Noname manuscript No.
(will be inserted by the editor)

Formalizing the Dependency Pairs Criterion for Innermost Termination

Ariane Alves Almeida · Mauricio Ayala-Rincón

Received: date / Accepted: date

Abstract The dependency pair criterion is a well-known mechanism to analyze termination of term rewriting systems, which are an adequate framework for reasoning about functional specifications, and thus about their termination. Functional specifications with an operational semantics based on eager evaluation are related, in the rewriting framework, to the innermost reduction relation. This paper presents a formalization in PVS of the dependency pair criterion for the innermost reduction relation: a term rewriting system is innermost noetherian if and only if it is terminating by the dependency pair criterion.

Keywords Automating Termination · Termination of Rewriting Systems · Dependency Pairs · Innermost Reduction

1 Introduction

Although closely related to the Halting Problem [Tur36], and thus undecidable, termination is a relevant property for computational objects. This property is crucial to state correctness of programs, since it can state that every input produces an adequate output only if there is indeed an output provided for such input. Even in concurrent and reactive systems, important properties as progress and liveness are related with termination.

It is well-known that Term Rewriting Systems (TRSs) are an adequate formal framework to reason about functional programs. In this context, dependency pairs (DPs), introduced by Arts and Giesl in [AG98], provide a robust criterion to analyze termination. Instead of checking decreasingness of rewriting rules, this criterion aims to check just decreasingness of the relevant fragments of rewriting

Ariane Alves Almeida
Universidade de Brasília
E-mail: arianealves@aluno.unb.br

Mauricio Ayala-Rincón
Universidade de Brasília
E-mail: ayala@unb.br

rules built from defined symbols. Indeed, a dependency pair consists of the left-hand side (*lhs*) of a rewriting rule and a subterm of the right-hand side (*rhs*) of the rule headed by a defined symbol. Thus, a dependency pair expresses the dependency of a function on (even recursive) calls of any function as done in implementations of the *ranking function* criterion that aims to provide a measure over data exchanging points of a program that decreases regarding some well-founded order ([Tur49]). For functional programs, such measures are given over the arguments of each possible (recursive) function call, and it is expected that they decrease after each function call. This is indeed the semantics of termination used in several proof assistants; in particular, in the Prototype Verification Systems (PVS) such *ranking functions* should be provided by the specifier, as part of each recursive definition, and the decreasingness requirements are implemented through the so called *termination Type Correctness Conditions* (for short, termination TCCs). Termination TCCs are *proof obligations* built by static analysis over the recursive definitions, stating that the measure of the actual parameters of each recursive call strictly decreases regarding the measure of the formal parameters.

Eager evaluation conducts the operational semantics of several functional languages, and in particular of the functional language PVS0 specified in PVS for the verification of equivalence of different criteria to automate termination (available as part of the NASA LaRC PVS library at `https://github.com/nasa/pvslib`). The operational semantics and semantics of termination of this language is described in [RMAR$^+$18]. Eager evaluation of functional programs corresponds to innermost (rewriting) derivations. Thus to provide formal support to adaptations of the DP criterion over functional programming it is essential to verify the DP criterion for innermost reductions [AG00].

Main contribution. This work presents a complete formalization of the DP criterion for innermost reduction. The formalization extends the PVS library for TRSs `TRS` that embraces the basic notions of rewriting as well as some elaborated results (e.g., [GAR10], [ROGAR17]). This library includes specifications of terms, positions, substitutions, abstract reduction relations, term rewriting systems which are adequate for the development of formalizations that remain close to article and textbook proofs, as the one presented in this paper. Although having notions such as noetherianity, `TRS` did not provide some elements required to fulfill the objective of formalizing the innermost DP criterion. In this sense, this work brings as a minor contribution specifications and formalizations related with the innermost reduction, non-root reduction and reduction over descendant relations, and as a major one, the formalization of the equivalence between the innermost DP criterion and the noetherianity of the innermost reduction relation.

Outline. Section 2 gives a brief overview on the basic notions of rewriting and the Dependency Pairs criterion, along with definitions of specific rewriting strategies required in the formalization ahead. Section 3 presents the elements of the theory `TRS` and those new ones that were required for this formalization. Section 4 describes the proof that innermost noetherianity implies termination by the dependency pair criterion, and Section 5 the converse. Section 6 discusses related work and Section 7 concludes and discusses future work. The formalization is available as part of the `TRS` library at `http://trs.cic.unb.br` and also at the NASA PVS library `https://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/`.

2 Basic Notions

Standard rewriting notation for terms, subterms, positions and substitutions (e.g., [BN98]), will be used. Given any relation R, R^+ and R^* denote, respectively, its transitive and reflexive-transitive closure. The application of R^* will be referred as *derivation*. For a relation R and element s, if there exists t such that $s\ R\ t$ holds, then s is said to be R-reducible, otherwise, it is said to be in R-normal form, denoted by $nf_R(s)$.

A TRS E is a set of rewriting rules that are ordered pairs of terms in $T(\Sigma, V)$, the set of terms freely generated from a countable set of variables V according to a signature Σ. Whenever the set E is clear from the context, it will be omitted in the notation. Each term $t \in T(\Sigma, V)$ is thus given as a variable or as a function symbol g applied to a tuple of terms of length given by the arity of g according to the signature Σ. In order to keep the notation close to the one in the specification, the symbol `f` is not used as a function symbol, but as the special operator that returns the root function symbol of application terms, which is automatically created when the datatype for terms is specified. Positions of terms are given as sequences of naturals, as usual: the set of positions of a term t, denoted as $Pos(t)$ includes the *root* position that is the empty sequence, denoted as λ, and if t is an application, say $g(t_1, \ldots, t_n)$, all positions of the form $\{i\pi \mid 1 \leq i \leq n, \pi \in Pos(t_i)\}$. Given a position $\pi \in Pos(t)$, the subterm of t at position π is denoted as $t|_\pi$. The subterm relation is denoted by $\trianglerighteq$: $s \trianglerighteq t$, if there exists $\pi \in Pos(t)$ such that $s = t|_\pi$. If such given position π is such that $\pi \neq \lambda$, s is called a proper subterm of t, which is denoted as $s \triangleright t$. Notation $s[\pi \leftarrow t]$ is used to denote the term resulting from replacing the subterm at position $\pi (\in Pos(s))$ of s by t.

