Formalizing Bounded Increase[⋆]

René Thiemann

Institute of Computer Science, University of Innsbruck, Austria
rene.thiemann@uibk.ac.at

Abstract. Bounded increase is a termination technique where it is tried to find an argument x of a recursive function that is increased repeatedly until it reaches a bound b, which might be ensured by a condition $x < b$. Since the predicates like $<$ may be arbitrary user-defined recursive functions, an induction calculus is utilized to prove conditional constraints.

In this paper, we present a full formalization of bounded increase in the theorem prover Isabelle/HOL. It fills one large gap in the pen-and-paper proof, and it includes generalized inference rules for the induction calculus as well as variants of the Babylonian algorithm to compute square roots. These algorithms were required to write executable functions which can certify untrusted termination proofs from termination tools that make use of bounded increase. And indeed, the resulting certifier was already useful: it detected an implementation error that remained undetected since 2007.

1 Introduction

A standard approach to proving termination of recursive programs is to find a well-founded order $\succ$, such that every recursive call *decreases* w.r.t. this order: whenever $f(\ell_1,\ldots,\ell_n) = C[f(r_1,\ldots,r_n)]$ is a defining equation for f, then $(\ell_1,\ldots,\ell_n) \succ (r_1,\ldots,r_n)$ has to hold. For example, one can use a measure function m which maps values into the naturals where $(\ell_1,\ldots,\ell_n) \succ (r_1,\ldots,r_n)$ is defined as $m(\ell_1,\ldots,\ell_n) > m(r_1,\ldots,r_n)$. For this approach to be sound, one uses well-foundedness of the $>$-order on the naturals.

However, often termination can also be concluded since some argument is *increased* until it exceeds a bound, as demonstrated in the following algorithm which computes $\sum_{i=0}^{n} i$.

$$\begin{aligned} \mathit{compute\text{-}sum}\ n &= \mathit{sum}\ 0\ n \\ \mathit{sum}\ i\ n &= \mathsf{if}\ i \leq n\ \mathsf{then}\ i + \mathit{sum}\ (i+1)\ n\ \mathsf{else}\ 0 \end{aligned}$$

Also here an appropriate measure can be chosen which is decreased in every iteration, e.g. the difference of the parameters: $m(i,n) = n - i$. However, one needs more information to conclude termination: the measure maps into the integers (and not into the naturals), where the $>$-order is not well-founded.

[⋆] Supported by the Austrian Science Fund (FWF), project P22767.

Only in combination with the condition $i \leq n$ one can conclude that the value $n - i$ is bounded from below and thus, termination of the function.[1]

In the remainder of the paper, we are focussing on term rewrite systems (TRSs), a simple yet powerful computational model that underlies much of declarative programming and theorem proving—for example, both Haskell-functions and Isabelle/HOL-functions can be translated into TRSs such that termination of the TRSs implies termination of the functions [7,15]. For TRSs, the above kind of termination argument has been introduced as the termination technique *bounded increase* [9], where measure functions like $m(i, n) = n - i$ are allowed in the form of polynomial orders [16].

To improve the reliability of automated termination tools for TRSs like AProVE [8] and $\mathsf{T_TT_2}$ [13], we already formalized several termination techniques (IsaFoR) which resulted in a certifier (CeTA) which validates or invalidates untrusted termination proofs [20]. Other certifiers are based on the alternative formalizations Coccinelle [6] and CoLoR [5]. Due to the certifiers, bugs have been detected in the automated termination tools, which remained without notice for several years and could easily be fixed after their detection.

However, CeTA can only certify those proofs where all applied termination techniques have been formalized. And in order to achieve a high coverage, the integration of bounded increase is essential. For example, in the latest international termination competition in 2012, two versions of AProVE participated: one unrestricted version, and one which only uses techniques that are supported by CeTA; a detailed inspection of the proofs revealed that for more than half of the termination proofs where only the unrestricted version was successful, the method of bounded increase was applied.

Although the termination argument behind bounded increase looks quite intuitive, we want to stress that the underlying soundness proofs are far from being trivial. To this end, consider the following TRS which is a reformulation of the previous algorithm as TRS. We list three problems that are solved in [9].

$$\begin{aligned}
\mathsf{compute\text{-}sum}(n) &\to \mathsf{sum}(\mathsf{0}, n)\\
\mathsf{sum}(i, n) &\to \mathsf{if_{sum}}(i \leq n, i, n)\\
\mathsf{if_{sum}}(\mathsf{true}, i, n) &\to i + \mathsf{sum}(i + \mathsf{1}, n)\\
\mathsf{if_{sum}}(\mathsf{false}, i, n) &\to \mathsf{0}
\end{aligned}$$

1. Since interpretations are used which interpreted into the integers, for strong normalization one needs to know that there is a bound on the integers, which is obtained from analyzing side conditions.
2. Usually, for termination analysis of TRSs one uses orders which are weakly monotone w.r.t. contexts, i.e., whenever $\ell \succsim r$ then $C[\ell] \succsim C[r]$. However, polynomials like $n - i$ result in non-monotone orders $\succsim$.

[1] Alternatively, one can define $m(i, n) = n + 1 \dot{-} i$ which maps into the naturals using the truncating subtraction $\dot{-}$. But again one requires the condition $i \leq n$ for the termination proof, namely to ensure $m(i, n) > m(i + 1, n)$.

3. Since there is no builtin arithmetic for pure term rewriting, one adds rewrite rules which compute $\leq$, $+$, etc., like the following ones in $\mathcal{R}_{\mathsf{arith}}$:

$$\begin{aligned} \mathsf{0} \leq y &\to \mathsf{true} & \mathsf{0} + y &\to y \\ \mathsf{s}(x) \leq \mathsf{0} &\to \mathsf{false} & \mathsf{s}(x) + y &\to \mathsf{s}(x + y) \\ \mathsf{s}(x) \leq \mathsf{s}(y) &\to x \leq y \end{aligned}$$

Concrete numbers n in $\mathcal{R}_{\mathsf{sum}}$ are directly replaced by $\mathsf{s}^n(\mathsf{0})$ where s and $\mathsf{0}$ are the constructors for natural numbers.

As a consequence, conditions like $i \leq n$ are not arithmetic constraints, but rewrite constraints $i \leq n \to_{\mathcal{R}}^* \mathsf{true}$ where $\mathcal{R}$ is a TRS which contains the rules of $\mathcal{R}_{\mathsf{arith}}$. As a result, one has to be able to solve conditional constraints of the form $t_1 \to_{\mathcal{R}}^* t_2 \longrightarrow t_3 \succ t_4$.

The paper is structured as follows: after the preliminaries in Sect. 2, we shortly recapitulate the solutions in [9] to all three problems in Sect. 3–5 and present their formalization. Here, major gaps in the original proofs are revealed, and the existing results are generalized. In Sect. 6 we illustrate complications that occurred when trying to extend our certifier towards bounded increase, which also forced us to formalize a precise algorithm to compute square roots.

2 Preliminaries

We briefly recall some basic notions of term rewriting [2].

