Certification of Complexity Proofs using CeTA

Martin Avanzini[1], Christian Sternagel[2], and René Thiemann[2]

1 **Università di Bologna & INRIA, Sophia Antipolis, Italy**
`martin.avanzini@uibk.ac.at`
2 **University of Innsbruck, Austria**
`{christian.sternagel|rene.thiemann}@uibk.ac.at`

Abstract

Nowadays certification is widely employed by automated termination tools for term rewriting, where certifiers support most available techniques. In complexity analysis, the situation is quite different. Although tools support certification in principle, current certifiers implement only the most basic technique, namely, suitably tamed versions of reduction orders. As a consequence, only a small fraction of the proofs generated by state-of-the-art complexity tools can be certified. To improve upon this situation, we formalized a framework for the certification of modular complexity proofs and incorporated it into CeTA. We report on this extension and present the newly supported techniques (match-bounds, weak dependency pairs, dependency tuples, usable rules, and usable replacement maps), resulting in a significant increase in the number of certifiable complexity proofs. During our work we detected conflicts in theoretical results as well as bugs in existing complexity tools.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases complexity analysis, certification, match-bounds, weak dependency pairs, dependency tuples, usable rules, usable replacement maps

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.x

1 Introduction

The last decade saw a wealth of techniques for automated termination tools, closely followed by techniques and tools for automated complexity analysis in recent years. In individual proofs, such tools often apply several techniques in combination, making human inspection ever more unrealistic, due to their sheer size. Moreover, the increasing power of automated tools comes at the cost of amplified complexity, reducing reliability; hence the interest in automatic certification of termination and complexity proofs.

Whereas our certifier CeTA [18] is already able to certify most proofs generated by current termination tools for term rewrite systems (TRSs), initial support for complexity proofs was added only recently [17]. In this paper we present a significant extension of CeTA towards the certification of complexity proofs. To this end, we formalized several techniques for complexity analysis within the proof assistant Isabelle/HOL [14] as part of our formal library IsaFoR.[1] On top of these general results, we augmented CeTA by corresponding functions, that check whether specific applications of techniques, encountered inside automatically generated complexity proofs, are correct.

As a result, the power of CeTA for certifying complexity proofs has almost tripled in comparison to last year [17], and more than 75 % of all tool-generated proofs can be certified.

[1] `http://cl-informatik.uibk.ac.at/software/ceta`

26th International Conference on Rewriting Techniques and Applications (RTA'15).
Editor: Maribel Fernández; pp. 1–17
lipics-logo-bw.pdf Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Moreover, via certification we detected and fixed several bugs in current complexity tools, some of which had remained undetected for more than five years.

Contribution and Overview. After giving some preliminaries in Section 2, we present our main contributions. In Section 3, we explain our formalization of a framework which admits us to certify composite complexity proofs. At this point, we also report on conflicting notions of basic complexity definitions in the literature. In Section 4 we describe our formalization of the match-bounds technique. Here, the transition from termination to complexity results is surprisingly easy. Concerning the integration of match-bounds for relative rewriting, we provide a new example showing that two existing variants are incomparable. In Section 5, we discuss our formalization of two dependency pair related techniques: weak dependency pairs and dependency tuples. We choose to conduct the respective proofs using two slightly different approaches—one focusing on contexts, the other on sets of positions—and comment on our findings. In Section 6, we slightly generalize one variant of usable rules, and also support another variant for innermost rewriting, for which we reuse existing proofs from termination analysis. Furthermore, we present a new theorem combining usable rules, usable replacement maps, and argument filters. Finally, in Section 7, we discuss conducted experiments and conclude.

All of the proofs that are presented (or omitted) in the following have been formalized and made available as part of IsaFoR. Direct links to the formalization are available from the following website, that also contains all details on our experiments.

`http://cl-informatik.uibk.ac.at/software/ceta/experiments/complexity/`

2 Preliminaries

We assume basic familiarity with term rewriting (as for example obtained by reading the textbook by Baader and Nipkow [3]) but shortly recall some basic notions and notations that are used later on.

By $\mathcal{T}(\mathcal{F},\mathcal{V})$ we denote the set of *(first-order) terms* w.r.t. a *signature* $\mathcal{F}$ and a set of *variables* $\mathcal{V}$, and by $\mathcal{T}(\mathcal{F})$ the set of ground terms. We write $\mathsf{root}(t)$ for the root symbol of a non-variable term t. The *size* $|t|$ of a term t is defined by $|x| = 1$ for $t = x \in \mathcal{V}$, and $|f(t_1,\ldots,t_n)| = 1 + \sum_{i=1}^{n} |t_i|$, otherwise. A *(multihole) context* is a term that may contain an arbitrary number of *holes*, represented by the special symbol $\Box$. Replacing the holes in a given multihole context C by terms $t_1,\ldots,t_n$ is written $C[t_1,\ldots,t_n]$. (At this point it might be worth mentioning that in our formalization we have to make sure that the number of holes in C corresponds to the number of terms n. For simplicity's sake we do not make this explicit in the remainder). Whenever $s = C[t]$ for some context C ($\neq \Box$), then t is called a *(proper) subterm* of s. We write $t\sigma$ for the application of a substitution σ to a term t.

A *TRS* $\mathcal{R}$ is a set of *(rewrite) rules*, where a rule $\ell \to r$ is a pair of terms such that $\ell \notin \mathcal{V}$ and only variables already occurring in ℓ are allowed in r. The *defined symbols* of $\mathcal{R}$, written $\mathcal{D}(\mathcal{R})$, are those that are roots of left-hand sides of its rules. We use $\mathcal{F}\mathsf{un}(\cdot)$ to denote the set of function symbols occurring in a given term, context, or TRS. A TRS is *left-linear* (*non-duplicating*) if and only if for all rules $\ell \to r \in \mathcal{R}$, no variable occurs more than once in ℓ (more often in r than in ℓ).

The standard way of uniquely referring to subterms is via *positions*, denoted by lists of natural numbers. The subterm of a term t at position p is written $t|_p$. We use $\leq$ for the usual partial order on positions, and denote by $p \mid\mid q$ that positions p and q are *parallel*, i.e., incomparable by $\leq$. The strict part of $\leq$ is denoted by $<$.

There is a *rewrite step* from term s to term t w.r.t. the rewrite relation induced by TRS $\mathcal{R}$, denoted $s \to_{\mathcal{R}} t$, whenever there are C, σ, and $\ell \to r \in \mathcal{R}$ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. Equivalently, we say that s *rewrites* to t at position p, where p is the unique position of $\Box$ in C. The subterm $\ell\sigma$ above is called an *($\mathcal{R}$-)redex*. Terms not containing any $\mathcal{R}$-redexes are called $\mathcal{R}$-normal or *normal forms*, and we write $\mathsf{NF}(\mathcal{R})$ for the set of all $\mathcal{R}$-normal forms. We sometimes use the same notion not only for TRSs but also for sets of terms, since right-hand sides of rules are irrelevant for the existence of redexes anyway.

For termination analysis *Q-restricted rewriting* (named after the additional parameter, a set of terms, which is usually denoted $\mathcal{Q}$) was introduced in order to cover full rewriting and innermost rewriting (as well as variations that lie somewhere in between) under a single framework [9]. Here, a rewrite step $C[\ell\sigma] \xrightarrow{\mathcal{Q}}_{\mathcal{R}} C[r\sigma]$ is a standard rewrite step $C[\ell\sigma] \to_{\mathcal{R}} C[r\sigma]$ whose redex $\ell\sigma$ additionally satisfies the condition that all its proper subterms are $\mathcal{Q}$-normal, i.e., do not match any term in $\mathcal{Q}$ (in that way standard rewriting is Q-restricted rewriting with empty $\mathcal{Q}$ and for innermost rewriting we take the left-hand sides of $\mathcal{R}$ as $\mathcal{Q}$). This proves convenient also for complexity analysis and its notions of runtime complexity and innermost runtime complexity. Additionally, *relative rewriting* is important for complexity analysis, since it can be employed to obtain modular proofs. A *relative rewrite system* consists of two TRSs $\mathcal{S}$ and $\mathcal{W}$, and is denoted by $\mathcal{S}/\mathcal{W}$. The corresponding *relative rewrite relation*, written $\to_{\mathcal{S}/\mathcal{W}}$, is given by $\to_{\mathcal{W}}^* \cdot \to_{\mathcal{S}} \cdot \to_{\mathcal{W}}^*$. Combined this leads to *Q-restricted relative rewriting*, where $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ denotes the relation $\xrightarrow{\mathcal{Q}}_{\mathcal{W}}^* \cdot \xrightarrow{\mathcal{Q}}_{\mathcal{S}} \cdot \xrightarrow{\mathcal{Q}}_{\mathcal{W}}^*$. Note that we fix the same $\mathcal{Q}$ for "strict" and "weak" steps, which is sufficient for our purposes.[2]

Given a binary relation $\to$ and a set A, we define $\to(A) = \{b \mid \exists a \in A.\, a \to b\}$.