A rewriting rule is denoted by $l \longrightarrow r$, and should satisfy the additional restrictions that $l \notin V$ and that each variable occurring in its right-hand side (for short, *rhs*) r also occurs in its left-hand side (for short, *lhs*) l. Given a TRS E, a term s is said to be *reducible at position* $\pi \in Pos(s)$ if there exist some rule $l \longrightarrow r$, substitution σ and term t such that $l\sigma = s|_\pi$ and $t = s[\pi \leftarrow r\sigma]$; then s is said to reduce to t at position π and is denoted as $s \xrightarrow{\pi} t$. If no specific position is given, but there exists some position $\pi \in Pos(s)$ and term t such that $s \xrightarrow{\pi} t$, s is said to be *reducible*, and whenever t is given, s is said to *reduce* to t, denoted as $s \longrightarrow_E t$.

In some specific implementations, such as the one used in this work to deal with chains of *Dependency Pairs*, it is interesting to avoid reductions at root position of terms. For this, one uses the relation *non-root reduction*, which is denoted by $\longrightarrow_{>\lambda}$, is induced by a TRS E and relates terms s and t whenever $s \xrightarrow{\pi} t$ for some $\pi \in Pos(s)$ such that $\pi \neq \lambda$.

A term s is said to be *innermost reducible at position* $\pi \in Pos(s)$ if $nf_{\longrightarrow_{>\lambda}}(s|_\pi)$ and $s \xrightarrow{\pi}_E t$ for some term t; this is denoted as $s \xrightarrow{\pi}_{in} t$. If no specific position is given, but there exists some position $\pi \in Pos(s)$ and term t such that $s \xrightarrow{\pi}_{in} t$, s is said to be *innermost reducible*, and whenever t is given, s is said to *innermost reduce* to t; this is denoted as $s \longrightarrow_{in} t$. Whenever the innermost reduction takes place at a position $\pi \neq \lambda$, one has a so called *non-root innermost reduction*, denoted by $\longrightarrow_{in_{>\lambda}}$.

Another important relation in this paper is the descendants of a given term though a given relation. The *reduction relation restricted to (descendants of) a* *term* t is induced by pairs of terms u, v derived from t, that is $t \longrightarrow^* u$ and $t \longrightarrow^* v$, and such that $u \longrightarrow v$. The notation used is $\underset{t}{\longrightarrow}$. For pairs of terms that are descendants of t and related one with the other by a reduction at specific position π, the notation $\overset{\pi}{\underset{t}{\longrightarrow}}$ is used. Analogous notation applies to innermost and non-root reductions. Also, regarding specific terms, a term s is (innermost) terminating if no infinite derivation starts with it, and the notation used is $\downarrow(s)$ ($\downarrow_{in}(s)$, for the case of innermost terminating). If the term is not terminating, the notation $\uparrow$ (or $\uparrow_{in}$) is used. Whenever a term is not terminating, but all its proper subterms are, one says the term is *minimal non terminating* (*mnt* for short, denoted by $\Uparrow$), and for innermost termination one says *minimal innermost non terminating* (*mint* for short, denoted by $\Uparrow_{in}$).

The termination analysis for rewriting systems aims to verify the non existence of infinite reduction steps (derivations) for every term whereupon the reduction relation is applied. In order to do this, the DP technique, proposed in [AG97], analyzes the possible reductions in a term resulting from a previous reduction, i.e., those that can arise from defined symbols on the *rhs*'s of rules. Thus, it analyzes the *defined symbols* of a TRS E, i.e., the set given by $D_E = \{g \mid \exists(l \longrightarrow r \in E) : \mathtt{f}(l) = g\}$.

Definition 1 (Dependency Pairs) Let E be a TRS. The set of *Dependency Pairs* for E is given as

$$DP(E) = \{\langle l, t\rangle \mid l \longrightarrow r \in E \ \wedge \ r \trianglerighteq t \ \wedge \ \mathtt{f}(t) \in D_E\}$$

Standard definitions of DPs substitute defined symbols by new *tuple symbols* to avoid (innermost) reductions at root positions, which is required for the analysis of termination (e.g., [AG00], [TG03]). In this formalization instead of extending the language with such tuple symbols, reductions at root position are avoided through the restriction to non-root (innermost) derivations. This choice will be made clearer in Section 3. The advantage of our approach is that in this manner, it is not required dealing with new reduction relations over the extended signature.

Each dependency pair represents the possibility of a future reduction after one (innermost) reduction step. However, distinct rewriting redexes can appear in terms after (possibly) several (innermost) reduction steps, which can also give rise to another possible reduction, producing a *Dependency Chain*.

Definition 2 (Dependency Chain) A *dependency chain* to a TRS E, E-chain, is a finite or infinite sequence of dependency pairs $\langle s_1, t_1\rangle\langle s_2, t_2\rangle \dots$ for which there exists a substitution σ such that $t_i\sigma \longrightarrow^*_{>\lambda} s_{i+1}\sigma$, for every i below the length of the sequence, after renaming the variables of pairs with disjoint new variables.

Similarly, the notion of *Innermost Dependency Chain* is given:

Definition 3 (Innermost Dependency Chain) An *innermost dependency chain* to a TRS E, E-in-chain, is a finite or infinite sequence of dependency pairs $\langle s_1, t_1\rangle\langle s_2, t_2\rangle \dots$ for which there exists a substitution σ such that, for every i below the length of the sequence, $t_i\sigma \longrightarrow^*_{in_{>\lambda}} s_{i+1}\sigma$ and $nf_{\longrightarrow_{>\lambda}}(s_i)$, after renaming the variables of pairs with disjoint new variables.

Termination is then defined as the absence of infinite (innermost) dependency chains (cf., Theorems 3.2 and 4 of [AG97]).

3 Specification

This paper brings an extension of the PVS term rewriting library `TRS`. This theory is a development that already contains the basic elements of abstract reduction systems and TRS, such as reducibility, confluence and noetherianity regarding a given relation, notions of subterms and replacement, etc. Furthermore, this theory embraces several elaborated formalizations regarding such systems, such as confluence of abstract reduction systems (see [GAR08]), the Critical Pair Theorem (see [GAR10]) and orthogonality and its confluence (see [ROGAR17]).