The set of (first-order) terms over some signature $\mathcal{F}$ and variables $\mathcal{V}$ is written as $\mathcal{T}(\mathcal{F}, \mathcal{V})$. A context C is a term with one hole $\Box$, and $C[t]$ is the term where $\Box$ is replaced by t. The term t is a subterm of s iff $s = C[t]$ for some C, and it is a proper subterm if additionally $s \neq t$. A substitution σ is a mapping from variables to terms. It is extended homomorphically to terms where we write $t\sigma$ for the application of σ on t. A TRS is a set of rules $\ell \to r$ for terms ℓ and r. The rewrite relation of a TRS $\mathcal{R}$ is defined as $s \to_{\mathcal{R}} t$ iff $s = C[\ell\sigma]$ and $t = C[r\sigma]$ for some $\ell \to r \in \mathcal{R}$, σ, and C. By $\mathit{NF}(\mathcal{R})$ we denote the normal forms of $\mathcal{R}$, i.e., those terms s where there is no t such that $s \to_{\mathcal{R}} t$. A substitution σ is *normal w.r.t.* $\mathcal{R}$ iff $\sigma(x) \in \mathit{NF}(\mathcal{R})$ for all x. A symbol f is *defined* w.r.t. $\mathcal{R}$ iff there is some rule $f(\dots) \to r \in \mathcal{R}$. The remaining symbols are called *constructors* of $\mathcal{R}$.

Throughout this paper we are interested in the innermost rewrite relation $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$ which is defined like $\to_{\mathcal{R}}$ with the additional requirement that all proper subterms of $\ell\sigma$ must be normal forms w.r.t. $\mathcal{R}$.[2] We write $\mathit{SIN}_{\mathcal{R}}(t)$ to indicate

[2] In IsaFoR we consider a more general definition of innermost rewriting, where the proper subterms of $\ell\sigma$ must be normal forms w.r.t. some TRS $\mathcal{Q}$ which is independent from $\mathcal{R}$. All results in this paper have been proven for this generalized notion of rewriting under the condition that $\mathit{NF}(\mathcal{Q}) \subseteq \mathit{NF}(\mathcal{R})$. This generalization was important, as it is also utilized in the termination proofs that are generated by AProVE. Moreover, the generalized innermost rewrite relation has the advantage that it is monotone w.r.t. $\mathcal{R}$. However, to improve the readability, in this paper we just use $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$, i.e., we fix $\mathcal{Q} = \mathcal{R}$.

that the term t is strongly normalizing w.r.t. the innermost rewrite relation, i.e., there is no infinite derivation $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathsf{i}}_{\mathcal{R}} \dots$, and $\mathit{SIN}(\mathcal{R})$ denotes innermost termination of $\mathcal{R}$, i.e., strong normalization of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.

A popular way to prove innermost termination of TRSs is to use dependency pairs [1,10], which capture all calls of a TRS.

Example 1. Let $\mathcal{R} := \mathcal{R}_{\mathsf{sum}} \cup \mathcal{R}_{\mathsf{arith}}$ where additionally the number 1 in $\mathcal{R}_{\mathsf{sum}}$ has been replaced by $\mathsf{s}(0)$. Then the following set $\mathcal{P}$ contains those dependency pairs of $\mathcal{R}$ which correspond to recursive calls.

$$\mathsf{sum}^\sharp(i,n) \to \mathsf{if}^\sharp_{\mathsf{sum}}(i \le n, i, n) \quad (1) \qquad \mathsf{s}(x) \le^\sharp \mathsf{s}(y) \to x \le^\sharp y \quad (3)$$

$$\mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, i, n) \to \mathsf{sum}^\sharp(i + \mathsf{s}(0), n) \quad (2) \qquad \mathsf{s}(x) +^\sharp y \to x +^\sharp y \quad (4)$$

Here, the sharped-symbols are fresh *tuple*-symbols ($\mathcal{F}^\sharp$) which can be treated differently from their defined counterparts when parametrizing orders, etc.

The major theorem of dependency pairs tells us for Ex. 1 that $\mathcal{R}$ is innermost terminating iff there is no (minimal) innermost $(\mathcal{P},\mathcal{R})$-chain. Here, a chain is an infinite derivation of the form

$$s_1\sigma_1 \to_{\mathcal{P}} t_1\sigma_1 \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s_2\sigma_2 \to_{\mathcal{P}} t_2\sigma_2 \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} \dots \qquad (\star)$$

where all $s_i \to t_i \in \mathcal{P}$ and all $s_i\sigma_i \in \mathit{NF}(\mathcal{R})$. The chain is minimal if $\mathit{SIN}_{\mathcal{R}}(t_i\sigma_i)$ for all i. Intuitively, the step from $s_i\sigma$ to $t_i\sigma$ corresponds to a recursive call, and the step from $t_i\sigma$ to $s_{i+1}\sigma_{i+1}$ evaluates the arguments before the next call.

In the remainder of this paper we assume some fixed TRS $\mathcal{R}$ and in examples we always use the TRS $\mathcal{R}$ of Ex. 1. Consequently, in the following we use the notion of $\mathcal{P}$-chain instead of $(\mathcal{P},\mathcal{R})$-chain where $\mathcal{P}$ is some set of (dependency) pairs $s \to t$. We call $\mathcal{P}$ *terminating* iff there is no minimal $\mathcal{P}$-chain.[3]

One major technique to show termination of $\mathcal{P}$ is the reduction pair processor [10] which removes pairs from $\mathcal{P}$—and termination can eventually be concluded if all pairs have been removed. This technique has been formalized for all certifiers [5,6,20].

Theorem 2. *Let* $(\succsim, \succ)$ *be a* reduction pair, *i.e.,* $\succ$ *is an order which is strongly normalizing,* $\succsim$ *is a quasi-order satisfying* $\succsim \circ \succ \circ \succsim \subseteq \succ$, *both* $\succ$ *and* $\succsim$ *are stable (closed under substitutions), and* $\succsim$ *is monotone (closed under contexts). A set of pairs* $\mathcal{P}$ *is terminating if* $\mathcal{P} \subseteq \succ \cup \succsim$, $\mathcal{R} \subseteq \succsim$, *and* $\mathcal{P} \setminus \succ$ *is terminating.*

One instance of reduction pairs stems from polynomial interpretations over the naturals [16] where every symbol is interpreted by a polynomial over the naturals, and $s \succsim t$ is defined as $[\![s]\!] \ge [\![t]\!]$ (similarly, $s \succ t$ iff $[\![s]\!] > [\![t]\!]$).

[3] In the literature termination of $\mathcal{P}$ is called finiteness of $(\mathcal{P},\mathcal{R})$. We use the notion of termination here, so that "$\mathcal{P}$ is finite" does not have two meanings: finiteness of the set $\mathcal{P}$ or absence of $\mathcal{P}$-chains. In IsaFoR, the latter property is called *finite-dpp*, but as for the innermost rewrite relation, IsaFoR uses a more general notion of chain, which is essential for other termination techniques. To be more precise we do not only consider the 3 components ($\mathcal{P}$, $\mathcal{R}$, and a minimality flag) to define chains, but 7: strict and weak pairs, strict and weak rules, strategy, minimality flag, and a flag to indicate whether substitutions have to return normal forms on free variables.

3 First Problem: No Strong Normalization

Since for bounded increase interpretations into the integers are used where $>$ is not strongly normalizing, the reduction pair processor has to be adapted, as the resulting strict order $\succ$ is not strongly normalizing [9]. Instead, it is required that $\succ$ is *non-infinitesimal*, which is defined as absence of infinite sequences $t_1 \succ t_2 \succ \dots$ which additionally satisfy $t_i \succ \mathsf{c}$ for all i where c is some constant. However, weakening strong normalization to being non-infinitesimal requires to strengthen other preconditions in the reduction pair processor: in detail, termination of $\mathcal{P} \setminus \succ$ does no longer suffice, but additionally one requires that termination is ensured if all bounded pairs are removed, i.e., termination of $\mathcal{P} \setminus \{s \to t \mid s \succsim \mathsf{c}\}$.