3 A Framework for Modular Complexity Proofs

In complexity analysis of TRSs we are usually interested in the maximal number of steps that are possible when starting from a given set of terms. To this end, the basic ingredient of our formalization is the *derivation bound* (defined in theory `Complexity`; see also [17]), where a function g constitutes a derivation bound of relation R w.r.t. *starting elements* from a family of sets S, written $\mathsf{db}_R^S(g)$, if and only if for every $n \in \mathbb{N}$ and $x \in S_n$, every sequence of R-steps starting at x is of length at most $g(n)$. The intuition is that S_n contains "objects" of size n. This, more or less, corresponds to the usual notion of complexity. To be more precise, Avanzini and Moser [2] define $\mathsf{cp}(n, T, R) = \mathsf{max}\{\mathsf{dh}(t, R) \mid \exists t \in T.\, |t| \leq n\}$, where dh denotes the *derivation height* of a term, and derivational complexity as well as runtime complexity are obtained by suitably instantiating T and R. However, as argued earlier [17], using the derivation bound g as argument avoids undefined situations that arise with the usual definition, e.g., taking the maximum of a potentially infinite set. Whenever $\mathsf{cp}(n, T, R)$ is defined, we have $\mathsf{db}_R^S(g)$ with $S_n = \{t \in T \mid |t| \leq n\}$ and $g(n) = \mathsf{cp}(n, T, R)$, as well as $h(n) \geq \mathsf{cp}(n, T, R)$ for all other derivation bounds h. That is, our bounds are not tight, but arbitrary upper bounds.

Depending on the set of starting elements, we obtain the usual notions of *derivational complexity* and *runtime complexity*, respectively. For the former we consider all terms of size n w.r.t. a given signature $\mathcal{F}$, whereas the latter is based on *basic terms* of size n. Given two sets of function symbols $\mathcal{D}$ (defined symbols) and $\mathcal{C}$ (constructors), and a set of variables $\mathcal{V}$, the set of basic terms $\mathsf{BT}(\mathcal{D}, \mathcal{C}, \mathcal{V})$ consists of those terms which are rooted by a symbol from

[2] A more general relation with separate $\mathcal{Q}$s for $\mathcal{S}$ and $\mathcal{W}$ would be imaginable. However, since tools do not support this variation, we stick to the simpler case.

$\mathcal{D}$ and where all arguments are terms of $\mathcal{T}(\mathcal{C},\mathcal{V})$. At this point we would like to mention that there are conflicting notions of basic terms: Hirokawa and Moser [10] and Noschinski et al. [15] use the above definition of basic terms. In contrast, Avanzini [1] additionally restricts basic terms to be ground, intending that constructor ground terms correspond to values, and thus, basic terms correspond to function application on input values. Since IsaFoR does not enforce basic terms to be ground, every (upper) derivation bound that is certified by CeTA also is valid w.r.t. Avanzini's notion of basic terms. However, there might be valid derivation bounds w.r.t. the ground semantics which cannot be certified in the non-ground setting:

▶ **Example 1.** Let $\mathcal{R} = \{\mathsf{f}(\mathsf{f}(x)) \to \mathsf{g}(x), \mathsf{g}(x) \to \mathsf{f}(\mathsf{f}(x)), \mathsf{f}(\mathsf{a}) \to \mathsf{a}\}$. Then there are only two basic ground terms, $\mathsf{f}(\mathsf{a})$ and $\mathsf{g}(\mathsf{a})$. Since the longest innermost derivation starting from these terms is of length 3, $\mathcal{R}$ has constant innermost runtime complexity w.r.t. ground basic terms. But there is an infinite innermost derivation starting from the non-ground basic term $\mathsf{g}(x)$.

We adopt the following notions from Avanzini and Moser [2]. A *(complexity) problem* $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ consists of two TRSs $\mathcal{S}$, $\mathcal{W}$, and two sets of terms $\mathcal{Q}$, $\mathcal{T}$. We asses the complexity of a problem $\mathcal{P}$ by a *(complexity) judgment* of the form $\vdash \mathcal{P}\colon g$, which is *valid* whenever g is a bound for $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$-derivations starting from $\mathcal{T}$. For sets of functions G we define that $\vdash \mathcal{P}\colon G$ is valid whenever $\vdash \mathcal{P}\colon G$ is valid for some $g \in G$. Often, G is an asymptotic complexity class like $\mathcal{O}(n^3)$. A *(complexity) processor* turns a given judgment $\vdash \mathcal{P}\colon G$ into a list of judgments $\vdash \mathcal{P}_1\colon G_1, \ldots, \vdash \mathcal{P}_n\colon G_n$. It is *sound* whenever the validity of each of $\vdash \mathcal{P}_i\colon G_i$ also implies validity of $\vdash \mathcal{P}\colon G$. A processor is *terminal* if the returned list of judgments is empty.

The problem $\mathcal{P}$ is called a *runtime complexity problem* if $\mathcal{T} = \mathsf{BT}(\mathcal{D},\mathcal{C},\mathcal{V})$, with $\mathcal{S}$ and $\mathcal{W}$ not defining any constructor $\mathcal{C}$, i.e., $\mathcal{D}(\mathcal{S}\cup\mathcal{W})\cap\mathcal{C} = \varnothing$. The problem $\mathcal{P}$ is called an *innermost problem* if $\mathsf{NF}(\mathcal{Q}) \subseteq \mathsf{NF}(\mathcal{S}\cup\mathcal{W})$. In this case, $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ is a composition of innermost rewrite steps with respect to $\mathcal{S}\cup\mathcal{W}$.

As a first example processor, we formulate a theorem by Zankl and Korp [20, Thm. 3.6] within our framework which admits modular complexity proofs.

▶ **Theorem 2** (Split Processor). *Let $\mathcal{P} = \langle \mathcal{S}_1 \cup \mathcal{S}_2/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ be a complexity problem and define $\mathcal{P}_1 = \langle \mathcal{S}_1/\mathcal{S}_2 \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ and $\mathcal{P}_2 = \langle \mathcal{S}_2/\mathcal{S}_1 \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$. The* split processor *translates the judgment $\vdash \mathcal{P}\colon \mathcal{O}(g)$ into the judgments $\vdash \mathcal{P}_1\colon \mathcal{O}(g)$ and $\vdash \mathcal{P}_2\colon \mathcal{O}(g)$.*

The split processor is sound.

▶ **Example 3.** The split processor is used whenever rules should be shifted from the strict into the weak component, e.g., when applying match-bounds for relative rewriting or when using orderings. As an example, consider a TRS with rules numbered from 1 to 5 where cubic complexity has been proven. In the proof, first rules 2, 3, and 4 have been oriented strictly and rules 1 and 5 are oriented weakly by some ordering o_1 with quadratic complexity. Afterwards rule 1 could be moved into the weak component by match-bounds, and finally rule 5 is oriented strictly by some ordering o_2 with cubic complexity, where the remaining rules 1 to 4 are oriented weakly. This proof is restructured via split as follows. First, the initial complexity judgment $\vdash \langle\{1,2,3,4,5\}/\varnothing, \mathcal{Q}, \mathcal{T}\rangle\colon \mathcal{O}(n^3)$ is splitted into $\vdash \langle\{2,3,4\}/\{1,5\}, \mathcal{Q}, \mathcal{T}\rangle\colon \mathcal{O}(n^3)$ and $\vdash \langle\{1,5\}/\{2,3,4\}, \mathcal{Q}, \mathcal{T}\rangle\colon \mathcal{O}(n^3)$ by the split processor where the former judgment is validated via o_1. The latter complexity problem is split again into $\vdash \langle\{1\}/\{2,3,4,5\}, \mathcal{Q}, \mathcal{T}\rangle\colon \mathcal{O}(n^3)$ and $\vdash \langle\{5\}/\{1,2,3,4\}, \mathcal{Q}, \mathcal{T}\rangle\colon \mathcal{O}(n^3)$ where the former judgment is validated via match-bounds, and the latter one via o_2.

The example demonstrates that via splitting it suffices to restrict match-bounds and orderings to terminal complexity processors. This is the reason why we present both techniques as terminal processors in Sections 4 and 6.

4 Match-Bounds

The match-bounds technique was introduced as termination method by Geser et al. [7]. We shortly recapitulate the main underlying ideas, before explaining our formalization (theory `Matchbounds`) and the necessary adaptations to use it for complexity analysis [19, 20].

Let $\mathcal{F}$ be a signature containing at least one constant. For match-bounds, $\mathcal{F}$ is expanded such that symbols are labeled by natural numbers, i.e., $\mathcal{F}' = \mathcal{F} \times \mathbb{N}$. Moreover, we define auxiliary functions $\mathsf{base} : \mathcal{T}(\mathcal{F}', \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, $\mathsf{lift}_d : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}', \mathcal{V})$, and $\mathsf{lab} : \mathcal{T}(\mathcal{F}', \mathcal{V}) \to 2^{\mathbb{N}}$, where base removes all labels of a term, lift_d labels all symbols of a term by d, and lab returns the set of labels of a term. For a non-duplicating TRS $\mathcal{R}$ over signature $\mathcal{F}$ we construct the TRS $\mathcal{R}' = \mathsf{match}(\mathcal{R})$ over $\mathcal{F}'$.

$$\mathsf{match}(\mathcal{R}) = \{\ell' \to \mathsf{lift}_d(r) \mid \ell \to r \in \mathcal{R}, \mathsf{base}(\ell') = \ell, d = 1 + \min(\mathsf{lab}(\ell'))\} \tag{1}$$

Then for left-linear $\mathcal{R}$, every rewrite step $s \to_{\mathcal{R}} t$ can be simulated by a step $s' \to_{\mathcal{R}'} t'$ with $\mathsf{base}(t') = t$, provided $\mathsf{base}(s') = s$. Hence, every (possibly) infinite derivation (2) gives rise to a step-wise simulation (3) provided $\mathsf{base}(t'_0) = t_0$, which is ensured by choosing $t'_0 = \mathsf{lift}_0(t_0)$.

$$t_0 \quad \to_{\mathcal{R}} \quad t_1 \quad \to_{\mathcal{R}} \quad t_2 \quad \to_{\mathcal{R}} \quad \cdots \tag{2}$$

$$t'_0 \quad \to_{\mathcal{R}'} \quad t'_1 \quad \to_{\mathcal{R}'} \quad t'_2 \quad \to_{\mathcal{R}'} \quad \cdots \tag{3}$$

$$\mathsf{mul}(t'_0) >_{\mathsf{ms}} \mathsf{mul}(t'_1) >_{\mathsf{ms}} \mathsf{mul}(t'_2) >_{\mathsf{ms}} \quad \cdots \tag{4}$$

As the next step, a function mul maps every term t'_i to the multiset of negated labels, where by construction of $\mathsf{match}(\mathcal{R})$ every step with $\mathcal{R}'$ results in a strict decrease w.r.t. the standard multiset-order $>_{\mathsf{ms}}$ on the integers, and thus we can construct (4) from (3).