Terms in the theory `TRS` are specified as a datatype with three parameters: nonempty types for variables and function symbols, and the arity function of these symbols. Terms are either variables or applications built as function symbols with a sequence of terms of length equal to its arity. The predicate `app?` holds for application terms and, as previously mentioned, the operator `f` extracts the root function symbol of an application.

The theory `rewrite_rules.pvs` specifies rewriting rules (as pairs of terms, restricted as usual) and the notion of set of defined symbols for a set of rewriting rules E (i.e., D_E) given as below.

```
defined?($E$)($d : symbol$): bool =
  $\exists (e \in E)$: f($lhs(e)$) $= d$
```

Basic elements and results were imported in this formalization, such as terms and rules mentioned above and predicates to represent pertinence of positions of a term (`positonsOF` in theory `positions.pvs`), functions to provide subterm of specific position (`subtermOF` in theory `subterm.pvs`), the replacement operation (`replaceTerm` in theory `replacement.pvs`) and so on. However, specification of some general definitions regarding TRS's required to specify DPs and formalization of several properties were missed and filled as part of this work. Some of these new basic notions and results were included either in already existent theories such as the notion of non-root reduction ($\longrightarrow_{>\lambda}$), specified in theory `reduction.pvs`, or in new complementary basic theories such as `innermost_reduction.pvs` and `restricted_reduction.pvs`, where the relations $\longrightarrow_{in}$ and $\underset{t}{\longrightarrow}$ are found.

Furthermore, the new basic definitions are, mostly, specializations of previously ones, such as the notions presented by pseudocode below, which specify the predicates for relations $\xrightarrow{\pi}$ and $\longrightarrow$ (in theory `reduction.pvs`). Notice that such relations are specified as predicates over pairs of terms in a curryfied way, a discipline followed through the whole `TRS` library that allows one to rely, for instance, on parameterizable definitions and properties provided for arbitrary abstract reductions systems, such as closures of relations (in theory `relations_closure.pvs`), reducibitity and normalization (in theory `ars_terminology.pvs`), noetherianity (in theory `noetherian.pvs`), etc.

```
reduction_fix?($E$)($s, t$ :term, $\pi \in Pos(s)$): bool =
  $\exists (e \in E, \sigma)$:
     $(s|_\pi = lhs(e)\sigma \wedge t = s[\pi \leftarrow rhs(e)\sigma]$

reduction($E$)($s, t$ :term): bool =
  $\exists (e \in E, \sigma, \pi \in Pos(s)$:
     $s|_\pi = lhs(e)\sigma \wedge t = s[\pi \leftarrow rhs(e)\sigma]$
```

The first four pseudocodes in the box below specify, respectively, the new required relations $\longrightarrow_{>\lambda}$, $nf_{\longrightarrow_{>\lambda}}$, $\xrightarrow{\pi}_{in}$, $\longrightarrow_{in}$ and $\longrightarrow_{in_{>\lambda}}$ and are found in theory `innermost_reduction.pvs`. The notion of $\underset{s}{\longrightarrow}$ is given by the fifth predicate in the box (i.e., `rest?`) and is specified for any binary relation R in theory `restricted_reduction.pvs`. At last, `arg_rest?` is a specialization of restricted relations for term rewriting, allowing to fix the argument where innermost reductions can take place between given descendants of a term s (i.e., relation $\underset{t}{\xrightarrow{\pi}}$), which is specified in theory `innermost_reduction.pvs`.

```
non_root_reduction?$(E)(s,t)$: bool =
  $\exists(\pi \in Pos(s) | \pi \neq \lambda)$:
    reduction_fix?$(E)(s,t,\pi)$

is_inn_normal_form?$(E)(s)$: bool =
  $\forall(\pi \in Pos(s) | \pi \neq \lambda)$:
    is_normal_form?(reduction?$(E)$)$(s|_{\pi})$

innermost_reduction_fix?$(E)(s,t,(\pi \in Pos(s)))$: bool =
  is_inn_normal_form?$(E)(s|_{\pi})$ $\wedge$ reduction_fix?$(E)(s,t,\pi)$

innermost_reduction?$(E)(s,t)$: bool =
  $\exists(\pi \in Pos(s))$:
    innermost_reduction_fix?$(E)(s,t,\pi)$

non_root_innermost_reduction?$(E)(s,t)$: bool =
  $\exists(\pi \in Pos(s) | \pi \neq \lambda)$:
    is_inn_normal_form?$(E)(s|_{\pi})$ $\wedge$ reduction_fix?$(E)(s,t,\pi)$

rest?$(R,s)(u,v)$: bool =
  $(s\,R^*\,u) \wedge (s\,R^*\,v) \wedge (u\,R\,v)$

arg_rest?$(E)(s)(k)(u,v)$: bool =
  rest?$(\longrightarrow_{in_{>\lambda}}, s)(u,v)$ $\wedge$
  $\exists(\pi \in Pos(s) | \pi \neq \lambda)$:
    first$(\pi) = k$ $\wedge$
    innermost_reduction_fix?$(E)(u,v,\pi)$
```

Previously mentioned discipline of curryfication and modularity of `TRS` that allows generic application of rewriting predicates and their properties over general rewriting relations is followed. For instance, in the specification of `arg_rest?` above, the predicate `rest?` receives as parameter the relation $\longrightarrow_{in_{>\lambda}}$, that is `non_root_innermost_reduction?`$(E)$.