Another adaptation is required when trying to solve term constraints, which is a distinction between symbols of $\mathcal{F}$ and tuple symbols. It is easily motivated: if every symbol is mapped to an integer, then polynomial constraints like $2x \geq x$ are no longer satisfied, since x might be instantiated by some negative value. However, $2x \geq x$ holds over the naturals. Since one wants to be able to conclude $2x \geq x$, one uses interpretations where all $\mathcal{F}$-symbols are interpreted by naturals, and only the tuple symbols in $\mathcal{F}^\sharp$ may interpret into the integers. As a consequence, $2x \geq x$ can be concluded, but $\succ$ and $\succsim$ are no longer stable, since we only get closure under those substitutions which instantiate all variables by terms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$—this property is called $\mathcal{F}$-stable and these substitutions are called $\mathcal{F}$-substitutions.

Of course, if one wants to use $\mathcal{F}$-stable orders $\succsim$ and $\succ$ in the reduction pair processor, one has to know that all substitutions σ_i in a chain ($\star$) can be chosen as $\mathcal{F}$-substitutions, a property which is named signature extension: if we can prove termination if we only consider $\mathcal{F}$-substitutions, then we can prove termination for arbitrary larger signatures like $\mathcal{F} \cup \mathcal{F}^\sharp$. In [9, proof of Thm. 11, technical report] signature extensions are taken for granted ("clearly"), most likely since signature extensions are possible for (innermost) termination analysis on the level of TRSs. However, in [17] we proved that signature extensions are unsound on the level of minimal chains. Luckily, we were able to formalize that for (minimal) innermost chains, signature extensions are indeed sound. The efforts to get this result is in stark contrast to "clearly": $\approx$ 900 lines, cf. the locale `cleaning-innermost` within the theory `Signature-Extension.thy`.

Concerning the restriction to $\mathcal{F}$-stable orders, in the formalization we also added a new feature which is not present in [9]: the restriction to $\mathcal{F}$-monotone orders. For example, if we interpret $[\![f]\!](x) = x^2$, then the resulting order $\succsim$ is not monotone, as the arguments range over all integers. For $\mathcal{F}$-monotone orders it is instead only required that these are monotone if all arguments are terms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$, i.e., in the case of polynomial orders, if the arguments evaluate to natural numbers. Hence, the interpretation $[\![f]\!](x) = x^2$ is $\mathcal{F}$-monotone.

Since we need to prove termination of $\mathcal{P} \setminus \{s \to t \mid s \succsim \mathsf{c}\}$, it is essential that at least one of the pairs is bounded. However, if we use an interpretation like $[\![\mathsf{sum}^\sharp]\!](i, n) = [\![\mathsf{if}^\sharp_{\mathsf{sum}}]\!](b, i, n) = n - i$, then none of the pairs (1) and (2) is bounded. So, we will not make progress using this interpretation without further adaptations. To this end, conditions are added where for every pair in a chain, one obtains the conditions from the adjacent pairs in the chain: for example, when considering the preceding rewrite steps of some pair $s \to t$, instead of demanding $s \succsim \mathsf{c}$ for boundedness, one can demand that all implications $v \xrightarrow{\mathsf{i}}{}^* s \longrightarrow s \succsim \mathsf{c}$ are satisfied for every variable renamed pair $u \to v \in \mathcal{P}$ and every instantiation of the variables with normal forms.

More formally, we consider conditional constraints which are first-order formulas using atomic formulas of the form $s \xrightarrow{\mathsf{i}}{}^* t$, $s \succsim t$, and $s \succ t$. The satisfaction relation for normal $\mathcal{F}$-substitutions σ (written $\sigma \models \phi$) is defined as follows.

$$\sigma \models \phi \text{ iff } \begin{cases} s\sigma \succsim t\sigma, & \text{if } \phi = s \succsim t \\ s\sigma \succ t\sigma, & \text{if } \phi = s \succ t \\ s\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t\sigma \wedge \mathit{NF}(t\sigma) \wedge \mathit{SIN}(s\sigma), & \text{if } \phi = s \xrightarrow{\mathsf{i}}{}^* t \end{cases}$$

The other logic connectives $\longrightarrow$, $\wedge$, $\forall$ are treated as usual, and $\models \phi$ is defined as $\sigma \models \phi$ for all normal $\mathcal{F}$-substitutions σ. Using these definitions, a constraint like $s \succsim \mathsf{c}$ can be replaced by $\bigwedge u\ v.\ u \to v \in \mathcal{P} \implies \models v \xrightarrow{\mathsf{i}}{}^* s \longrightarrow s \succsim \mathsf{c}$.

The most tedious part when integrating this adaptation of conditional constraints into the formalization of the reduction pair processor was the treatment of variable renamed pairs: in ($\star$) we use several different substitutions of non-renamed pairs, and we had to show that instead one can use one substitution where all pairs are variable renamed. And of course, when certifying termination proofs, one again has to work modulo variable names, since one does not know in advance how the variables have been renamed apart in the termination tool.

4 Second Problem: No Monotonicity

Monotonicity and stability are important to ensure the implication $\mathcal{R} \subseteq \succsim \implies \xrightarrow{\mathsf{i}}_{\mathcal{R}} \subseteq \succsim$. Using this implication with the requirement $\mathcal{R} \subseteq \succsim$ in the reduction pair processor allows to replace all $\xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}}$ in ($\star$) by $\succsim$.

For innermost rewriting, the requirement that all rules have to be *weakly decreasing* ($\mathcal{R} \subseteq \succsim$ or equivalently, $\ell \succsim r$ for all $\ell \to r \in \mathcal{R}$) can be reduced to demand a weak decrease of only the *usable rules*. Here, the usable rules of a term t are those rules of $\mathcal{R}$ which can be applied when reducing $t\sigma$ at those positions which are relevant w.r.t. the order $\succsim$; and the usable rules of $\mathcal{P}$ are the usable rules of all right-hand sides of $\mathcal{P}$. We omit a formal definition (e.g., [10, Def. 21]) here, but just illustrate the usage in our running example.

Example 3. We use the reduction pair processor (without the adaptations from the previous section) in combination with the polynomial order $[\![\mathsf{sum}^\sharp]\!](i,n) = [\![\mathsf{if}^\sharp_{\mathsf{sum}}]\!](b,i,n) = 0$ and $[\![+^\sharp]\!](x,y) = [\![\leq^\sharp]\!](x,y) = [\![\mathsf{s}]\!](x) = 1 + x$.

The only rules that are required to evaluate the arguments of right-hand sides of $\mathcal{P} = \{(1)\text{–}(4)\}$ are the $\leq$-rules for (1) and the $+$-rules for (2). However, since the first arguments of $\mathsf{sum}^\sharp$ and $\mathsf{if}^\sharp_{\mathsf{sum}}$ are ignored by the polynomial order, there are no usable rules. Since moreover, all pairs are at least weakly decreasing, we can delete the strictly decreasing pairs (3) and (4) by the reduction pair processor. Hence, it remains to prove termination of $\mathcal{P} \setminus \succ = \{(1), (2)\}$.