Let us assume that the initial term t_0 is ground. Then $t'_0 = \mathsf{lift}_0(t_0)$ implies that the initial term in (3) is always a member of $\mathcal{T}(\mathcal{F} \times \{0\})$. We now try to find some bound $b \in \mathbb{N}$, such that $\to^*_{\mathcal{R}'}(\mathcal{T}(\mathcal{F} \times \{0\})) \subseteq \mathcal{T}(\mathcal{F} \times \{0, \ldots, b\})$. If this succeeds, then the labels in derivation (3) are bounded by b, and hence, all numbers in (4) are in the range $-b, \ldots, 0$. Since $>$ is well-founded on this domain, so is $>_{\mathsf{ms}}$. Hence, (4) cannot be infinite, and therefore, also (3), and (2) cannot be infinite, proving termination of $\mathcal{R}$ on ground terms. Moreover, since $\mathcal{F}$ contains at least one constant, termination on ground terms implies termination on all terms.

In total, we formalized the following theorem for termination analysis.

▶ **Theorem 4.** *If $\mathcal{R}$ is a non-duplicating, left-linear TRS over signature $\mathcal{F}$, and there is some language L satisfying $\to^*_{\mathcal{R}'}(\mathsf{lift}_0(\mathcal{T}(\mathcal{F}))) \subseteq L \subseteq \mathcal{T}(\mathcal{F} \times \{0, \ldots, b\})$, then $\mathcal{R}$ is terminating.*

Here, non-duplication is essential in the step from (3) to (4), and left-linearity is required to ensure the one-step simulation property (Lemma 5). The language L usually comes in the form of a finite automaton which has been constructed via tree automata completion [6].

▶ **Lemma 5.** *If $\mathcal{R}$ is left-linear, $s \to_{\mathcal{R}} t$, and $\mathsf{base}(s') = s$, then there exists a term t' such that $s' \to_{\mathcal{R}'} t'$ and $\mathsf{base}(t') = t$.*

The lemma is straightforward to prove on paper, and also its formalization posed no difficulties. Actually, it is no longer present, since IsaFoR now includes a full proof of a more general result by Korp and Middeldorp [12, Lemma 12], applying also to non-left-linear TRSs. It is the essential ingredient to obtain (3) from (2).

Concerning the step from (3) to (4), in the formalization we already require the bound b at this point. This allows us to include an index shift in mul, so that each label i is mapped onto $b - i \in \mathbb{N}$. Then the parameter $>$ of $>_{\mathsf{ms}}$ in (4) is the standard order on natural numbers.

In order to certify match-bounds proofs (which are required to contain L in the form of an automaton), CeTA must be able to check left-linearity and non-duplication, as well as that the given automaton indeed accepts all terms in $\to^*_{\mathcal{R}'}(\mathcal{T}(\mathcal{F} \times \{0\}))$. For the latter, we make use of earlier work by Felgenhauer and Thiemann [5], and for the former, we rounded off Isabelle/HOL's existing theory on multisets by algorithms for comparing multisets (since a rule is non-duplicating if and only if the multiset of variables of its right-hand side is a subset of the multiset of variables of its left-hand side).

In case $\mathcal{F}$ does not contain a constant, e.g., in case of string rewrite systems, CeTA does an automatic preprocessing step, which invents a fresh constant, includes it into the signature, and adjusts the automaton accordingly.

In the remainder of this section, we adapt Theorem 4 and the corresponding formalization towards complexity analysis, following Zankl and Korp [19, 20].

The first step is to integrate complexity bounds into (2), (3), and (4), starting from (4). Given a term of size n, the initial value $\mathsf{mul}(t'_0)$ is the multiset containing n times the value b. However, this does not immediately give a nice bound on the length of (4), since $>_{\mathsf{ms}}$ does not impose any bound on the length of derivations w.r.t. the initial multiset: $\{\{1\}\} >_{\mathsf{ms}} \{\{0,\dots,0\}\} >_{\mathsf{ms}} \cdots >_{\mathsf{ms}} \{\{0\}\} >_{\mathsf{ms}} \varnothing$. Thus, we replace $>_{\mathsf{ms}}$ by $>_{\mathsf{ms},k}$ in (4), where $>_{\mathsf{ms},k}$ is a bounded version of $>_{\mathsf{ms}}$ such that at most k elements may be added in each comparison: $X >_{\mathsf{ms},k} Y$ if and only if $X = U \cup V$, $Y = U \cup W$, $V >_{\mathsf{ms}} W$, and $|W| \leq k$.

Of course, we have to substitute $>_{\mathsf{ms},k}$ (with suitable k) for $>_{\mathsf{ms}}$ in all previous proofs. Doing so within the formalization was an easy task: take $k \geq 1$ as the maximum size of right-hand sides of $\mathcal{R}$. After this adaptation, it is shown that the length of $>_{\mathsf{ms},k}$-sequences is linearly bounded, using a result by Dershowitz and Manna [4, page 191]. To be more precise, we formalized that $X >^n_{\mathsf{ms},k} Y$ implies $n \leq \sum_{x \in X}(k+1)^x$, leading to the linear bound: Recall, that $\mathsf{mul}(t'_0) = \{b,\dots,b\}$ where the number of b's is $|t_0|$. Hence, sequence (4) can be of length at most $\sum_{x \in \mathsf{mul}(t'_0)}(k+1)^x = \sum_{1,\dots,|t_0|}(k+1)^b = (k+1)^b \cdot |t_0|$. As immediate consequence we conclude that also (3) and (2) are linearly bounded.

In total, we get the following result which is used in CeTA to check complexity proofs via match-bounds, where $\mathcal{T}_{gnd}$ is the set of all ground terms in $\mathcal{T}$. The restriction to ground terms is possible at this point (in contrast to Example 1) as $\mathcal{Q}$ is ignored in the analysis.

▶ **Theorem 6.** *Let $\mathcal{P} = \langle \mathcal{R}/\varnothing, \mathcal{Q}, \mathcal{T}\rangle$ be a complexity problem. If $\mathcal{R}$ is a non-duplicating and left-linear TRS over signature $\mathcal{F}$, and there is some language L satisfying $\to^*_{\mathcal{R}'}(\mathsf{lift}_0(\mathcal{T}_{gnd})) \subseteq L \subseteq \mathcal{T}(\mathcal{F} \times \{0,\dots,b\})$, then $\vdash \mathcal{P}\colon \mathcal{O}(n)$.*

The next step is to integrate relative rewriting. The main idea to handle weak rules is to use a modified version of match, which only has to ensure a decrease w.r.t. the weak multiset order $\geq_{\mathsf{ms},k}$. To this end, Zankl and Korp [19] define $\mathsf{match\text{-}rt}$ as in (1) except that the value of d in (1) is sometimes reduced. If $|\ell| \geq |r|$ and all labels in ℓ' are identical, then d is $\min(\mathsf{lab}(\ell'))$ instead of $1 + \min(\mathsf{lab}(\ell'))$. Hence, for some cases it is not required to increase the labels at all, and thus, it is more likely that a bound on the labels can be obtained.

In order to integrate match-rt into IsaFoR we could mostly reuse or slightly generalize the existing proofs.

Zankl and Korp give another optimization of match-rt, integrating the bound b: match-rt$_b$ is defined in the same way as match-rt, except that $\mathsf{lift}_d(r)$ is replaced by $\mathsf{lift}_{\min(b,d)}(r)$, which results in even smaller labels than match-rt, but which is restricted to non-collapsing strict rules. In total, we have formalized the following theorem.

▶ **Theorem 7.** *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ be a complexity problem. Let $\mathcal{S} \cup \mathcal{W}$ be a non-duplicating and left-linear TRS over signature $\mathcal{F}$. Assume that $\mathcal{R}' = \mathsf{match}(\mathcal{S}) \cup \mathsf{match\text{-}rt}(\mathcal{W})$, or both $\mathcal{R}' = \mathsf{match}(\mathcal{S}) \cup \mathsf{match\text{-}rt}_b(\mathcal{W})$ and $\mathcal{S}$ is non-collapsing. If there is some language L satisfying $\to^*_{\mathcal{R}'}(\mathsf{lift}_0(\mathcal{T}_{gnd})) \subseteq L \subseteq \mathcal{T}(\mathcal{F} \times \{0, \dots, b\})$, then $\vdash \mathcal{P} \colon \mathcal{O}(n)$.*

Note that $\mathcal{Q}$ is completely ignored in Theorem 7, since the whole analysis does not take the strategy into account. In fact, the theorem was first proven for $\mathcal{Q} = \varnothing$, while the above statement including $\mathcal{Q}$ follows from $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} \subseteq \to_{\mathcal{S}/\mathcal{W}}$. The sole reason for this naive integration of $\mathcal{Q}$ was to support match-bounds on innermost problems in the first place. An alternative might be a dedicated processor that transforms $\langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ into $\langle \mathcal{S}/\mathcal{W}, \varnothing, \mathcal{T} \rangle$.