In theory `dependency_pairs.pvs` the notion of DP and its termination criterion are specified. As previously mentioned, instead of extending the language with tuple symbols, DPs are specified with the same language of the given signature, and thus DPs chained through non-root (innermost) reduction.

```
dep_pair?$(E)(s,t)$: bool =
  app?$(t)$ $\wedge$ defined?$(E)$(f$(t)$) $\wedge$
  $\exists(e \in E): lhs(e) = s \wedge (\exists(\pi \in Pos(rhs(e))) : rhs(e)|_{\pi} = t)$
```

This specification of DPs follows the standard theoretical approach in a straightforward manner. However, it depends on two existential quantifiers that, throughout the proofs, would bring several difficulties about which rule and position had created the DP being analyzed. This is because PVS proofs are based on a sequent calculus, and thus, whenever these existential quantifiers appear in the antecedent of a proof, their Skolemization leads to some arbitrary rule and position making it difficult the constructive generation of derivations of terms associated with chained DPs. It is easy to see that different *rhs* positions, and even different rules can produce identical DPs; take for instance the TRS below, where $\langle h(x,y), g(x,y)\rangle$ can be built in three different manners.

$$\{h(x,y) \longrightarrow h(g(x,y), g(g(x,y),y),y), \quad h(x,y) \longrightarrow g(x,y), \quad g(x,y) \longrightarrow y\}$$

To discriminate the manner in which DPs are extracted from the rewriting rules and to circumvent the difficulties of existential quantifiers, an alternative notion of DP is specified as below.

```
dep_pair_alt?($E$)($e,\pi$): bool =
  $e \in E \wedge \pi \in Pos(rhs(e)) \wedge$
  app?($rhs(e)|_\pi$) $\wedge$ defined?($E$)(f($rhs(e)|_\pi$)
```

Having the rule and position that generate the DPs allows, for instance, further specification of recursive functions to "sew up" the contexts of any infinite chain of DPs in order to build the associated infinite derivations (more details are given in Section 4). Here is important to stress that for termination analysis and automation, whenever `dep_pair_alt?`$(E)(e,\pi)$ and `dep_pair_alt?`$(E)(e',\pi')$ are such that $lhs(e) = lhs(e')$ and $rhs(e)|_\pi = rhs(e')|_{\pi'}$, it is sufficient to consider only one of these DPs. Other easily implementable refinements are discussed in the Section 6 on related work.

In the remaining of the discussion, these two definitions will be distinguished if necessary, and in particular, for the sake of simplicity, the first and second elements of a DP will be identified with the *lhs* of the rule and the subterm at position π of the *rhs* of the rule.

Notice that both specifications for DPs are currifyed, allowing then to define the types `dep_pair`(E) and `dep_pair_alt`(E).

In order to check that an infinite sequence of DPs form an infinite (innermost) dependency chain, it is required, as given in Definitions 2 and 3, that every two consecutive DPs in this sequence be related though (innermost) non-root reductions, after renaming their variables, regarding some substitution. This gives rise to an imprecision since the type of substitutions does not allow infinite domains. This issue is circumvented by specifying sequences DPs in association with sequences of substitutions. So by allowing a different substitution for each DP in the sequence, it is possible to specify the notion of *(innermost) chained DPs* as below.

```
chained_dp?($E$)($dp1, dp2$ :dep_pair($E$))($\sigma_1,\sigma_2$): bool =
  $dp1'2\sigma_1 \longrightarrow^*_{>\lambda} dp2'1\sigma_2$

inn_chained_dp?($E$)($dp1, dp2$ :dep_pair($E$))($\sigma_1,\sigma_2$): bool =
  is_inn_normal_form?($E$)($dp1'1\sigma_1$) $\wedge$ is_inn_normal_form?($E$)($dp2'1\sigma_2$) $\wedge$
  $dp1'2\sigma_1 \longrightarrow^*_{in_{>\lambda}} dp2'1\sigma_2$
```

Above, the elements of a DP, say dp, are projected by the operator $\_'\_$, as $dp'1$ and $dp'2$, used to project elements of tuples in PVS. Using these specifications of (innermost) chained DPs, (innermost) infinite dependency chains are then given as pairs of sequences of DPs and substitutions that satisfy the predicates below.

```
infinite_dep_chain?($E$)($dps$ :sequence[dep_pair($E$)],
                              $sigmas$ :sequence[Sub]): bool =
  $\forall(i, j : nat | i \neq j)$ :
    chained_dp?($E$)($dps(i), dps(i+1)$)($sigmas(i), sigmas(i+1)$)

inn_infinite_dep_chain?($E$)($dps$ :sequence[dep_pair($E$)],
                                  $sigmas$ :sequence[Sub]): bool =
  $\forall(i, j : nat | i \neq j)$ :
    inn_chained_dp?($E$)($dps(i), dps(i+1)$)($sigmas(i), sigmas(i+1)$)
```

Finally, the DP (innermost) termination criterion is specified as below as the absence of such infinite chains. The two first predicates specify the criterion for the standard notion of DPs, and the third and fourth for the alternative one. Notice that alternative DPs are translated into standard DPs in the third and fourth predicates.

```
dp_termination?($E$): bool =
  $\forall$($dps$ :sequence[dep_pair($E$)], $sigmas$ :sequence[$Sub$]):
    $\neg$infinite_dep_chain?($E$)($dps, sigmas$)


inn_dp_termination?($E$): bool =
  $\forall$($dps$ :sequence[dep_pair($E$)], $sigmas$ :sequence[$Sub$]):
    $\neg$inn_infinite_dep_chain?($E$)($dps, sigmas$)

dp_termination_alt?($E$): bool =
  $\forall$($dps\_alt$ :sequence[dep_pair_alt($E$)], $sigmas$ :sequence[$Sub$]):
    LET $dps$ = LAMBDA($i$: nat): ($lhs(dps\_alt(i)'1)$,
                                  $rhs(dps\_alt(i)'1)|_{dps\_alt(i)'2}$) IN
    $\neg$infinite_dep_chain?($E$)($dps, sigmas$)

inn_dp_termination_alt?($E$): bool =
  $\forall$($dps\_alt$ :sequence[dep_pair_alt($E$)], $sigmas$ :sequence[$Sub$]):
    LET $dps$ = LAMBDA($i$: nat) : ($lhs(dps\_alt(i)'1)$,
                                  $rhs(dps\_alt(i)'1)|_{dps\_alt(i)'2}$) IN
    $\neg$inn_infinite_dep_chain?($E$)($dps, sigmas$)
```