From the soundness result for usable rules [10, Lem. 23] one derives that whenever $t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s$ for some normal substitution σ and the usable rules of t are weakly decreasing, then $t\sigma \succsim s$. Unfortunately, this lemma does no longer hold if $\succsim$ is not monotone. To solve this problem, in [9] a monotonicity function $\mathit{ord} :: \mathcal{F} \cup \mathcal{F}^\sharp \Rightarrow \mathbb{N} \Rightarrow \{0, 1, -1, 2\}$[4] is defined. It determines for each symbol f and each argument i, which relationship between s_i and t_i must be ensured to guarantee $C[s_i] \succsim C[t_i]$ for some context $C = f(u_1, \dots, u_{i-1}, \square, u_{i+1}, \dots, u_n)$:

- if there are no requirements on s_i and t_i then $\mathit{ord}(f, i)$ is defined as 0,
- if $s_i \succsim t_i$ is required then $\succsim$ is monotonically increasing in the i-th argument and $\mathit{ord}(f, i)$ is defined as 1,
- if $t_i \succsim s_i$ is required then $\succsim$ is monotonically decreasing in the i-th argument and $\mathit{ord}(f, i)$ is defined as -1,
- otherwise, $\mathit{ord}(f, i)$ is defined as 2 (and it is demanded that $s_i \succsim t_i \wedge t_i \succsim s_i$ suffices to ensure $C[s_i] \succsim C[t_i]$).

Using *ord* the generalized usable rules are defined as follows.

Definition 4. *Let $\mathcal{R}$ be some TRS. The generalized usable rules of term t are defined as the least set $\mathcal{U}(t)$ such that*

- *whenever $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}$, then $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{U}(f(t_1, \dots, t_n))$,*
- *whenever $\ell \to r \in \mathcal{U}(t)$ then $\mathcal{U}(r) \subseteq \mathcal{U}(t)$, and*
- *$\mathcal{U}(t_i)^{\mathit{ord}(f,i)} \subseteq \mathcal{U}(f(t_1, \dots, t_n))$.*

Here, $U^0 = \emptyset$, $U^1 = U$, $U^{-1} = \{r \to \ell \mid \ell \to r \in U\}$, and $U^2 = U \cup U^{-1}$.
The generalized usable rules of a set of pairs $\mathcal{P}$ is $\mathcal{U}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t)$.

Example 5. In Ex. 3 we remained with the pairs $\{(1), (2)\}$. If we choose the polynomial order defined by $[\![\mathsf{sum}^\sharp]\!](i, n) = [\![\mathsf{if}^\sharp_{\mathsf{sum}}]\!](b, i, n) = n - i$, $[\![+]\!](x, y) = x + y$, $[\![\mathsf{0}]\!] = 0$, and $[\![\mathsf{s}]\!](x) = 1 + x$, then $\mathcal{U}(\{(1), (2)\}) = \mathcal{U}(\mathsf{if}^\sharp_{\mathsf{sum}}(i \le n, i, n)) \cup \mathcal{U}(\mathsf{sum}^\sharp(i + \mathsf{s}(\mathsf{0}), n))$ are the two reversed $+$-rules $y \to \mathsf{0} + y$ and $\mathsf{s}(x + y) \to \mathsf{s}(x) + y$ which are both decreasing by this polynomial order, i.e., $\mathcal{U}(\{(1), (2)\}) \subseteq \succsim$.

The results on the generalized usable rules and usable rules are similar.

Lemma 6. *Whenever $t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s$ for some normal substitution σ, then $\mathcal{U}(t) \subseteq \succsim$ implies $t\sigma \succsim s$.*

Concerning the formalization of this result, we only added one indirection: Instead of directly defining $\mathcal{U}$ using *ord*, we define $\mathcal{U}_\pi$ like $\mathcal{U}$ where *ord* is replaced by some user defined function $\pi :: \mathcal{F} \cup \mathcal{F}^\sharp \Rightarrow \mathbb{N} \Rightarrow \{0, 1, -1, 2\}$ and then demand that π and $\succsim$ are *compatible*: Whenever $\pi(f, i) = k$ and $1 \le i \le \text{arity}(f)$, then $\succsim$ must satisfy the monotonicity condition which is required for $\mathit{ord}(f, i) = k$.

The reason is that usually, we can only approximate $\mathit{ord}(f, i)$, but not decide it. For example, for a polynomial interpretation over the naturals with $[\![f]\!](x) = x^2 - x$ we have $\mathit{ord}(f, 1) = 1$, since $x^2 - x$ is monotonically increasing over

[4] In IsaFoR, we use a datatype to indicate the monotonicity instead of $\{0, 1, -1, 2\}$. It consists of the elements `Ignore` (0), `Increase` (1), `Decrease` (-1), and `Wild` (2).

the naturals. However, we also allow to define $\pi(f,1) = 2 \neq \mathit{ord}(f,i)$ with the consequences, that there are more usable rules, but weaker requirements for the monotonicity check.

Using this adaptation we proved Lem. 6 where $\mathcal{U}$ is replaced by $\mathcal{U}_\pi$ and where compatibility of π and $\succsim$ is added as additional assumption. Even with these changes, we could show the result by minor adaptations of the original proof.

However, it turns out that Lem. 6 is not really helpful in the overall approach of bounded increase as it is assumes that $\succsim$ is stable. But from the last section we know that we are more interested in relations $\succsim$ which are only $\mathcal{F}$-stable. And then Lem. 6 does no longer hold, even if we restrict σ to be an $\mathcal{F}$-substitution. To still achieve a result similar to Lem. 6 for $\mathcal{F}$-stable $\succsim$, we could additionally demand that $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$. Although this would be sound, it is still not useful, as we require the result for terms t like $\mathsf{sum}^\sharp(i + \mathsf{s}(\mathsf{0}), n)$ where the root symbol is a tuple symbol that is not in $\mathcal{F}$.

Hence, we formalized the following lemma where we also improved the definition of *ord* (or compatibility) by adding ideas from $\mathcal{F}$-monotonicity: all terms s_i, t_i, and u_i that are occurring in the definition of *ord* (or compatibility) are only quantified over $\mathcal{T}(\mathcal{F},\mathcal{V})$. Note that every tuple symbol is a constructor.

Lemma 7. *If $\mathcal{R}$ is a TRS over signature $\mathcal{F}$, π and $\succsim$ are compatible, σ is a normal $\mathcal{F}$-substitution, $t = f(t_1,\ldots,t_n) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s$, f is a constructor of $\mathcal{R}$, and all $t_i \in \mathcal{T}(\mathcal{F},\mathcal{V})$, then $\mathcal{U}_\pi(t) \subseteq \succsim \implies t\sigma \succsim s$.*

The proof of this result was an extended version of the proof for Lem. 6 where the following main property was shown by induction on t for arbitrary k and s.

Lemma 8. *If $\mathcal{R}$ is a TRS over signature $\mathcal{F}$, π and $\succsim$ are compatible, σ is a normal substitution, $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} s$, $t\sigma \in \mathcal{T}(\mathcal{F},\mathcal{V})$ or $t\sigma = f(t_1,\ldots,t_n)$ with constructor f of $\mathcal{R}$ and all $t_i \in \mathcal{T}(\mathcal{F},\mathcal{V})$, then $\mathcal{U}_\pi(t)^k \subseteq \succsim \implies \{(t\sigma, s)\}^k \subseteq \succsim$ and there are u and a normal substitution δ such that $\mathcal{U}_\pi(u)^k \subseteq \mathcal{U}_\pi(t)^k$ and $s = u\delta$.*

Using these results we are ready to present our formalized result of the adapted, conditional general reduction pair processor which combines both modifications—where $\succ$ does not have to be strongly normalizing and $\succsim$ does not have to be monotone.[5]