When integrating match-rt$_b$ in the formalization, we encountered two problems. First, we wanted to get rid of the choice in Theorem 7 and always use the better match-rt$_b$ variant. The reason for this aim was that—while the non-collapsing condition on $\mathcal{S}$ appears inside their proofs—Zankl and Korp [20] did not state that its absence violates the main theorem. This is now shown by a counterexample.

▶ **Example 8** (Non-collapsing condition required)**.** Let $\mathcal{S} = \{\mathsf{f}(x) \to x\}$ and $\mathcal{W} = \{\mathsf{a} \to \mathsf{f}(\mathsf{a})\}$. For $\mathcal{R}' = \mathsf{match}(\mathcal{S}) \cup \mathsf{match\text{-}rt}_0(\mathcal{W}) = \{\mathsf{f}_i(x) \to x, \mathsf{a}_i \to \mathsf{f}_0(\mathsf{a}_0)\}$, the language $\to^*_{\mathcal{R}'}(\mathsf{lift}_0(\mathcal{T}(\mathcal{F})))$ is exactly $\mathcal{T}(\mathcal{F} \times \{0\})$. Without the non-collapsing condition within Theorem 7 one would be able to conclude linear derivational complexity of $\mathcal{S}/\mathcal{W}$, a contradiction.

Hence, the choices in Theorem 7 are really incomparable, and for certification it would be best to include both. Which brings us to the second problem: we did not want to copy and paste the existing proof for match-rt, and then incorporate all the tiny modifications that are required for match-rt$_b$. Thus, in IsaFoR we defined an auxiliary relation covering all of match, match-rt, and match-rt$_b$, and formalized the main proof step only once.

Currently, CeTA always chooses match-rt$_b$ for non-collapsing $\mathcal{S}$, and match-rt, otherwise—the same as in current complexity tools.

5 Certifying Weak Dependency Pairs and Dependency Tuples

The dependency pair framework [9] is a popular setting for termination analysis. Since dependency pairs (DPs for short) in their original definition are not suitable for ensuring small (i.e., polynomial) derivation bounds [13], two variants have been developed. Hirokawa and Moser [10] introduced *weak dependency pairs* (WDPs for short). In general however, one cannot concentrate on counting WDP steps alone. Rather, one also has to take the number of interleaved steps w.r.t. the original TRS into account. Overcoming this complication, Noschinski et al. [15] introduced a variation, called *dependency tuples* (DTs for short). The DT transformation is however only applicable to innermost problems and it is not complete, so that (non-confluent) TRSs with polynomial complexity can be turned into complexity problems of exponential complexity.

Both WDPs and DTs enjoy nice properties that enable us to restrict to usable rules and limit the monotonicity requirements for reduction pairs, which we discuss later. Since the

two techniques are incomparable but both used in modern complexity tools, we provide a formalization of either in IsaFoR. To be more precise, we have formalized the corresponding complexity processors of Avanzini and Moser [2], which—unlike DPs—allow us to apply WDPs and DTs also to relative problems.

As a case study, we decided to perform two different styles of proof: For DTs, we stuck more to the original paper proof, where parallel *positions* are used to point to subterms that are potential redexes; while for WDPs, we instead focused on *contexts* around potential redexes. The former requires us to reason about valid positions, whereas the latter makes it necessary to explicitly manage properties of contexts. Although both paper proofs are of comparable length, in our formalization the theories on WDPs are around 30 % shorter than those on DTs (see also `DT_Transformation(_Impl)` and `WDP_Transformation(_Impl)`). We suspect that this is not mere coincidence, but caused by the fact that contexts can be mostly treated via explicit recursive functions, while positions require a different style of proof that is not as amenable to automation.

For the remainder of this section, we fix a runtime complexity problem $\langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ over signature $\mathcal{F}$. For each $f \in \mathcal{F}$, let $f^\sharp$ be a function symbol fresh with respect to $\mathcal{F}$. For a term t we denote *sharping* its root symbol by $\sharp(t)$, where $\sharp(x) = x$ and $\sharp(f(t_1, \ldots, t_n)) = f^\sharp(t_1, \ldots, t_n)$. Sharping is homomorphically extended to sets and lists of symbols and terms.

Weak Dependency Pairs

We start with our formalization of WDPs as defined by Hirokawa and Moser [10].

▶ **Definition 9.** Let $\mathcal{R}$ be a TRS with defined symbols $\mathcal{D}(\mathcal{R})$. For every rule $\ell \to r \in \mathcal{R}$, let $\mathsf{WDP}(\ell \to r)$ denote the new rule $\sharp(\ell) \to \mathsf{COM}(\sharp(u_1), \ldots, \sharp(u_n))$, where $u_1, \ldots, u_n$ are the maximal subterms of r that are either variables or have a root symbol in $\mathcal{D}(\mathcal{R})$. Then the *weak dependency pairs* of $\mathcal{R}$ are defined by $\mathsf{WDP}(\mathcal{R}) = \{\mathsf{WDP}(\ell \to r) \mid \ell \to r \in \mathcal{R}\}$.

In the above definition COM denotes a "function" that assigns fresh function symbols of appropriate arity (a common optimization is to omit such symbols in case the argument list is singleton, i.e., $\mathsf{COM}(t) = t$) to a given list of terms. The thusly generated symbols are called *compound symbols*. Note that Definition 9 implies that for each rule $\ell \to r$ there is a unique ground context C such that $r = C[u_1, \ldots, u_n]$. This is captured by the following two functions:

$$\mathsf{cap}_{\mathcal{D}}(t) = \begin{cases} \Box & \text{if } t \in \mathcal{V} \text{ or } \mathsf{root}(t) \notin \mathcal{C} \\ f(\mathsf{cap}_{\mathcal{D}}(t_1), \ldots, \mathsf{cap}_{\mathcal{D}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \in \mathcal{C} \end{cases}$$

$$\mathsf{max}_{\mathcal{D}}(t) = \begin{cases} t & \text{if } t \in \mathcal{V} \text{ or } \mathsf{root}(t) \notin \mathcal{C} \\ \mathsf{max}_{\mathcal{D}}(t_1), \ldots, \mathsf{max}_{\mathcal{D}}(t_n) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \in \mathcal{C} \end{cases}$$

where $\mathcal{C}$ is a set of symbols—which is supposed to contain the compound symbols and the constructors of $\mathcal{S} \cup \mathcal{W}$—that is disjoint from sharped $\mathcal{F}$-symbols and the defined symbols of $\mathcal{S} \cup \mathcal{W}$, i.e., $(\mathcal{D}(\mathcal{S} \cup \mathcal{W}) \cup \sharp(\mathcal{F})) \cap \mathcal{C} = \varnothing$. Intuitively, $\mathsf{max}_{\mathcal{D}}(t)$ results in the list of maximal subterms of t that are either variables or have a root not in $\mathcal{C}$ (the latter usually implies that the root is a defined symbol; hence the notation), whereas $\mathsf{cap}_{\mathcal{D}}(t)$ computes the surrounding context. Together these two functions constitute a unique decomposition of a given term t, satisfying the property $t = (\mathsf{cap}_{\mathcal{D}}(t))[\mathsf{max}_{\mathcal{D}}(t)]$.

For certification we never actually have to construct the set of WDPs.[3] Instead it suffices to check whether a given pair of terms (p, q) constitutes a WDP for a given rule $\ell \to r$. This is done via the predicate:

$$\mathsf{is\text{-}WDP}(p,q)(\ell \to r) \iff p = \sharp(\ell) \wedge (\exists C.\, \mathsf{ground}(C) \wedge \mathcal{F}\mathsf{un}(C) \subseteq \mathcal{C} \wedge q = C[\sharp(\mathsf{max}_{\mathcal{D}}(r))])$$

In preparation for later results, we somewhat ambiguously use $\mathsf{WDP}(\mathcal{R})$ for $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{W}$ to denote an arbitrary set of rules (to be provided by the certificate) that is obligated to contain a WDP for each rule in $\mathcal{R}$, i.e.,

$$\forall \ell \to r \in \mathcal{R}.\, \exists (p,q) \in \mathsf{WDP}(\mathcal{R}).\, \mathsf{is\text{-}WDP}(p,q)(\ell \to r) \tag{5}$$

The main ingredient for soundness of WDPs is a simulation lemma that states that when two terms are in a certain relation, then every $\mathcal{R}$-rewrite sequence starting from the first term can be simulated by a $\mathsf{WDP}(\mathcal{R}) \cup \mathcal{R}$-rewrite sequence starting from the second one. The mentioned relation is crafted to fit the definition of WDPs. Intuitively, it relates terms whose respective maximal defined subterms (computed by $\mathsf{max}_{\mathcal{D}}$) only differ by sharp symbols. We write $s_1, \ldots, s_n \trianglelefteq_\sharp t_1, \ldots, t_n$ when for each $i \leq n$ we have that either $s_i = t_i$ or $\sharp(s_i) = t_i$. Then the informal statement from above can be formalized as follows.

▶ **Definition 10.** A term t is *good for* a term s, written $t \gg s$, if and only if $\mathcal{F}\mathsf{un}(s) \subseteq \mathcal{F}$ and there are terms $t_1, \ldots, t_n$ and a ground context C with $\mathcal{F}\mathsf{un}(C) \subseteq \mathcal{C}$ such that $\mathsf{max}_{\mathcal{D}}(s) \trianglelefteq_\sharp t_1, \ldots, t_n$ and $t = C[t_1, \ldots, t_n]$.