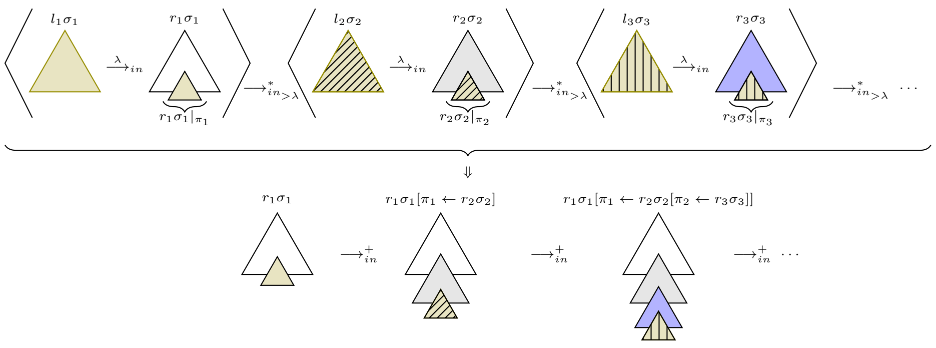
As shown above, several elements were specified to deal with various reduction relations, for which several properties were formalized but not discussed in this paper since the focus here is on formalization of innermost termination by DPs. Furthermore, the alternative version of DPs is used aiming to simplify proofs, and to ensure that the corresponding innermost DP criteria are the same, the equivalence `inn_dp_termination?(E)` $\Leftrightarrow$ `inn_dp_termination_alt?(E)` was formalized as lemma `dp_termination_and_alt_eq` in theory `dependency_pairs.pvs`. This proof is quite simple, building by contraposition infinite sequences of standard chained DPs from alternative ones and vice versa.

4 Necessity for the Innermost Dependency Pairs Criterion

This result is formalized in lemma `inn_noetherian_implies_inn_dp_termination`, specified as below.

```
inn_noetherian_implies_inn_dp_termination: LEMMA
  $\forall(E)$:
    noetherian?(innermost_reduction?($E$)) $\Longrightarrow$ inn_dp_termination?($E$)
```

The formalization follows by contraposition, by building an infinite sequence of terms associated with an infinite innermost derivation from an infinite chain of dependency pairs. In order to build these terms, it is necessary to accumulate the contexts where the reductions would take place regarding the *rhs* of the rule that generates each DP in the chain. The intuition of this formalization follows directly from the theory, and is summarized in the sketch given in Figure 1.



$l_1\sigma_1$
$\xrightarrow{\lambda}_{in}$
$r_1\sigma_1$
$r_1\sigma_1|_{\pi_1}$
$\longrightarrow^*_{in_{>\lambda}}$
$l_2\sigma_2$
$\xrightarrow{\lambda}_{in}$
$r_2\sigma_2$
$r_2\sigma_2|_{\pi_2}$
$\longrightarrow^*_{in_{>\lambda}}$
$l_3\sigma_3$
$\xrightarrow{\lambda}_{in}$
$r_3\sigma_3$
$r_3\sigma_3|_{\pi_3}$
$\longrightarrow^*_{in_{>\lambda}}$ $\cdots$
$\Downarrow$
$r_1\sigma_1$
$\longrightarrow^+_{in}$
$r_1\sigma_1[\pi_1 \leftarrow r_2\sigma_2]$
$\longrightarrow^+_{in}$
$r_1\sigma_1[\pi_1 \leftarrow r_2\sigma_2[\pi_2 \leftarrow r_3\sigma_3]]$
$\longrightarrow^+_{in}$ $\cdots$

Fig. 1 Proof sketch: building infinite innermost derivations from infinite innermost DP-chains.

Since there is a root reduction associated with each DP in the sequence, from its *lhs* to the *rhs* of the related rule, and a non-root innermost derivation to reach the *lhs* of the next DP from the *rhs* of the current DP, it is relatively simple to manipulate the rules and positions using the alternative dependency chain specification to build recursively a sequence of terms related by $\longrightarrow^+_{in}$ through the replacement operation.

To perform this construction, the recursive function `term_pos_dps_alt` is used, taking sequences of DPs and substitutions and producing indexed pairs of term and position accumulating contexts in such a way that the terms are related by $\longrightarrow^+_{in}$ whenever the given sequence is chained. As illustrated in the Figure 1, if the sequence is chained, the first pair of term and position is computed as $(r_1\sigma_1, \pi_1)$; the second as $(r_1[\pi_1 \leftarrow r_2\sigma_2], \pi_1 \circ \pi_2)$; and so on. The function `term_pos_dps_alt` uses as previous context, `Ca`, and replaces the *rhs* of the current DP by the *rhs* of the next DP in the sequence. Positions to perform the replacement are given by accumulation of the positions in the alternative definition of DPs.

```
 term_pos_dps_alt(E)(dps:sequence[dep_pair_alt(E)],
                          sigmas:sequence[Sub], i:nat):
                          RECURSIVE {(C, π) | π ∈ Pos(C)}=
   IF  i = 0  THEN
     (rhs(dps(0)'1)sigmas(i), dps(0)'2)
   ELSE LET (C, π) = term_pos_dps_alt(E)(dps, sigmas, i − 1) IN
     (C[π ←rhs(dps(i)'1)sigmas(i)], π ∘ dps(i)'2)
   ENDIF
 MEASURE i
```

Then, an infinite sequence of terms can be built from an infinite chain given by sequences of DPs and substitutions *dps* and *sigmas* as:

```
LAMBDA(i: nat):   term_pos_dps_alt(E)(dps, sigmas, i)'1
```

Notice that the function `term_pos_dps_alt` would provide an infinite sequence of terms for any pair of infinite sequences of DPs and substitutions, disregarding if they form an infinite innermost chain or not. To prove that the generated infinite sequence indeed describes an infinite derivation for the relation $\longrightarrow_{in}$, this function should be applied to a pair *dps* and *sigmas* that constitutes an infinite chain.