Theorem 9. *Let c be some constant. Let $\succsim$ and $\succ$ be $\mathcal{F}$-stable orders where $\succsim$ and $\succ$ are compatible, and $\succ$ is non-infinitesimal. Let π be compatible with $\succsim$.[6] Let $\mathcal{P}$, $\mathcal{P}_\succ$, $\mathcal{P}_\succsim$, and $\mathcal{P}_{bound}$ be arbitrary sets of pairs. Then $\mathcal{P}$ is terminating if all of the following properties are satisfied.*

- *$\mathcal{P} \subseteq \mathcal{P}_\succ \cup \mathcal{P}_\succsim$ and $\mathcal{U}_\pi(\mathcal{P}) \subseteq \succsim$*
- *$\mathcal{P} \setminus \mathcal{P}_\succ$ and $\mathcal{P} \setminus \mathcal{P}_{bound}$ is terminating*
- *$\mathcal{R}$ is a TRS over signature $\mathcal{F}$*
- *for all $s \to t \in \mathcal{P}_\succ$ and variable renamed $u \to v \in \mathcal{P}$: $\models v \xrightarrow{\mathsf{i}}{}^* s \longrightarrow s \succ t$*
- *for all $s \to t \in \mathcal{P}_\succsim$ and variable renamed $u \to v \in \mathcal{P}$: $\models v \xrightarrow{\mathsf{i}}{}^* s \longrightarrow s \succsim t$*

[5] In IsaFoR, this processor also integrates the generalization below Thm. 11 in [9], where arbitrary adjacent pairs can be chosen instead of only one preceding pair, cf. lemma `conditional-general-reduction-pair-proc` in `Generalized-Usable-Rules.thy`.

[6] In IsaFoR, all these properties are summarized in the locale `non-inf-order`.

- *for all* $s \to t \in \mathcal{P}_{bound}$ *and variable renamed* $u \to v \in \mathcal{P}$*:* $\models v \xrightarrow{\mathsf{i}}{}^* s \longrightarrow s \succsim \mathsf{c}$
- *all pairs in* $\mathcal{P}$ *are of the form* $f(s_1, \ldots, s_n) \to g(t_1, \ldots, t_m)$ *where* g *is a constructor of* $\mathcal{R}$ *and* $s_i, t_j \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *for all* i *and* j
- *the set of all symbols is countable and* $\mathcal{V}$ *is countably infinite.*

Note that Thm. 9 is similar to [9, Thm. 11] with the difference, that for defining *ord* for the generalized rules we only require $\mathcal{F}$-monotonicity.

The last condition on the cardinality of $\mathcal{F}$ and $\mathcal{V}$ is enforced by our theorem on signature extensions. We believe that it can be weakened to $\mathcal{V}$ being infinite, but this would make the proof for signature extensions more tedious.

Even after having proven Thm. 9, there remain two tasks.

First, we actually require an executable function which computes the set of usable rules (which is defined as an inductive set) for some concrete TRS $\mathcal{R}$ and set of pairs $\mathcal{P}$. To this end, we have not been able to use Isabelle's predicate compiler [3], but instead we used an approach as in [18]: We characterized the generalized usable rules via a reflexive transitive closure. And afterwards, we could just invoke a generic algorithm to compute reflexive transitive closures as the one in [19]. More details can be seen in `Generalized-Usable-Rules-Impl.thy`.

The second task is to deal with conditional constraints that arise from the conditional reduction pair processor, which is discussed in the upcoming section.

5 Third Problem: Solving Conditional Constraints

We start by generating conditional constraints for Ex. 5 via Thm. 9.

Example 10. Using the polynomial order from Ex. 5 we already have a weak decrease for the usable rules. Moreover, all pairs are weakly decreasing and (2) is strictly decreasing, even without considering the conditions: $[\![\mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, i, n)]\!] = n - i > n - (i + 1) = [\![\mathsf{sum}^\sharp(i + \mathsf{s}(0), n)]\!]$. To show that (2) is also bounded—so that it can be removed—we must show $\models \phi_1$ and $\models \phi_2$.

$$\mathsf{if}^\sharp_{\mathsf{sum}}(i' \le n', i', n') \xrightarrow{\mathsf{i}}{}^* \mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, i, n) \longrightarrow \mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, i, n) \succsim \mathsf{c} \qquad (\phi_1)$$
$$\mathsf{sum}^\sharp(i' + \mathsf{s}(0), n') \xrightarrow{\mathsf{i}}{}^* \mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, i, n) \longrightarrow \mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, i, n) \succsim \mathsf{c} \qquad (\phi_2)$$

To handle conditional constraints like ϕ_1 and ϕ_2, in [9] an induction calculus is developed with rules to transform conditional constraints, where constraints above the line can be transformed into constraints below the line, cf. Fig. 1. Its aim is to turn rewrite conditions $s \xrightarrow{\mathsf{i}}{}^* t$ into conditions involving the order.

Example 11. Using the induction calculus one can transform ϕ_1 and ϕ_2 into ϕ_3 and ϕ_4. Both of these new constraints are satisfied for the polynomial order of Ex. 5 if one chooses $[\![\mathsf{c}]\!] \le 0$, since then ϕ_3 and ϕ_4 evaluate to $v \ge [\![\mathsf{c}]\!]$ and $v - u \ge [\![\mathsf{c}]\!] \longrightarrow (v + 1) - (u + 1) \ge [\![\mathsf{c}]\!]$, respectively.

$$\mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, 0, v) \succsim \mathsf{c} \qquad (\phi_3)$$
$$\mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, u, v) \succsim \mathsf{c} \longrightarrow \mathsf{if}^\sharp_{\mathsf{sum}}(\mathsf{true}, \mathsf{s}(u), \mathsf{s}(v)) \succsim \mathsf{c} \qquad (\phi_4)$$

In [9] soundness of the induction calculus is proven: whenever ϕ can be transformed into ϕ' using one of the inference rules, then $\models \phi'$ implies $\models \phi$. During

I. **Constructor and Different Function Symbol**

$$\frac{f(p_1,\ldots,p_n) \xrightarrow{\mathsf{i}}{}^{*} g(q_1,\ldots,q_m) \wedge \varphi \;\longrightarrow\; \psi}{\mathit{TRUE}} \qquad \text{if } f \text{ is a constructor and } f \neq g$$

II. **Same Constructors on Both Sides**

$$\frac{f(p_1,\ldots,p_n) \xrightarrow{\mathsf{i}}{}^{*} f(q_1,\ldots,q_n) \wedge \varphi \;\longrightarrow\; \psi}{p_1 \xrightarrow{\mathsf{i}}{}^{*} q_1 \wedge \ldots \wedge p_n \xrightarrow{\mathsf{i}}{}^{*} q_n \wedge \varphi \;\longrightarrow\; \psi} \qquad \text{if } f \text{ is a constructor}$$

III. **Variable in Equation**

$$\frac{x \xrightarrow{\mathsf{i}}{}^{*} q \wedge \varphi \;\longrightarrow\; \psi}{\varphi\,\sigma \;\longrightarrow\; \psi\,\sigma} \quad \text{if } x \in \mathcal{V} \text{ and } \sigma = [x/q] \qquad \frac{q \xrightarrow{\mathsf{i}}{}^{*} x \wedge \varphi \;\longrightarrow\; \psi}{\varphi\,\sigma \;\longrightarrow\; \psi\,\sigma} \quad \text{if } x \in \mathcal{V},\ q \text{ has no defined symbols, } \sigma = [x/q]$$

IV. **Delete Conditions**

$$\frac{\varphi_1 \wedge \ldots \wedge \varphi_n \;\longrightarrow\; \psi}{\varphi'_1 \wedge \ldots \wedge \varphi'_m \;\longrightarrow\; \psi} \qquad \text{if } \{\varphi'_1,\ldots,\varphi'_m\} \subseteq \{\varphi_1,\ldots,\varphi_n\}$$