We borrow the terminology *good for* from Avanzini [1], although the above definition slightly differs from the original one. As indicated above, its intuition is that two related terms have the same redexes (or rather an over-approximation, namely, subterms with defined root) where in addition those in the left term may be sharped.

Before we state the main lemma, we give some useful properties of $\mathsf{max}_{\mathcal{D}}$.

▶ **Lemma 11.** *Let t be a term with $\mathsf{max}_{\mathcal{D}}(t) = t_1, \ldots, t_n$. Then:*

1. *If $\mathcal{F}\mathsf{un}(t) \subseteq \mathcal{F}$ and $\mathsf{max}_{\mathcal{D}}(t) \trianglelefteq_\sharp u_1, \ldots, u_n$, then $\mathsf{max}_{\mathcal{D}}(u_i) = u_i$ for all $i \leq n$.*
2. *If $\mathcal{F}\mathsf{un}(t\sigma) \subseteq \mathcal{F}$ then $\mathsf{max}_{\mathcal{D}}(t\sigma) \trianglelefteq_\sharp \mathsf{max}_{\mathcal{D}}(\sharp(t_1)\sigma), \ldots, \mathsf{max}_{\mathcal{D}}(\sharp(t_n)\sigma)$.*

In the main simulation lemma below, $\mathcal{Q}$ is extended to a set of terms $\mathcal{Q}'$ taking extensions of the signature $\mathcal{F}$ (by sharped and compound symbols) into account. In particular, the assumption on $\mathcal{Q}'$ ensures that innermost problems are translated to innermost problems, thereby allowing a proof-in-progress to continue with techniques that are specific to the innermost case. The following lemma shows that this does not pose any problems for rewriting, where $\mathcal{Q}_{\neg\mathcal{F}} = \{f(t_1, \ldots, t_n) \mid f \notin \mathcal{F}\}$.

▶ **Lemma 12.** *Every term t with $\mathcal{F}\mathsf{un}(t) \subseteq \mathcal{F}$ that is $\mathcal{Q}$-normal is also $\mathcal{Q}'$-normal for any $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$.*

Proof. Assume that t is not $\mathcal{Q}'$-normal. Then $t = C[\ell\sigma]$ for some C, σ, and $\ell \in \mathcal{Q}'$; thus either $\ell \in \mathcal{Q}$, contradicting $\mathcal{Q}$-normality, or $\ell \in \mathcal{Q}_{\neg\mathcal{F}}$, contradicting $\mathcal{F}\mathsf{un}(t) \subseteq \mathcal{F}$. ◀

▶ **Lemma 13.** *Let $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{W}$ and $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$. If $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ and $u \gg s$ then there is a term v such that $u \xrightarrow{\mathcal{Q}'}_{\mathsf{WDP}(\mathcal{R})\cup\mathcal{R}} v$ and $v \gg t$.*

[3] Which allows us to avoid a tedious formalization of COM that would have to manage the generation of *fresh* symbols using a state monad or similar concept.

Proof. From $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ we obtain $\ell \to r \in \mathcal{R}$, σ, and C with $s = C[\ell\sigma]$ and $t = C[r\sigma]$. Moreover, all proper subterms of $\ell\sigma$ are $\mathcal{Q}$-normal. Let $s_1, \ldots, s_n$ denote the result of $\mathsf{max}_{\mathcal{D}}(s)$. Since every redex has a defined root, and all subterms of s with defined root are either contained in, or subterms of one of $s_1, \ldots, s_n$, we further obtain a context D such that $s_i = D[\ell\sigma]$ for some $i \leq n$. By Definition 10 and $u \gg s$, we get $u_1, \ldots, u_n$ with $s_1, \ldots, s_n \leq_\sharp u_1, \ldots, u_n$ and a ground context D' such that $u = D'[u_1, \ldots, u_n]$ and $\mathcal{F}\mathsf{un}(D') \subseteq \mathcal{C}$. Intuitively, it is easy to see that the above, together with Lemma 11(1), implies $\mathsf{max}_{\mathcal{D}}(u) = u_1, \ldots, u_n$ (although the corresponding formalization is somewhat tedious). Recall that $s = (\mathsf{cap}_{\mathcal{D}}(s))[s_1, \ldots, s_n]$ and the considered redex is a subterm of s_i, thus $t = (\mathsf{cap}_{\mathcal{D}}(s))[\mathsf{max}_{\mathcal{D}}(t)]$ with $\mathsf{max}_{\mathcal{D}}(t) = s_1, \ldots, \mathsf{max}_{\mathcal{D}}(D[r\sigma]), \ldots, s_n$. Moreover, from $s_1, \ldots, s_n \leq_\sharp u_1, \ldots, u_n$ we have $u_i = \sharp(D[\ell\sigma]) \vee u_i = D[\ell\sigma]$ and thus proceed by case analysis:

- Assume $u_i = \sharp(D[\ell\sigma])$.
  - If $D \neq \Box$, then $\mathsf{max}_{\mathcal{D}}(D[r\sigma]) = D[r\sigma]$ and $\mathsf{cap}_{\mathcal{D}}(D[r\sigma]) = \Box$. We define $v = (\mathsf{cap}_{\mathcal{D}}(u))[u_1, \ldots, \sharp(D[r\sigma]), \ldots, u_n]$. Then, $u \xrightarrow{\mathcal{Q}'}_{\mathsf{WDP}(\mathcal{R})\cup\mathcal{R}} v$ with the same rule $\ell \to r$, justified by choosing the context $(\mathsf{cap}_{\mathcal{D}}(u))[u_1, \ldots, \sharp(D), \ldots, u_n]$ and employing Lemma 12. Then, $v \gg t$, by definition of v and $\mathsf{max}_{\mathcal{D}}(t) \leq_\sharp u_1, \ldots, \sharp(D[r\sigma]), \ldots, u_n$.
  - If $D = \Box$, then the WDP corresponding to $\ell \to r$ is used. From (5), we obtain a term q and a ground context E with $(\sharp(\ell), q) \in \mathsf{WDP}(\mathcal{R})$ and $q = E[\sharp(\mathsf{max}_{\mathcal{D}}(r))]$. Define $v = (\mathsf{cap}_{\mathcal{D}}(u))[u_1, \ldots, q\sigma, \ldots, u_n]$. Then $u \xrightarrow{\mathcal{Q}'}_{\mathsf{WDP}(\mathcal{R})\cup\mathcal{R}} v$ as witnessed by $u = (\mathsf{cap}_{\mathcal{D}}(u))[u_1, \ldots, \sharp(\ell)\sigma, \ldots, u_n] \to (\mathsf{cap}_{\mathcal{D}}(u))[u_1, \ldots, q\sigma, \ldots, u_n] = v$ together with Lemma 12 (and noting that u is a proper subterm of $\sharp(\ell)\sigma$ if and only if u is a proper subterm of $\ell\sigma$). Moreover, let $\mathsf{max}_{\mathcal{D}}(r) = r_1, \ldots, r_k$, $E_j = \mathsf{cap}_{\mathcal{D}}(\sharp(r_j)\sigma)$, and $v_j = \mathsf{max}_{\mathcal{D}}(\sharp(r_j)\sigma)$ for all $j \leq k$. Hence, $v \gg t$, with $E' = (\mathsf{cap}_{\mathcal{D}}(u))[\ldots, E[E_1, \ldots, E_k], \ldots]$ and observing that $v = E'[u_1, \ldots, u_{i-1}, v_1, \ldots, v_k, u_{i+1}, \ldots, u_n]$ as well as $\mathsf{max}_{\mathcal{D}}(t) \leq_\sharp u_1, \ldots, u_{i-1}, v_1, \ldots, v_k, u_{i+1}, \ldots, u_n$. The latter follows from $\mathsf{max}_{\mathcal{D}}(r\sigma) \leq_\sharp v_1, \ldots, v_k$, which in turn is a consequence of Lemma 11(2) above.
- Assume $u_i = D[\ell\sigma]$. Then we can again employ the original rule $\ell \to r$. Let $E = (\mathsf{cap}_{\mathcal{D}}(u))[\ldots, \mathsf{cap}_{\mathcal{D}}(D[r\sigma]), \ldots]$ and $v = E[u_1, \ldots, u_{i-1}, \mathsf{max}_{\mathcal{D}}(D[r\sigma]), u_{i+1}, \ldots, u_n]$. We conclude $u \xrightarrow{\mathcal{Q}'}_{\mathsf{WDP}(\mathcal{R})\cup\mathcal{R}} v$ and $v \gg t$ in a similar fashion as in the previous case. ◀

At this point, we obtain a simulation property for relative rewriting as an easy corollary.