This is proved by showing the non-noetherianity of $\longrightarrow^{+}_{in}$ that relates consecutive terms generated by the function `term_pos_dps_alt`. The proof follows by induction, whereas for the induction basis it must be proved that the first term generated is related with the second by $\longrightarrow^{+}_{in}$. `term_pos_dps_alt` builds these terms just using the first and second DPs and substitutions, say $((l_1, r_1), \pi_1)$, $((l_2, r_2), \pi_2)$, and σ_1 and σ_2 as in Figure 1, in the chained input. The first term is $r_1\sigma_1$ and the second $r_1\sigma_1[\pi_1 \leftarrow r_2\sigma_2]$, which is equal to $r_1\sigma_1[\pi_1 \leftarrow l_2\sigma_2[\lambda \leftarrow r_2\sigma_2]]$. Since contiguous pairs in the sequence are innermost chained and $\longrightarrow^{*}_{in_{>\lambda}}$ is compatible with contexts (by monotony of closures, since $\longrightarrow_{in}$ is compatible with contexts and $\longrightarrow_{in_{>\lambda}} \subseteq \longrightarrow_{in}$), one has that $r_1\sigma_1 \longrightarrow^{*}_{in_{>\lambda}} r_1\sigma_1[\pi_1 \leftarrow l_2\sigma_2]$. And, also by the innermost chained property, $l_2\sigma_2$ is a normal instance of the *lhs* of a rule, i.e., a single innermost reduction step can be applied only at root position giving $r_2\sigma_2$. Since a single innermost reduction step corresponds directly to a replacement operation, and in this case at root position, one would have one innermost reduction step $r_1\sigma_1[\pi_1 \leftarrow l_2\sigma_2] \xrightarrow{\pi_1}_{in} r_1\sigma_1[\pi_1 \leftarrow r_2\sigma_2]$. Thus, one would have $r_1\sigma_1 \longrightarrow^{+}_{in} r_1\sigma_1[\pi_1 \leftarrow r_2\sigma_2]$. The inductive step considers analogously contiguous DPs and substitutions in the chained input, the only extra details are regarding the current term and position computed in the previous recursive step by `term_pos_dps_alt`. Notice that in the i^{th} iteration the current term can be seen as a context C with a hole at the accumulated position, say π, filled with term $r_i|_{\pi_i}\sigma_i$. Indeed, in the induction basis the context is given by $r_1\sigma_1$ with a hole at position π_1. The term and accumulated position generated by `term_pos_dps_alt` are given as $C[\pi \leftarrow r_{i+1}\sigma_{i+1}]$ and $\pi \circ \pi_{i+1}$. Notice that this term can be seen as a context with a hole at the accumulated position filled with the term $r_{i+1}|_{\pi_{i+1}}$. Finally, observe that $C[r_i|_{\pi_i}\sigma_i] \longrightarrow^{+}_{in} C[r_{i+1}\sigma_{i+1}]$.

Notice that this formalization is very similar to its analytic version, disregarding the specification. However, the construction of an actual function to generate each pair of accumulated context and position simplifies the inductive and constructive proof of the existence of the infinite derivation. Furthermore, proof elements that can seem too trivial must be precisely used, such as the mentioned closure of context, monotony of closures, subset properties and properties regarding composition of positions in replacements. The last specifically used for instance for proving correctness of the *predicate subtyping* condition $\{(C, \pi) \mid \pi \in Pos(C)\}$ of the pairs built by the function `term_pos_dps_alt`. These properties are formalized in the theory `TRS` in a general manner allowing its application for arbitrary rewriting relations.

5 Sufficiency for the Innermost Dependency Pairs Criterion

The formalization is by contraposition. The core of the proof follows the idea in [AG00] to construct infinite chains from infinite innermost derivations. In an implementional level, to go from infinite derivations to infinite sequences of DPs that would create an infinite chain is challenging. Indeed, constructing the DPs requires, initially, choosing *mint* subterms from those terms leading to infinite innermost derivations; afterwards, choosing non-root innermost normalized terms; and, finally, choosing instances of rules that apply at root positions of these terms from which DPs can be constructed. All these choices are based on existential proof techniques. Figure 2 illustrates the main steps of the kernel of the construction of chained DPs:

– Existence of *mint* subterms of innermost non terminating terms is represented as the small triangles inside big ones. This part of the development is explained in Subsection 5.1.
– Existence of non-root innermost normalized terms obtained by derivations (through relation $\longrightarrow_{in_{>\lambda}}$) from these *mint* subterms, represented as vertically striped triangles, is detailed in Subsection 5.2.
– Existence of DPs from rules and substitutions that reduce non-root innermost normalized terms at root position, which also are innermost non terminating, into innermost non terminating terms. The DPs are represented by pairs of small vertically striped and small plain triangles and the latter by reductions (through relation $\xrightarrow{\lambda}_{in}$) from vertically to diagonally striped triangles. This result is explained in Subsection 5.3.

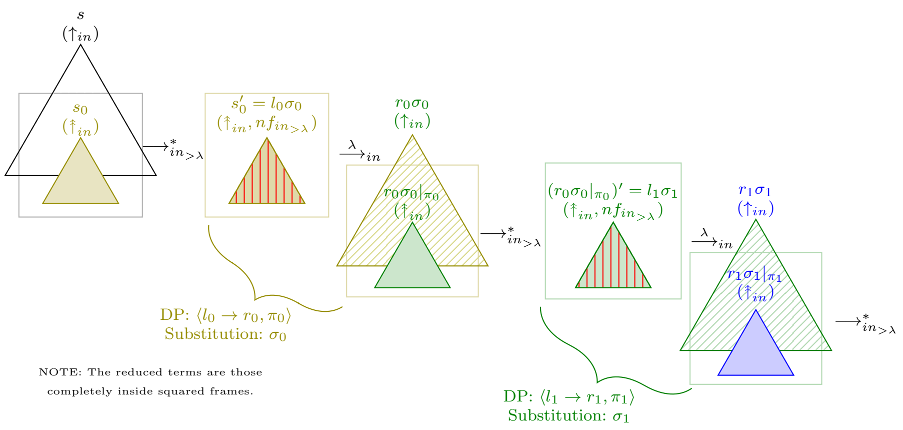

s
$(\uparrow_{in})$
s_0
$(\dagger_{in})$
$\longrightarrow^*_{in_{>\lambda}}$
$s'_0 = l_0\sigma_0$
$(\dagger_{in}, nf_{in_{>\lambda}})$
$\xrightarrow{\lambda}_{in}$
$r_0\sigma_0$
$(\uparrow_{in})$
$r_0\sigma_0|_{\pi_0}$
$(\dagger_{in})$
DP: $\langle l_0 \to r_0, \pi_0 \rangle$
Substitution: σ_0
$\longrightarrow^*_{in_{>\lambda}}$
$(r_0\sigma_0|_{\pi_0})' = l_1\sigma_1$
$(\dagger_{in}, nf_{in_{>\lambda}})$
$\xrightarrow{\lambda}_{in}$
$r_1\sigma_1$
$(\uparrow_{in})$
$r_1\sigma_1|_{\pi_1}$
$(\dagger_{in})$
$\longrightarrow^*_{in_{>\lambda}}$
DP: $\langle l_1 \to r_1, \pi_1 \rangle$
Substitution: σ_1
NOTE: The reduced terms are those completely inside squared frames.