V. **Induction**

$$\frac{f(x_1,\ldots,x_n) \xrightarrow{\mathsf{i}}{}^{*} q \wedge \varphi \;\longrightarrow\; \psi}{\displaystyle\bigwedge_{f(\ell_1,\ldots,\ell_n)\to r\in\mathcal{R}} (r \xrightarrow{\mathsf{i}}{}^{*} q\,\sigma \wedge \varphi\,\sigma \wedge \varphi' \;\longrightarrow\; \psi\,\sigma)} \qquad \text{if } f(x_1,\ldots,x_n) \text{ does not unify with } q$$

where $\sigma = [x_1/\ell_1,\ldots,x_n/\ell_n]$

$$\text{and} \quad \varphi' = \begin{cases} \forall y_1,\ldots,y_m.\ f(r_1,\ldots,r_n) \xrightarrow{\mathsf{i}}{}^{*} q\mu \wedge \varphi\,\mu \longrightarrow \psi\,\mu, & \text{if} \\ \quad \bullet\ r \text{ contains the subterm } f(r_1,\ldots,r_n), & \\ \quad \bullet\ \text{there is no defined symbol in any } r_i, & \\ \quad \bullet\ \mu = [x_1/r_1,\ldots,x_n/r_n], \text{ and} & \\ \quad \bullet\ y_1,\ldots,y_m \text{ are all occurring variables except } \mathcal{V}(r) & \\ \mathit{TRUE}, & \text{otherwise} \end{cases}$$

VI. **Simplify Condition**

$$\frac{\varphi \wedge (\forall y_1,\ldots,y_m.\,\varphi' \longrightarrow \psi') \;\longrightarrow\; \psi}{\varphi \wedge \psi'\,\sigma \;\longrightarrow\; \psi} \qquad \begin{array}{l}\text{if } \mathit{DOM}(\sigma) \subseteq \{y_1,\ldots,y_m\},\\ \text{there is no defined symbol and}\\ \text{no tuple symbol in any } \sigma(y_i),\\ \text{and } \varphi'\,\sigma \subseteq \varphi\end{array}$$

VII. **Defined Symbol without Pairwise Different Variables**

$$\frac{f(p_1,\ldots,p_i,\ldots,p_n) \xrightarrow{\mathsf{i}}{}^{*} q \wedge \varphi \;\longrightarrow\; \psi}{p_i \xrightarrow{\mathsf{i}}{}^{*} x \wedge f(p_1,\ldots,x,\ldots,p_n) \xrightarrow{\mathsf{i}}{}^{*} q \wedge \varphi \longrightarrow \psi} \qquad \text{if } x \text{ is fresh and } p_i \in \mathcal{T}(\mathcal{F},\mathcal{V})$$

Fig. 1. Corrected and generalized rules of the induction calculus from [9]

our formalization of this result in `Bounded-Increase(-Impl).thy`, we detected five interesting facts.

Fact 1. Some of the side-conditions in the original definition are not required, but are most likely used to derive a strategy which chooses which rules to apply. For example, in both (V) and (VII) there have been additional side-conditions in [9]. To be more precise, [9] demands that f is defined in (V). Consequently, formulas with a precondition like $f(x) \xrightarrow{\mathsf{i}}{}^{*} x$ for some constructor f can only be dropped using our version of (V), but not by any of the original rules in [9].

Fact 2. Rule (VII) is unsound in [9] where we had to add the new side-condition $p_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

Fact 3. The textual explanation in [9] for the soundness of the induction rule (V) is completely misleading: it is stated that induction can be performed using the length of the reduction $f(x_1, \ldots, x_n)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} q\sigma$ which is "obviously longer" than the reduction $f(r_1, \ldots, r_n)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} q\sigma$; only in a footnote it is clarified, that for the actual proof one uses the induction relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}} \circ \trianglerighteq$ which is restricted to innermost terminating terms. This restriction actually is the only reason why minimal chains are considered and why in the semantics of $\sigma \models s \xrightarrow{\mathsf{i}}{}^{*} t$ one demands $\mathit{SIN}(s\sigma)$.

Note that if one indeed were able to perform induction on the length of the reduction, then one could drop the condition $\mathit{SIN}(s\sigma)$ and consider arbitrary, non-minimal chains. This would have the immediate advantage that bounded increase would be applicable on sets of pairs and TRSs which arise from transformations like [15], where termination of each $t_i\sigma$ in $(\star)$ cannot be ensured, cf. [15, Footnote 3].

The bad news is that we were able to find a counterexample proving that an induction over the length in combination with non-minimal chains is unsound. In detail, consider $\mathcal{R} = \{\mathsf{f} \to \mathsf{g}(\mathsf{f}), \mathsf{f} \to \mathsf{b}, \mathsf{g}(x) \to \mathsf{a}\}$ and $\mathcal{P} = \{\mathsf{h}^\sharp(\mathsf{a}) \to \mathsf{h}^\sharp(\mathsf{f})\}$. Then there is a non-minimal innermost chain $\mathsf{h}^\sharp(\mathsf{a}) \to_{\mathcal{P}} \mathsf{h}^\sharp(\mathsf{f}) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{h}^\sharp(\mathsf{g}(\mathsf{f})) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{h}^\sharp(\mathsf{g}(\mathsf{b})) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{h}^\sharp(\mathsf{a}) \to_{\mathcal{P}} \ldots$, so we should not be able to prove termination of $\mathcal{P}$ if minimality is dropped from the definition of termination.

However, we show that using the induction calculus—where now $\sigma \models s \xrightarrow{\mathsf{i}}{}^{*} t$ is defined as $s\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} t\sigma \wedge t\sigma \in \mathit{NF}(\mathcal{R})$—in combination with the conditional reduction pair processor we actually can derive termination of $\mathcal{P}$, where we will see that the problem is the induction rule. To this end, we let $\mathsf{h}^\sharp(\mathsf{f}) \xrightarrow{\mathsf{i}}{}^{*} \mathsf{h}^\sharp(\mathsf{a}) \longrightarrow \phi$ be some arbitrary conditional constraint that may arise from the generalized reduction pair processor where ϕ might be $\mathsf{h}^\sharp(\mathsf{a}) \succ \mathsf{h}^\sharp(\mathsf{f})$ or $\mathsf{h}^\sharp(\mathsf{a}) \succsim \mathsf{c}$. Using (II) this constraint is simplified to $\mathsf{f} \xrightarrow{\mathsf{i}}{}^{*} \mathsf{a} \longrightarrow \phi$. Next, we apply (V), leading to $\mathsf{g}(\mathsf{f}) \xrightarrow{\mathsf{i}}{}^{*} \mathsf{a} \wedge (\mathsf{f} \xrightarrow{\mathsf{i}}{}^{*} \mathsf{a} \longrightarrow \phi) \longrightarrow \phi$ and $\mathsf{b} \xrightarrow{\mathsf{i}}{}^{*} \mathsf{a} \longrightarrow \phi$ where the latter constraint is immediately solved by (I). And the former constraint is always satisfied, since $\mathsf{f}\sigma = \mathsf{f} \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} \mathsf{a} = \mathsf{a}\sigma \in \mathit{NF}(\mathcal{R})$ for every $\mathcal{F}$-normal substitution σ.

Fact 4. The main reason for the unsoundness of the induction rule in the previous counterexample is the property that $\mathcal{R}$ is not uniquely innermost normalizing (UIN): f can be evaluated to the two different normal forms a and b.