▶ **Corollary 14.** *$u \gg s \xrightarrow{\mathcal{Q}}{}^{n}_{\mathcal{S}/\mathcal{W}} t$ implies $u \xrightarrow{\mathcal{Q}'}{}^{n}_{\mathsf{WDP}(\mathcal{S})\cup\mathcal{S}/\mathsf{WDP}(\mathcal{W})\cup\mathcal{W}} v \gg t$ for some v.*

▶ **Theorem 15** (WDP Processor). *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ be a runtime complexity problem. Then the WDP processor transforms $\mathcal{P}$ into $\mathcal{P}' = \langle \mathsf{WDP}(\mathcal{S}) \cup \mathcal{S}/\mathsf{WDP}(\mathcal{W}) \cup \mathcal{W}, \mathcal{Q}', \sharp(\mathcal{T})\rangle$ for an arbitrary $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$, and $\vdash \mathcal{P}' \colon G$ implies $\vdash \mathcal{P} \colon G$.*

Proof. Assume $\vdash \mathcal{P}' \colon g$ for some $g \in G$. Moreover, for the sake of a contradiction, assume that there is a term $s \in \mathcal{T}$ of size n and a rewrite sequence $s \xrightarrow{\mathcal{Q}}{}^{m}_{\mathcal{S}/\mathcal{W}} t$ of length $m > g(n)$. Since $s \in \mathcal{T}$, we have $\sharp(s) \in \mathcal{T}^\sharp$ and trivially $\sharp(s) \gg s$. Moreover, by Corollary 14, we obtain a term v with $\sharp(s) \xrightarrow{\mathcal{Q}'}{}^{m}_{\mathsf{WDP}(\mathcal{S})\cup\mathcal{S}/\mathsf{WDP}(\mathcal{W})\cup\mathcal{W}} v$, thereby contradicting the initial complexity judgment. ◀

▶ Remark. Note that when $\mathcal{P}$ is an innermost problem, by setting $\mathcal{Q}' = \mathcal{Q} \cup \sharp(\mathcal{Q})$ the WDP processor generates again an innermost problem. In contrast, Avanzini and Moser [1, 2] set $\mathcal{Q}'$ to $\mathcal{Q}$, thereby not retaining the innermost status as claimed.

Dependency Tuples

According to Theorem 15, we cannot focus on applications of weak dependency pairs in $\mathsf{WDP}(\mathcal{S})$ alone, but also have to account for applications of rules from $\mathcal{S}$. This may have severe consequences for a proof-in-progress. In the case of reduction pairs for instance, rather strict monotonicity requirements have to be imposed even after the WDP transformation. DTs overcome this weaknesses, but the corresponding transformation is sound only on innermost problems. In contrast to WDPs, which capture outermost calls, a DT captures *all* calls in a rule. The following definition is due to Noschinski et al. [15].

▶ **Definition 16.** Let $\mathcal{R}$ be a TRS with defined symbols $\mathcal{D}(\mathcal{R})$. For every rule $\ell \to r \in \mathcal{R}$, let $\mathsf{DT}(\ell \to r)$ denote the new rule $\sharp(\ell) \to \mathsf{COM}(\sharp(u_1), \ldots, \sharp(u_n))$, where $u_1, \ldots u_n$ are all subterms of r that have a root symbol in $\mathcal{D}(\mathcal{R})$. Then the *dependency tuples* of $\mathcal{R}$ are defined by $\mathsf{DT}(\mathcal{R}) = \{\mathsf{DT}(\ell \to r) \mid \ell \to r \in \mathcal{R}\}$.

As for weak dependency pairs, our formalization uses a predicate to decide whether a pair of terms (p, q) constitutes a dependency tuple of a rule $\ell \to r$. For a term t, let $\mathsf{Pos}_{\mathcal{D}}(t)$ denote the set of positions of subterms rooted by defined symbols of $\mathcal{S} \cup \mathcal{W}$.

$$\mathsf{is\text{-}DT}(p, q)(\ell \to r) \iff p = \sharp(\ell) \wedge \big(\exists C\ p_1\ \ldots\ p_n.\, \mathsf{Pos}_{\mathcal{D}}(r) = \{p_1, \ldots, p_k\} \wedge q = C[\sharp(r|_{p_1}), \ldots, \sharp(r|_{p_n})]\big)\ .$$

In the following, we use the notation $\mathsf{DT}(\mathcal{R})$ where $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{W}$, for a set satisfying

$$\forall \ell \to r \in \mathcal{R}.\, \exists (p, q) \in \mathsf{DT}(\mathcal{R}).\, \mathsf{is\text{-}DT}(p, q)(\ell \to r)\ . \tag{6}$$

In the remainder, we provide a simulation lemma akin to Lemma 13 for DTs. For a term s, let $\mathsf{RPos}(s)$ denote the restriction of $\mathsf{Pos}_{\mathcal{D}}(s)$ to redex-positions. More precisely, $\mathsf{RPos}(s) = \{q \in \mathsf{Pos}_{\mathcal{D}}(s) \mid \exists t.\ s|_q \xrightarrow{\mathcal{Q}}_{\mathcal{S} \cup \mathcal{W}} t\}$. Closely following the proof by Avanzini [1], we use the following notion of *good for*.

▶ **Definition 17.** A term t is *good for* a term s, written $t \ggg s$, if and only if $\mathcal{F}\mathsf{un}(s) \subseteq \mathcal{F}$ and there is a context C such that $t = C[\sharp(s|_{q_1}), \ldots, \sharp(s|_{q_k})]$ for positions $\{q_1, \ldots, q_k\} = \mathsf{RPos}(s)$.

We now show that each $\mathcal{R}$-derivation of length n can be simulated by a corresponding derivation of $\mathsf{DT}(\mathcal{R})$ relative to $\mathcal{R}$, of length n. In the proof of the central simulation lemma, we use the following key observations.

▶ **Lemma 18.** *Let $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{W}$. Suppose $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ is a step at redex position p with rule $\ell \to r$. Abbreviate $P = \{pq \mid q \in \mathsf{Pos}_{\mathcal{D}}(r)\}$ and $Q = \{q \in \mathsf{RPos}(s) \mid q < p \vee q \parallel p\}$. Then:*

1. *If $\mathsf{NF}(\mathcal{Q}) \subseteq \mathsf{NF}(\mathcal{S} \cup \mathcal{W})$ then $\mathsf{RPos}(t) \subseteq P \cup Q$;*
2. *$\sharp(s|_p) \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(\mathcal{R})} C[\sharp(t|_{p_1}), \ldots, \sharp(t|_{p_n})]$ for some context C and $\{p_1, \ldots, p_n\} = P$;*
3. *$\sharp(s|_q) \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} \sharp(t|_q)$ for all positions $q \in Q$.*

▶ **Lemma 19.** *Let $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{W}$, suppose $\mathsf{NF}(\mathcal{Q}) \subseteq \mathsf{NF}(\mathcal{S} \cup \mathcal{W})$, and let $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$. If $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ and $u \ggg s$, then there is a term v such that $u \xrightarrow{\mathcal{Q}'}{}^*_{\mathcal{R}} \cdot \xrightarrow{\mathcal{Q}'}_{\mathsf{DT}(\mathcal{R})} v$ and $v \ggg t$.*

Proof. Consider terms s, t and u with $u \ggg s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$. Let p denote the corresponding redex position. Define a function f from positions in s to marked terms as follows: $f(q) = \sharp(t|_q)$ if $q < p$ or $q \parallel p$ and $f(q) = \sharp(s|_q)$ otherwise. Since u is good for s, by Definition 17 we obtain a context C such that $u = C[\sharp(s|_{q_1}), \ldots, \sharp(s|_{q_k})]$ for positions $\{q_1, \ldots, q_k\} = \mathsf{RPos}(s)$. From Lemma 18(3) and the definition of f we see that $\sharp(s|_{q_i}) \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} f(q_i)$ $(i = 1, \ldots, k)$ holds.

Since $\mathcal{F}\mathsf{un}(s_j) \subseteq \mathcal{F}$ holds by assumption for all arguments of s_j of $\sharp(s|_{q_i})$, with Lemma 12 we can refine these sequences to $\sharp(s|_{q_i}) \xrightarrow{\mathcal{Q}'}{}^*_{\mathcal{R}} f(q_i)$ $(i = 1, \ldots, k)$.

Observe that $p \in \mathsf{RPos}(s)$, i.e. $p = q_i$ for some $i \in \{1, \ldots, k\}$. In particular $\sharp(s|_{q_i}) = \sharp(s|_p) = f(q_i)$ by definition of f. Lemma 18(2) yields a context D such that $f(q_i) \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(\mathcal{R})} D[\sharp(t|_{p_1}), \ldots, \sharp(t|_{p_n})]$, and consequently $f(q_i) \xrightarrow{\mathcal{Q}'}_{\mathsf{DT}(\mathcal{R})} D[\sharp(t|_{p_1}), \ldots, \sharp(t|_{p_n})]$, for positions $\{p_1, \ldots, p_n\} = \{pq \mid q \in \mathsf{Pos}_{\mathcal{D}}(r)\}$. Putting things together, we can thus construct a rewrite sequence

$$C[\sharp(s|_{q_1}), \ldots, \sharp(s|_{q_i}), \ldots, \sharp(s|_{q_k})] \xrightarrow{\mathcal{Q}'}{}^*_{\mathcal{R}} C[f(q_1), \ldots, f(q_i), \ldots, f(q_k)] \xrightarrow{\mathcal{Q}'}_{\mathsf{DT}(\mathcal{R})} C[f(q_1), \ldots, D[\sharp(t|_{p_1}), \ldots, \sharp(t|_{p_n})], \ldots, f(q_k)] \,.$$

Let v be the last term of this sequence. We claim that v is good for t. Abbreviate $Q = \{q \in \mathsf{RPos}(s) \mid q < p \vee q \parallel p\}$. Observe that by Lemma 18(1), $\mathsf{RPos}(t) \subseteq Q \cup \{p_1, \ldots, p_n\}$ holds. Since in particular $Q \subseteq \{q_1, \ldots, q_{i-1}, q_{i+1}, \ldots q_k\}$ and $f(q) = \sharp(t|_q)$ holds by definition of f for all positions $q \in Q$, it is not difficult to see that $v \ggg t$ holds. ◀

The lemma is straightforward to generalize to Q-restricted relative rewrite sequences.

