.

Besides the stand-alone version of KBCV there also is a Java-Applet version available online. The stand-alone version has three different modes: The text mode where one can interact with KBCV via the console, the graphic mode using a graphical user interface implemented in `java.swing`, and the hybrid mode where the text mode and the graphic mode are combined.

In text mode typing `help` yields a list of all available commands, whereas in graphic (hybrid) mode or the Java-Applet you can select *Help* → *User Manual* to get a description of the user interface.

The stand-alone version of KBCV is able to call third party termination checkers whereas the Java-Applet version is limited to the internal LPO for termination proofs. As input KBCV supports the XML-format for TRSs[5] and also a subset of the older TRS-format.[6] (Only one `VAR` and one `RULES` section are allowed in this order. No theory or strategy annotations are supported.) In both cases rules are interpreted as equations.

In addition KBCV supports another file format for the export and import of command logs to save and load user specific settings of KBCV. This format lists all executed commands within KBCV in a human readable form, like:

[3] `http://cl-informatik.uibk.ac.at/software/cpf`
[4] `http://www.scala-lang.org/`
[5] `http://www.termination-portal.org/wiki/XTC_Format_Specification`
[6] `http://www.lri.fr/~marche/tpdb/format.html`

	LPO			termination tool		
	KBCV	MKBTT	MAXCOMP	KBCV	MKBTT	Slothrop
completed	85	70	86	86	81	71
`LS94_P1`	✓			✓		
`SK90_3.26`	✓			✓		
`Slothrop_cge`					✓	
`Slothrop_equiv_proof_or`					✓	
`WS06_proofreduction`					✓	

Table 1: Experimental results on 115 systems.

```
load ../examples/gene.trs
orient > 1
simplify
...
```

Saving the current command log is done via (*File* → *Export Command Log*) and loading works alike (*File* → *Load Command Log*). Command logs saved in the file `.kbcvinit` are loaded automatically on program startup.

Although the major attraction of KBCV clearly is its interactive mode, in the sequel experimental results demonstrate that its automatic mode can compete with state-of-the-art completion tools. To this end we extend [5, Table 1] with data for KBCV (considering 115 problems from the distribution of MKBTT).[7] Hence Table 1[8] compares KBCV with MKBTT [8], MAXCOMP [5], and Slothrop [11]. Within a time limit of 300 seconds, KBCV completes 85 systems using its internal LPO and succeeds on an additional system when calling the external termination tool $\mathsf{T_TT_2}$ [6]. Slothrop [11] was the first tool to construct reduction orders on the fly using external termination tools and obtains 71 completed systems. MKBTT [8] adopts this approach, but additionally features *multi-completion*, i.e., considering multiple reduction orderings at the same time. Finally, the strategy of MAXCOMP [5] is to handle all suitable candidate TRSs (terminating and maximal) at once. MAXCOMP can complete 86 systems with LPO but since the search for maximal TRSs is coupled with the search for the reduction order this approach does not support external termination tools. All tools together can complete 95 systems. The lower part of Table 1 shows those systems which only one tool could complete within the given time limit. Here KBCV completed two systems where all other tools failed.

All 86 completion proofs found by KBCV (Table 1) could be certified by CeTA [10] (see Section 2.3). Since recently, MKBTT can also provide proof certificates but currently neither MAXCOMP nor Slothrop support them.

[7] `http://cl-informatik.uibk.ac.at/software/mkbtt`

[8] KBCV data available from `http://cl-informatik.uibk.ac.at/software/kbcv/experiments/12ijcar`.

4 Conclusion

In this paper we have presented KBCV, a tool that supports interactive completion proofs. Hence it is of particular interest for students and users that are exposed to the area of completion for the first time or want to follow a completion proof step by step. Its automatic mode can compete with modern completion tools (Slothrop, MKBTT, MAXCOMP) that use more advanced techniques for completion (completion with external termination tools, multi-completion, maximal-completion) but lack an interactive mode. Since KBCV records how rules have been derived, it can produce certifiable output of completion proofs and can construct (dis)proofs in equational logic.

Unfailing completion [3] is a variant of Knuth-Bendix completion, which sacrifices confluence for ground confluence. One possible direction for future work would be to integrate unfailing completion into KBCV. Another issue is to gain further efficiency by a smart design of the employed data structure [7].

Acknowledgments We thank Christian Sternagel and the reviewers for helpful comments and suggestions concerning efficiency.

References

1. Baader, F., Nipkow, T.: Term Rewriting and *All That*. Cambridge University Press, New York (1999)
2. Bachmair, L., Dershowitz, N.: Equational inference, canonical proofs, and proof orderings. Journal of the ACM 41(2), 236–276 (1994)
3. Bachmair, L., Dershowitz, N., Plaisted, D.: Completion without failure. In: Resolution of Equations in Algebraic Structures, Vol. 2: Rewriting Techniques. 1–30(1989)
4. Huet, G.P.: A complete proof of correctness of the Knuth-Bendix completion algorithm. J. Comput. Syst. Sci. 23(1), 11–21 (1981)
5. Klein, D., Hirokawa, N.: Maximal completion. In: Schmidt-Schauß, M. (ed.) RTA. LIPIcs, vol. 10, pp. 71–80. Schloss Dagstuhl, Dagstuhl (2011)
6. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean Termination Tool 2. In: Treinen, R. (ed.) RTA. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009)
7. Lescanne, P.: Completion procedures as transition rules + control. In: Díaz, J., Orejas, F. (eds.) TAPSOFT. LNCS, vol. 351, pp. 28–41. Springer, Heidelberg (1989)
8. Sato, H., Winkler, S., Kurihara, M., Middeldorp, A.: Multi-completion with termination tools (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS (LNAI), vol. 5195, pp. 306–312. Springer, Heidelberg (2008)
9. Sternagel, T., Thiemann, R., Zankl, H., Sternagel, C.: Recording completion for finding and certifying proofs in equational logic. In: IWC. (2012)
10. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs. LNCS, vol. 5674, pp. 452–468. Springer, Heidelberg (2009)
11. Wehrman, I., Stump, A., Westbrook, E.: Slothrop: Knuth-Bendix completion with a modern termination checker. In: Pfenning, F. (ed.) RTA. LNCS, vol. 4098, pp. 287–296. Springer, Heidelberg (2006)