And indeed, we could show that instead of minimality of the chain, one can alternatively demand UIN of $\mathcal{R}$ to ensure soundness of the induction rule. To be more precise, using UIN of $\mathcal{R}$ we were able to prove soundness of the induction rule via an induction on the distance of $f(x_1, \ldots, x_n)\sigma$ to some normal form.

To have a decidable criterion for ensuring UIN, we further proved that UIN is ensured by confluence of $\mathcal{R}$, and that confluence is ensured by weak orthogonality, i.e., for left-linear $\mathcal{R}$ that only have trivial critical pairs. Here, especially for the latter implication, we required several auxiliary lemmas like the parallel moves lemma, cf. [2, Chapter 6.4] and `Orthogonality.thy`.

Fact 5. As we want to apply Isabelle's code generator [11] on our certification algorithm, we have to formalize conditional constraints via a deep embedding. Therefore, we had the choice between at least two alternatives how to deal with bound variables: we can use a dedicated approach like Nominal Isabelle [21,22], or we perform renamings, α-equivalence, ... on our own.[7]

We first formalized everything using the latter alternative, where we just defined an inductive datatype [4] with constructors for atomic constraints, one constructor for building implications of the shape $\cdot \wedge \cdots \wedge \cdot \longrightarrow \cdot$ (where the set of premises is conveniently represented as a list), and one constructor for universal quantification which takes a variable and a constraint as argument.

Afterwards, we just had to deal with manual renamings at exactly one place: in the definition of applying a substitution on a constraint. Here, for quantified constraints we define $(\forall x.\phi)\sigma = \forall y.(\phi(\sigma\{x := y\}))$ where y is a fresh variable w.r.t. the free variables in σ and ϕ. Note that α-equivalence of two constraints ϕ and ψ can then also easily be checked by just applying the empty substitution on both ϕ and ψ, since the substitution algorithm returns the same result on α-equivalent constraints. Further note, that in our application—certification of transformations on constraints—we require subsumptions checks instead of α-equivalence. Therefore, being able to use equality instead of α-equivalence does not help that much. So, by using an inductive datatype we had minimal effort to integrate renamings. Furthermore, all required algorithms could easily be formulated using the comfort of Isabelle's function package [14].

We additionally tried to perform the same formalization using Nominal Isabelle, since we have been curious what we would gain by using a dedicated package to treat bound variables. Here, our initial attempts have been quite disappointing for the following reasons.

We could not define the datatype for conditional constraints as before, since for the implications we had to manually define two datatypes: one for constraints and one for lists of constraints. Moreover, for several functions which have been accepted without problems using the function package, their nominal counterparts require additional properties which had to be manually proven, including

[7] Of course, also for building sequences of pairs $\mathcal{P}$ (sets of rules) we could have used nominal to avoid manual renamings in rewrite rules. However, since IsaFoR is quite large (over 100,000 lines) we did not want to change the representation of rules until there are extremely good reasons.

new termination proofs. The overall overhead of these manual proofs was in our case by far larger than the one for manual renaming. Moreover, as far as we know, also for code generation some manual steps would be necessary. In the end, we aborted our attempt to have a fully formalized version of conditional constraints which are based on Nominal Isabelle.

6 Babylonian Square Root Algorithms

Eventually, all required theorems for bounded increase have been formalized, and certification algorithms have been integrated in CeTA. Clearly, we tried to certify the bounded increase proofs that have been generated by AProVE. Here, nearly all proofs have been accepted, except for two problematic kinds of proofs.

The one problem was that CeTA discovered a real implementation bug which remained undetected since 2007: AProVE applies (IV) also inside induction hypotheses, i.e., under certain circumstances it simplified $(\psi \longrightarrow \phi) \longrightarrow \phi$ to $(\emptyset \longrightarrow \phi) \longrightarrow \phi$ as $\emptyset \subseteq \{\psi\}$. After the bug has been fixed, for one of the TRSs, a termination proof could not be generated anymore.

The other problem has been an alternative criterion to ensure boundedness, cf. the following example.

Example 12. For the TRS GTSSK07/cade14 from the termination problem database, the generalized reduction pair processor is used with the interpretation where $[\mathsf{diff}^\sharp](x_1, x_2) = -1 + x_1^2 + x_2^2 - 2x_1x_2$, $[\mathsf{s}](x) = x+1$, $[\mathsf{0}] = 0$, and $[\mathsf{c}] = -1$.[8] After applying the induction calculus, one of the constraints is $\mathsf{diff}^\sharp(\mathsf{s}(\mathsf{0}), x) \succsim \mathsf{c}$ which is equivalent to $x^2 - 2x \geq -1$. This inequality is valid, but it cannot be shown using the standard criterion of absolute positiveness [12], which is also the criterion that is used in CeTA. Here, the trick is, that $[\mathsf{diff}^\sharp](x_1, x_2) = (x_1 - x_2)^2 - 1$ and hence $[\mathsf{diff}^\sharp](\dots) \geq 0 - 1 \geq -1 = [\mathsf{c}]$.

In fact, in the previous example AProVE was configured such that tuple symbols are interpreted as $[f^\sharp](x_1, \dots, x_n) = f_0 + (f_1x_1 + \dots + f_nx_n)^2$ for suitable values f_i. Then $[\mathsf{c}]$ is chosen as $\min_{f^\sharp \in \Sigma} f_0$ such that by construction, $f^\sharp(\dots) \succsim \mathsf{c}$ always holds and no constraints have to be checked for boundedness.

If we knew that the tuple symbols are interpreted in this way and all values f_i were provided, it would be easy to conclude boundedness. However, we refrained from adding a special format to express interpretations of this shape, as it would require a new dedicated pretty printer in AProVE. Instead, we want to be able to detect during certification, whether for some arbitrary interpretation $[f^\sharp](x_1, \dots, x_n) = p$ we can find values f_i such that p is equivalent to $f_0 + (f_1x_1 + \dots + f_nx_n)^2$ (which is equivalent to $f_0 + \sum f_i^2x_i^2 + \sum 2f_if_jx_ix_j$).

To this end, we first transform p into summation normal form $a_0 + \sum a_ix_i^2 + \dots$ with concrete values a_i, and afterwards we figure out all possible values for each f_i: $f_0 = a_0$ and $f_i = \pm\sqrt{a_i}$ for $i > 0$. For each possible combination of $(f_0, \dots, f_n)$ we can then just check whether $p = f_0 + (f_1x_1 + \dots + f_nx_n)^2$.

[8] The detailed proof including the TRS is available at `http://termcomp.uibk.ac.at/termcomp/competition/resultDetail.seam?resultId=415507`

However, for computing the values of f_i we need an executable algorithm to compute square roots of integers and rationals. To this end, we formalized variants of the Babylonian method to efficiently compute square roots. Afterwards, indeed all bounded increase proofs from AProVE could be certified by CeTA (version 2.10).

The original Babylonian algorithm is an instance of Newton's method. It approximates $\sqrt{n}$ by iteratively computing $x_{i+1} := \frac{\frac{n}{x_i}+x_i}{2}$ until x_i^2 is close enough to n. Although we did not require it—it does not deliver precise results—for completeness we formalized it, where the domain is some arbitrary linearly ordered field of type $'a$. Here, the main work was to turn the convergence proof into a termination proof. For example, we had to solve the problems that the value to which the function converges is not necessarily a part of the domain: consider $\sqrt{2}$ and $'a$ being the type of rationals.