▶ **Corollary 20.** *Suppose $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R} \cup \mathcal{S})$, and let $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$. Then $u \ggg s \xrightarrow{\mathcal{Q}}{}^n_{\mathcal{S}/\mathcal{W}} t$ implies $u \xrightarrow{\mathcal{Q}'}{}^n_{\mathit{DT}(\mathcal{S})/\mathit{DT}(\mathcal{W}) \cup \mathcal{S} \cup \mathcal{W}} v \ggg t$ for some v.*

▶ **Theorem 21** (DT Processor). *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ be an innermost runtime complexity problem. Then the DT processor transforms $\mathcal{P}$ into $\mathcal{P}' = \langle \mathit{DT}(\mathcal{S})/\mathit{DT}(\mathcal{W}) \cup \mathcal{S} \cup \mathcal{W}, \mathcal{Q}', \sharp(\mathcal{T}) \rangle$ for an arbitrary $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$, and $\vdash \mathcal{P}' \colon G$ implies $\vdash \mathcal{P} \colon G$.*

Proof. Soundness follows from Corollary 20, by reasoning similar to Theorem 15. ◀

Whenever WDPs or DTs are employed in a complexity proof, CeTA requires a clear indication which of the two methods has been applied. Moreover, the set of all WDPs (or DTs) has to be provided, as well as the extension of the strategy component: $\mathcal{Q}' \setminus \mathcal{Q}$. All other information is computed by CeTA, e.g., the set of compound symbols, renaming of variables, etc.

6 Usable Rules and Usable Replacement Maps

Computing usable rules is a simple syntactic technique for innermost termination; detecting that certain rules can never be applied in derivations starting from a given set of terms, and may thus be discarded. While for termination analysis, we start from right-hand sides of dependency pairs (instantiated by normal form substitutions); for complexity analysis, we employ the corresponding set of starting terms. Existing results on innermost usable rules for termination analysis made it quite easy to integrate usable rules for complexity analysis into IsaFoR, cf. `Usable_Rules_Complexity(_Impl)` and `Usable_Replacement_Map(_Impl)`.

Avanzini [1, Def. 14.44] as well as Hirokawa and Moser [10, Def. 14] define usable rules via usable symbols. Our formalization simplifies and generalizes both definitions.

▶ **Definition 22.** Let the set of starting terms $\mathcal{T}$ be included in $\mathcal{T}(\mathcal{F}', \mathcal{V})$. We define $\mathcal{US}$ to be a set of usable symbols and $\mathcal{U}$ a set of usable rules for $\mathcal{S}/\mathcal{W}$ w.r.t. $\mathcal{T}$, if the following three conditions are satisfied..

- $\mathcal{F}' \subseteq \mathcal{US}$
- whenever $\ell \to r \in \mathcal{S} \cup \mathcal{W}$ and $\mathcal{F}\mathsf{un}(\ell) \subseteq \mathcal{US}$, then $\ell \to r \in \mathcal{U}$, and
- whenever $\ell \to r \in \mathcal{U}$ then $\mathcal{F}\mathsf{un}(r) \subseteq \mathcal{US}$.

We believe the above definition to be simpler than previous ones, since we avoid reflexive transitive closures and do not distinguish between dependency pairs and other rules. Still, it is easy to check that Definition 22 simulates previous definitions, by choosing $\mathcal{US} = \mathcal{F}' \cup \mathcal{F}\mathsf{un}(\mathcal{U})$, where $\mathcal{U}$ is the set of usable rules as defined by Avanzini [1] and Hirokawa and Moser [10]. Moreover, Definition 22 is a generalization of the former, since all symbols of left-hand sides are considered, as opposed to just root symbols.

▶ **Example 23.** Let $\mathcal{S} \cup \mathcal{W} = \{\mathsf{f}^\sharp(\mathsf{g}) \to \mathsf{f}^\sharp(\mathsf{f}(\mathsf{g})), \mathsf{g}^\sharp \to \mathsf{com}, \mathsf{f}(\mathsf{g}) \to \mathsf{f}(\mathsf{f}(\mathsf{g})), \mathsf{g} \to \mathsf{a}\}$ and $\mathcal{T} = \mathsf{BT}(\{\mathsf{f}^\sharp, \mathsf{g}^\sharp\}, \{\mathsf{a}\}, \mathcal{V})$. Then according to [1, 10] all rules are usable, whereas Definition 22 allows us to use $\mathcal{F}' = \{\mathsf{f}^\sharp, \mathsf{g}^\sharp, \mathsf{a}\}$, $\mathcal{US} = \{\mathsf{f}^\sharp, \mathsf{g}^\sharp, \mathsf{a}, \mathsf{com}\}$, and $\mathcal{U} = \{\mathsf{g}^\sharp \to \mathsf{com}\}$.

For soundness of usable rules it is easy to prove that every derivation starting from $\mathcal{T}$ does only contain terms in $\mathcal{T}(\mathcal{US}, \mathcal{V})$. Hence we can remove all non-usable rules.

▶ **Theorem 24.** *If $\mathcal{U}$ is a set of usable rules for $\mathcal{S}/\mathcal{W}$ w.r.t. $\mathcal{T}$, then $\vdash \langle \mathcal{S} \cap \mathcal{U} / \mathcal{W} \cap \mathcal{U}, \mathcal{Q}, \mathcal{T} \rangle \colon G$ implies $\vdash \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle \colon G$.*

The whole formalization of this theorem via usable symbols, including definitions, occupies only 100 lines, without having to reuse existing results on usable rules in IsaFoR. This is in contrast to IsaFoR's integration of the variant of usable rules used in AProVE, cf. the end of Section 5.1 in [15]. Here, usable rules are based on unification and normal form checks, but only work for innermost rewriting. In this part of the formalization, we heavily reused the existing results for termination, and only little had to be added w.r.t. complexity analysis. As an example, for complexity with its relative rewrite relation, it was required to switch between a sequence of $\mathcal{S}/\mathcal{W}$-steps and a sequence that explicitly lists every single step in each relative $\to_{\mathcal{W}}^* \cdot \to_{\mathcal{S}} \cdot \to_{\mathcal{W}}^*$-step.

Since both variants of usable rules are incomparable, CeTA supports both. The certificate just requires the set of usable rules. It is then automatically inferred which of the two variants of usable rules is applicable.

Even less usable rules are obtained when employing argument filters from reduction pairs, a well-known technique from termination analysis. This technique has already been adapted for complexity, but we did not find any details in the literature. Thus, in the remainder of this section, we clarify how usable rules, reduction pairs, argument filters, and usable replacement maps can be combined. The upcoming theorem generalizes and improves existing complexity results on reduction pairs ([1, Thm. 14.10], [11, Cor. 20], and [15, Thm. 26]), since usable replacement maps can simulate safe reduction pairs of [11], cf. [1, Lemma 14.34].

Before presenting the main theorem, we first recapitulate the notion of usable replacement maps ([1, Def. 14.5] and [11, Def. 8]). These mainly indicate a superset of all positions where redexes may occur within terms of a derivation. To be more precise, for a replacement map μ, two TRSs $\mathcal{R}$ and $\mathcal{R}'$, and two sets of terms $\mathcal{Q}$ and $\mathcal{T}$; μ is a *usable replacement map* (written $\mathsf{URM}(\mu, \mathcal{Q}, \mathcal{R}, \mathcal{T}, \mathcal{R}')$), if for all $t \in \mathcal{T}$ and $t \xrightarrow{\mathcal{Q}}_{\mathcal{R}}^* s$, all redexes of s w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}'}$ are at μ-replacing positions of s.

Sufficient criteria to estimate usable replacement maps have been described in [11] for full and innermost rewriting, and in [1, Lemma 14.34] for WDPs and DTs, where currently CeTA only supports innermost rewriting, WDPs and DTs.

We will first present the main theorem, and then explain its ingredients and how to apply it. Here, a complexity pair $(\succ, \succsim)$ consists of two partial orders which are both closed under substitutions, which are compatible ($\succsim \cdot \succ \cdot \succsim \subseteq \succ$) and where $\succsim$ is reflexive. A reduction pair is a complexity pair where $\succsim$ is closed under contexts and $\succ$ is strongly normalizing.

▶ **Theorem 25.** *Let $\langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ be an innermost runtime complexity problem with $\mathcal{T} = \mathsf{BT}(\mathcal{D}, \mathcal{C}, \mathcal{V})$. Define $\mathcal{R} = \mathcal{S} \cup \mathcal{W}$. Let $\mu_\mathcal{S}, \mu_\mathcal{W}$ be replacement maps, let π be an argument filter, let $\mathcal{U}$ be a set of usable rules, and let $(\succ, \succsim)$ be a complexity pair. If all of the following conditions are satisfied, then $\vdash \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle : G$.*

1. *Whenever $i \notin \pi(f)$, then $\succsim$ ignores the i-th argument of f.*
2. *Both $\mathsf{URM}(\mu_\mathcal{S}, \mathcal{Q}, \mathcal{R}, \mathcal{T}, \mathcal{S})$ and $\mathsf{URM}(\mu_\mathcal{W}, \mathcal{Q}, \mathcal{R}, \mathcal{T}, \mathcal{W})$.*
3. *Whenever $i \in \mu_\mathcal{S}(f)$ ($\mu_\mathcal{W}(f)$), then $\succ$ ($\succsim$) is monotone in the i-th argument of f.*
4. *$\mathcal{S} \cap \mathcal{U} \subseteq \succ$ and $\mathcal{W} \cap \mathcal{U} \subseteq \succsim$.*
5. *If $\ell \to r \in \mathcal{P}$ and $\ell \in \mathcal{T}$ then $\ell \to r \in \mathcal{U}$.*
6. *$\mathcal{U}$ is closed under right-hand sides of usable rules w.r.t. $\mathcal{R}$ for both $\mu_\mathcal{S}$ and $\pi \cap \mu_\mathcal{W}$.*
7. *$\vdash \langle \succ/\varnothing, \varnothing, \mathcal{T}\rangle : G$*

In the theorem, we have two replacement maps $\mu_\mathcal{S}$ and $\mu_\mathcal{W}$ for the strict and weak rules as in [1, Thm. 14.10], but additionally there is the usual argument filter π indicating ignored argument positions of $\succsim$ which is used to reduce the set of usable rules. Let us shortly walk through all conditions of the theorem.