Fig. 2 Proof sketch: building infinite innermost DP-chains from infinite innermost derivations. Notice that the two DPs created, along with their respective substitutions, form chained DPs.

The last step of the construction above, permits as the first one, application of a lemma of existence of *mint* subterms (for innermost non terminating terms). In the last step, this result will allow constructing the required DPs.

Subsection 5.4 then discusses how getting adequate pairs of consecutive chained DPs and associated normal substitutions, and Subsection 5.5, finally, details the construction of the required chain of DPs.

5.1 Existence of *mint* Subterms

The *mint* property ($\Uparrow_{in}$) over terms is specified in the box below by predicate `minimal_non_innermost_terminating?`. Also in this box one has the specification of lemma `inn_non_terminating_has_mint`, whose formalization ensures the existence of *mint* subterms regarding innermost non terminating terms. The proof follows by induction on the structure of the term. The induction basis is trivial since variable terms are not reducible, so variables cannot give rise to infinite derivations. For the inductive step, whenever the term t has an empty list of arguments (that is, t is a constant), the only position it has is its root, thus, the *mint* subterm is the term itself; otherwise, either all its proper subterms are innermost terminating and then the term itself is *mint* or, by induction hypothesis, some of its arguments is innermost non terminating, say its *ith* argument, and then it has a *mint* subterm at some position π, thus, the *mint* subterm of t is chosen as $t|_{i\pi}$.

```
minimal_innermost_non_terminating?(E)(t:term): bool =
  $\uparrow_{in}$(t) ∧ ∀($\pi \in Pos(t) | \pi \neq \lambda$): $\downarrow_{in}$ ($t|_\pi$))

inn_non_terminating_has_mint: LEMMA
  ∀(E)(t: term | $\uparrow_{in}$(t)): ∃($\pi \in Pos(t)$)) : $\Uparrow_{in}$ ($t|_\pi$))
```

5.2 Non-root Innermost Normalization of *mint* Terms

The second step in the formalization proves that every *mint* term can be non-root innermost normalized (into an innermost non terminating term). This result appears to be, as given in analytic proofs, a simple observation since, by definition, every proper subterm of a *mint* term is innermost terminating, and consequently no argument of this term may give rise to an infinite innermost derivation. However, formalizing such result by contradiction requires several auxiliary functions and lemmas related with structural properties of such derivations that also consider positions and arguments in which each reduction step happens. These technicalities of the formalization are necessary to obtain a key result that assuming the existence of an infinite non-root innermost derivation from a *mint* term guarantees that from some of its arguments should start an infinite innermost derivation, which gives the contradiction.

5.2.1 mint *Terms are Non-root Innermost Terminating*

For the remainder of this subsection, as specified below, assume that s, *seqt* and *seqp* are fixed term, sequences of terms and positions, respectively, associated with an infinite non-root innermost derivation on non-root innermost descendants of s, such that the n^{th} term in the sequence *seqt* reduces into the $(n+1)^{th}$ term at position *seqp*(n). Also, l will denote a valid argument of s (and as it will be seen, also a valid argument of any of its descendants).

```
s:term | app?(s)

seqt: sequence[term] | ∀(n: nat): $s \longrightarrow_{in_{>\lambda}} seqt(n)$

seqp: sequence[position] | ∀(n:nat) : $seqp(n) \in Pos(seqt(n))$ ∧
```

```
                                    seqp(n) ≠ λ ∧ seqt(n) →in^seqp(n) seqt(n + 1)

l : posnat | l ≤ length(args(s))
```

The predicate **inf_red_arg_in_inf_nr_im_red** below holds whenever for a sequence of positions there is an infinite number of positions in the sequence starting with the same natural. For *seqp* and l as above, this predicate will be applied to state the existence of an infinite set of indexes in the sequence of terms *seqt* in which the reduction happens at the l^{th} argument. The function **args_of_pos_seq** is just used to give the argument of each position in a sequence of positions.

```
args_of_pos_seq(seq: sequence[position] | ∀(i:nat): seqp(i) ≠ λ)
                 (n : nat):posnat =
          first(seqp(n))

inf_red_arg_in_inf_nr_im_red(seq: sequence[position] |
                                          ∀(i:nat): seqp(i) ≠ λ)
                                     (i:posnat): bool =
  is_infinite(inverse_image(args_of_pos_seq(seq), i))
```

Then, for any l-th argument such that **inf_red_arg_in_inf_nr_im_red**$(seqt)(l)$ holds, the function **nth_index** below provides the index of the sequence in which the $(n+1)^{th}$ reduction at argument l happens.

```
nth_index(E)(s)(seqt)(seqp)(l)(n : nat) : nat =
  choose({m: nat | args_of_pos_seq(seqp)(m) = l ∧
                   card({k: nat | args_of_pos_seq(seqp)(k) = l∧
                                  k < m}) = n})
```

Notice that well-definedness of these functions is a consequence of the type of l that is a dependent type satisfying the predicate **inf_red_arg_in_inf_nr_im_red**, which means that reductions at the l^{th} argument happen infinitely many times. The main technical difficulty of formalizing well-definedness is related with guaranteeing non-emptiness of the argument of the built-in function `choose`. This constraint is fulfilled by the auxiliary lemma below.

```
  exists_nth_in_inf_nr_im_red : LEMMA
    ∀(n:nat): ∃(m:nat):
      args_of_pos_seq(seqp)(m) = l ∧
      card({k:nat | args_of_pos_seq(seqp)(k) = l ∧ k < m}) = n
```