The precise algorithms are based on the following adaptation. In order to compute square roots of integers we simply use integer divisions $x \div y = \lfloor \frac{x}{y} \rfloor$. Moreover, the result is returned as an option-type, i.e., either we return some number (the square-root), or we return nothing, indicating that there is no integer x such that $x^2 = n$.

$$\begin{aligned} \textit{int-main}\ x\ n = (&\mathsf{if}\ x < 0 \vee n < 0\ \mathsf{then}\ \textit{None}\ \mathsf{else}\ (\mathsf{if}\ x^2 \leq n \\ &\mathsf{then}\ (\mathsf{if}\ x^2 = n\ \mathsf{then}\ \textit{Some}\ x\ \mathsf{else}\ \textit{None}) \\ &\mathsf{else}\ \textit{int-main}\ ((n \div x + x) \div 2)\ n)) \end{aligned}$$

Termination can be proven more easily by just using x as measure. Soundness is also trivially proven as we only return *Some* x if $x^2 = n$. Indeed, the hardest part was to prove completeness which is stated as follows.

Theorem 13. $x \geq 0 \implies x^2 = n \implies y^2 \geq n \implies y \geq 0 \implies n \geq 0 \implies$ *int-main* y n = *Some* x.

To this end, we had to show that once the value of x^2 is below n, then there is no solution. Here, in the proof the non-trivial inequality $(x^2 \div y) \cdot y + y^2 \geq 2xy$ occurred. It trivially holds if one used standard division instead of integer division. However, it took quite a while to prove the desired inequality where we first ran into several dead ends as we tried to use induction on x or y. The solution was to express x^2 as $(y-x)^2+y\cdot(2x-y)$ and then divide both summands by y, cf. the detailed proof in `Sqrt-Babylon.thy`.

Using soundness and completeness of *int-main*, it was easy to write an algorithm *sqrt-int* which invokes *int-main* with a suitable starting value of x (larger than $\sqrt{n}$) and performs a case analysis on whether n is 0, positive, or negative.

Theorem 14. *set* (*sqrt-int* n) = $\{x :: \textit{int}.\ x^2 = n\}$.

In a similar way we construct a square-root algorithm *sqrt-nat* for the naturals, where *int-main* is invoked in combination with conversion functions between naturals and integers.

Theorem 15. *set* (*sqrt-nat* n) = $\{x :: \textit{nat}.\ x^2 = n\}$.

Note, that since *int-main* only works on the positive integers, it sounds more sensible to directly implement it over the naturals. Then conditions like $x \geq 0$ can just be eliminated. Actually, when we started our formalization, we followed this approach. But it turned out that it was by far more cumbersome to perform arithmetic reasoning on the naturals than on the integers. The reason was that differences like $x - y + z + y$ easily simplify to $x + z$ over the integers, but require side-conditions like $x \geq y$ on the naturals. And the amount of required side-conditions was by far larger than storing that certain values are non-negative.

We also formalized an algorithm *sqrt-rat* where we used known facts on co-primality from the Isabelle distribution. In brief, given some rational number $\frac{p}{q}$ for integers p and q, *sqrt-rat* returns $\pm\frac{\sqrt{p}}{\sqrt{q}}$, if this is well-defined, or nothing.

Theorem 16. *set* (*sqrt-rat* n) $= \{x :: rat.\ x^2 = n\}$.

Note that using Thm. 16 one can easily figure out that $\sqrt{2}$ is an irrational number, just by evaluating that *sqrt-rat* 2 is the empty list. Moreover, since the Babylonian algorithm is efficient, it also is no problem to check that $\sqrt{1234567890123456789012345678901234567890}$ is irrational.

7 Summary

We formalized the termination technique of bounded increase. To this end, in addition to the pen-and-paper proof we had to prove the missing fact that signature extensions are sound for innermost rewriting. Moreover, we not only showed that the "obvious reason" for soundness of the induction rule is wrong, but additionally provided a condition under which this obvious reason is sound: unique innermost normalization. This property follows from weak orthogonality, and our formalization contains—as far as we know—the first mechanized proof of the fact that weak orthogonality implies confluence. For the certification algorithm we also required some algorithm to precisely compute square roots, where we adapted the Babylonian approximation algorithm to obtain precise algorithms for the naturals, integers, and rationals.

All variants for computing square roots ($\approx$ 700 lines) have been made available in the archive of formal proofs, and the remaining formalization ($\approx$ 4,700 lines) is available at `http://cl-informatik.uibk.ac.at/software/ceta/`.

Acknowledgments. We thank Cezary Kaliszyk for his help on Nominal Isabelle and we thank the anonymous referees for their helpful comments and remarks.

References

1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236, 133–178 (2000)
2. Baader, F., Nipkow, T.: Term Rewriting and All That, Cambridge (1998)
3. Berghofer, S., Bulwahn, L., Haftmann, F.: Turning inductive into equational specifications. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 131–146. Springer, Heidelberg (2009)

4. Berghofer, S., Wenzel, M.: Inductive datatypes in HOL - lessons learned in formal-logic engineering. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 19–36. Springer, Heidelberg (1999)
5. Blanqui, F., Koprowski, A.: COLOR: A Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. Mathematical Structures in Computer Science 21(4), 827–859 (2011)
6. Contejean, E., Courtieu, P., Forest, J., Pons, O., Urbain, X.: Automated certified proofs with C*i*ME3. In: Proc. RTA 2011. LIPIcs, vol. 10, pp. 21–30 (2011)
7. Giesl, J., Raffelsieper, M., Schneider-Kamp, P., Swiderski, S., Thiemann, R.: Automated termination proofs for Haskell by term rewriting. ACM Transactions on Programming Languages and Systems 33(2), 7:1–7:39 (2011)
8. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: automatic termination proofs in the dependency pair framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 281–286. Springer, Heidelberg (2006)
9. Giesl, J., Thiemann, R., Swiderski, S., Schneider-Kamp, P.: Proving Termination by Bounded Increase. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 443–459. Springer, Heidelberg (2007) Proofs and examples available in technical report AIB-2007-03, `http://aib.informatik.rwth-aachen.de`
10. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
11. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 103–117. Springer, Heidelberg (2010)
12. Hong, H., Jakuš, D.: Testing positiveness of polynomials. Journal of Automated Reasoning 21(1), 23–38 (1998)
13. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean Termination Tool 2. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009)
14. Krauss, A.: Partial and nested recursive function definitions in higher-order logic. Journal of Automated Reasoning 44(4), 303–336 (2010)
15. Krauss, A., Sternagel, C., Thiemann, R., Fuhs, C., Giesl, J.: Termination of Isabelle functions via termination of rewriting. In: van Eekelen, M., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011. LNCS, vol. 6898, pp. 152–167. Springer, Heidelberg (2011)
16. Lankford, D.: On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA (1979)
17. Sternagel, C., Thiemann, R.: Signature extensions preserve termination. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 514–528. Springer, Heidelberg (2010)
18. Sternagel, C., Thiemann, R.: Certification of nontermination proofs. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 266–282. Springer, Heidelberg (2012)
19. Thiemann, R.: Executable Transitive Closures. In: The Archive of Formal Proofs (February 2012), `http://afp.sf.net/entries/Transitive-Closure-II.shtml`
20. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 452–468. Springer, Heidelberg (2009)
21. Urban, C.: Nominal reasoning techniques in Isabelle/HOL. Journal of Automated Reasoning 40(4), 327–356 (2008)
22. Urban, C., Kaliszyk, C.: General bindings and alpha-equivalence in Nominal Isabelle. Logical Methods in Computer Science 8(2) (2012)