1. π is the standard argument filter as known from termination proofs via reduction pairs, e.g., if $[\mathsf{f}(x_1, x_2, x_3)] = 2x_2 + \frac{1}{2}x_3$, then $\pi(\mathsf{f}) = \{2, 3\}$.
2. Both $\mu_\mathcal{S}$ and $\mu_\mathcal{W}$ are estimated usable replacement maps, which can be computed by one of the methods above, where especially [1, Lemma 14.34] is often only applicable to generate $\mu_\mathcal{S}$.
3. The maps $\mu_\mathcal{S}$ and $\mu_\mathcal{W}$ indicate at which positions redexes may occur, and hence the corresponding orders $\succ$ and $\succsim$ must be monotone w.r.t. these positions.
4. Only usable rules have to be oriented by the complexity pair.
5. In the generation of usable rules, one starts to include all rules which have basic terms on their left-hand sides
6. and then performs the closure of usable rules w.r.t. an argument filter as in [16].
7. Finally, one extracts the derivation bound from the strict order $\succ$, and eventually derives the same bound for the input complexity problem.

We included this theorem into CeTA, where in the certificate just the complexity pair and the usable rules have to be provided, in combination with the strict rules for the split processor of Theorem 2. Since currently IsaFoR only has an interface for reduction pairs the latter condition in 3 does not have to be checked at runtime. All other information will be automatically inferred. To this end, we had to modify our interface of reduction pairs which now has to provide means of querying monotonicity of $\succ$ w.r.t. specific positions.

Using this theorem, CeTA could now certify most combinations of applying a complexity pair with usable rules and/or usable replacement maps in our experiments. Possible improvements at this point are the inclusion of better estimations of usable replacement maps, and better support for the complexity pairs itself, e.g., by removing the restriction to upper triangular matrix interpretations.

7 Experiments

We have tested our new formalization in combination with the only two complexity tools that apply several of the methods described in this paper: AProVE [8] (version 2015.01) and TcT [2] (version 2.2). Both were run on the *termination problem data base*, version 9.0.2,[4]

[4] The TPDB is available at `http://termcomp.uibk.ac.at/`.

	Full Rewriting			Innermost Rewriting				
	TcT			TcT			AProVE	
	certification		full	certification		full	certification	full
	new	old		new	old		new	
constant	0	0	18	0	0	38	1	53
linear	134	67	182	234	117	278	159	249
quadratic	165	107	201	291	157	341	250	350
cubic	165	110	202	299	160	354	283	387
polynomial	165	110	203	301	160	361	283	387

Table 1 Experimental Results

which was also used for the complexity category of the FLoC Olympic Games of 2014. All tests were conducted on a machine with 8 dual core AMD Opteron™ 885 processors running at 2.60 GHz on 64 Gb of RAM and within a timeout of 60 seconds per test.

Table 1 collects our experimental findings.Here we show totals on estimated upper bounds (from constant to polynomial of unknown degree) on runtime complexities w.r.t. full and innermost rewriting, the former being only supported by TcT. To delineate the extend of our new formalization, we have compared the tools when run in various modes:

- In *certification mode* (columns *certification new*) we restrict tools to those methods that can also be certified by CeTA version 2.19. We contrast this data with results obtained from the version of TcT that ran in certification mode at the recent termination competition (columns *certification old*). Note that until now, AProVE did not feature certification support, consequently respective results are not present in the table.
- In *full mode* (columns *full*) we show totals when tools are run in their default setting, i.e., possibly employing methods that cannot be certified by CeTA.

Overall, the experiments confirm significant improvements of CeTA's support for complexity analysis. For instance with TcT we certified polynomially bounded innermost runtime complexity of 301 systems. This corresponds to 83 % of the systems that can be handled by TcT when run in full mode. In contrast, relying on our old formalization TcT could handle only 44 % of the systems. The statement remains essentially correct for AProVE and TcT w.r.t. full rewriting.

Even more important might have been our preliminary experiments, where several proofs have been rejected by CeTA. Although the reason have often just been bugs in the proof-output of the tools, we also revealed and fixed (or at least reported to the developers) some more severe problems: one tool modified the sets $\mathcal{D}$ and $\mathcal{C}$ in the set of starting terms $\mathcal{T} = \mathsf{BT}(\mathcal{D}, \mathcal{C}, \mathcal{V})$ when deleting rules by the usable rules processor in a way that made the tool unnecessarily weak (and unsound for lower complexity bounds); one tool had a bug when computing usable rules which could be exploited to generate linear derivation bounds for non-terminating TRSs; and also some match-bounds certificates have been rejected where the corresponding code had to be disabled. Finally, also the required adaptation of $\mathcal{Q}$ to $\mathcal{Q}' \subseteq \mathcal{Q} \cup \mathcal{Q}_{\neg\mathcal{F}}$, as discussed in Section 5, was only detected by earlier versions of CeTA which did not support this possibility.

Conclusion. We presented our formalization of several techniques for complexity analysis that are now part of the formal library IsaFoR: match-bounds, weak dependency pairs, dependency tuples, usable rules, and usable replacement maps. Moreover, we reported on the

resulting increase in power of our certifier CeTA, which is now able to certify more than three quarters of all complexity proofs that are generated by state-of-the-art tools.

Acknowledgments. This work was partially supported by Austrian Science Fund (FWF) projects Y757, J3563, and P27502. The authors are listed in alphabetical order regardless of individual contributions or seniority. We thank the anonymous reviewers for their helpful remarks.

References

1 Martin Avanzini. *Verifying Polytime Computability Automatically.* PhD thesis, University of Innsbruck, 2013. `http://cl-informatik.uibk.ac.at/~zini/publications/diss.pdf`.

2 Martin Avanzini and Georg Moser. A combination framework for complexity. In *Proc. 24th International Conference on Rewriting Techniques and Applications*, volume 21 of *Leibniz International Proceedings in Informatics*, pages 55–70, 2013. 10.4230/LIPIcs.RTA.2013.55.

3 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

4 Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979. 10.1145/359138.359142.

5 Bertram Felgenhauer and René Thiemann. Reachability analysis with state-compatible automata. In *Proc. 8th International Conference on Language and Automata Theory and Applications*, volume 8370 of *Lecture Notes in Computer Science*, pages 347–359, 2014. 10.1007/978-3-319-04921-2_28.

6 Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability analysis over term rewriting systems. *Journal of Automated Reasoning*, 33:341–383, 2004. 10.1007/s10817-004-6246-0.

7 Alfons Geser, Dieter Hofbauer, Johannes Waldmann, and Hans Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Information and Computation*, 205(4):512–534, 2007. 10.1016/j.ic.2006.08.007.

8 Jürgen Giesl, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Proving termination of programs automatically with AProVE. In *Proc. 7th International Joint Conference on Automated Reasoning*, volume 8562 of *Lecture Notes in Computer Science*, pages 184–191, 2014. 10.1007/978-3-319-08587-6_13.

9 Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331, 2005. 10.1007/978-3-540-32275-7_21.

10 Nao Hirokawa and Georg Moser. Automated complexity analysis based on the dependency pair method. In *Proc. 4th International Joint Conference on Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 364–379, 2008. 10.1007/978-3-540-71070-7_32.

11 Nao Hirokawa and Georg Moser. Automated complexity analysis based on context-sensitive rewriting. In *Proc. Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications*, volume 8560 of *Lecture Notes in Computer Science*, pages 257–271, 2014. 10.1007/978-3-319-08918-8_18.

12 Martin Korp and Aart Middeldorp. Match-bounds revisited. *Information and Computation*, 207(11):1259–1283, 2009. 10.1016/j.ic.2009.02.010.

13 Georg Moser and Andreas Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3:1):1–38, 2011. 10.2168/LMCS-7(3:1)2011.

14 Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. 10.1007/3-540-45949-9.

15 Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *Journal of Automated Reasoning*, 51(1):27–56, 2013. 10.1007/s10817-013-9277-6.

16 Christian Sternagel and René Thiemann. Certified subterm criterion and certified usable rules. In *Proc. 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 325–340, 2010. 10.4230/LIPIcs.RTA.2010.325.

17 Christian Sternagel and René Thiemann. Formalizing monotone algebras for certification of termination and complexity proofs. In *Proc. Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications*, volume 8560 of *Lecture Notes in Computer Science*, pages 441–455, 2014. 10.1007/978-3-319-08918-8_30.

18 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *Proc. 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468, 2009. 10.1007/978-3-642-03359-9_31.

19 Harald Zankl and Martin Korp. Modular complexity analysis via relative complexity. In *Proc. 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 385–400, 2010. 10.4230/LIPIcs.RTA.2010.385.

20 Harald Zankl and Martin Korp. Modular complexity analysis for term rewriting. *Logical Methods in Computer Science*, 10(1:19):1–34, 2014. 10.2168/LMCS-10(1:19)2014.