The formalization of this lemma follows by induction on n and, although simple, requires several auxiliary lemmas over sets. In the induction basis, since one has infinite reductions at argument l, the set of indexes where such reductions take place is infinite, and thus, nonempty (by application of the PVS prelude lemma **infinite_nonempty**). Thus, it is possible to use PVS function `min` (over nonempty sets) to choose the smallest index of this set. By the definition of this `min` function, it is ensured that the set of indexes smaller than this minimum in this set is empty, and thus has cardinality zero (by applying PVS prelude lemma **card_empty?**). For the inductive step, one must provide the index where one has a reduction at argument l such that it has exactly $n+1$ indexes smaller than it where reductions at argument l occur. By induction hypothesis, there exists an index m for which reduction take place at argument l, and for which the cardinality of indexes smaller than m with reductions at argument l is n. Thus, the required index is built as the minimum index bigger than m for which the reduction happens at argument l. Correctness of such indexes follows similarly to the induction basis. First, since the predicate `inf_red_arg_in_inf_nr_im_red` holds, it is possible to ensure that the set of indexes greater than index m for which reductions happen at argument l^{th} is infinite, which allows application of the function `min`. Then one builds an equivalent set to the one of all indexes smaller than this minimum as the addition of index m to the set of indexes smaller than m (where one has reductions at argument l^{th}). This construction allows one to use another prelude lemma regarding cardinality of addition of elements in finite sets (`card_add`) to state that the cardinality of this new set is $n + 1$.

Soundness of `nth_index` follows from auxiliary properties such as its monotony and *completeness*, meaning the latter that this function covers exactly (all) the indexes in which reductions happen at the l^{th} argument. The formalization of these properties follows directly from the conditions fulfilled by the natural numbers chosen as the indexes in `nth_index` and prelude lemmas over cardinality of subsets (`card_subset`), since each index provided gives rise to a subset of the next one. These properties allow an easy formalization of a useful auxiliary result stating that for every index of *seqt* below `nth_index(0)` and between `nth_index(`i`)`$+1$ and `nth_index(`$i+1$`)` there happen no reductions in the l^{th} argument (lemma `argument_protected_in_non_nth_index`). And then it is possible to ensure that there are only finitely many non-root innermost reductions regarding a term with *mint* property, which is stated in the lemma below.

mint_is_nr_inn_terminating : LEMMA $\uparrow_{in}(s) \Longrightarrow$ noetherian?$(\underset{s}{\longrightarrow}{}_{in_{>\lambda}})$

This proof follows by contraposition, by assuming the non noetherianity of the $\underset{s}{\longrightarrow}{}_{in_{>\lambda}}$ relation and building then an infinite derivation for some argument of s, as illustrated in Figure 3. Thus, initially one would have an infinite sequence *seqt* of descendants of term s where each one is related to the next one by one step of non-root reduction. From this sequence, since there is a finite number of possible arguments where the reductions can take place and infinitely many reductions taking place in non-root positions, i.e., argument positions, one uses the pigeonhole principle to ensure that there exists some argument position l that satisfies the predicate `inf_red_arg_in_inf_nr_im_red`. This allows the use of function `nth_index` to extract exactly the index of the sequence where such reduction occurs. Then the required infinite derivation is built in two steps. First, since one has, by definition, that $s \longrightarrow_{>\lambda} seqt(0)$, this leads to a finite sequence of reduced terms that will be used. Given that every argument of a term innermost reduces at root position to the argument of a reduced term by non-root reductions (lemma `non_root_rtc_reduction_of_argument` in theory `innermost_reduction.pvs`), the subterms of each element of this derivation at the chosen argument position is used to the first portion of the infinite sequence. Finally, the function `nth_index` is used to extract from sequence *seqt* those indexes where reductions occur in the selected argument, keeping this argument intact whenever the reduction does not occur in such indexes (result given in lemma `argument_protected_in_non_nth_index`). Then, for each term obtained by a reduction on the l-th argument on this (now infinite) derivation, its subterm at argument l is used to build the second and final portion of the infinite sequence.

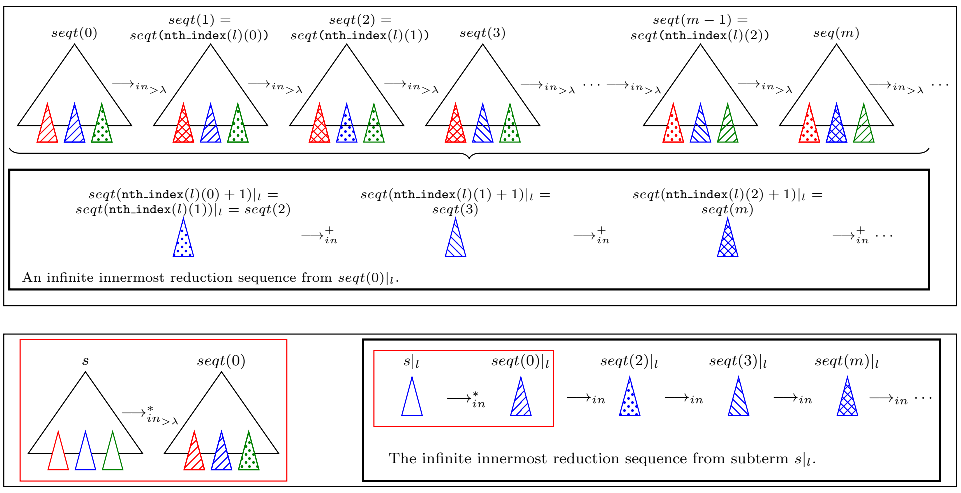

$seqt(0)$
$seqt(1) =$ $seqt(\texttt{nth\_index}(l)(0))$
$seqt(2) =$ $seqt(\texttt{nth\_index}(l)(1))$
$seqt(3)$
$seqt(m-1) =$ $seqt(\texttt{nth\_index}(l)(2))$
$seq(m)$
$\longrightarrow_{in_{>\lambda}}$
$seqt(\texttt{nth\_index}(l)(0)+1)|_l =$ $seqt(\texttt{nth\_index}(l)(1))|_l = seqt(2)$
$seqt(\texttt{nth\_index}(l)(1)+1)|_l =$ $seqt(3)$
$seqt(\texttt{nth\_index}(l)(2)+1)|_l =$ $seqt(m)$
$\longrightarrow^{+}_{in}$
An infinite innermost reduction sequence from $seqt(0)|_l$.
s
$seqt(0)$
$\longrightarrow^{*}_{in_{>\lambda}}$
$s|_l$
$seqt(0)|_l$
$\longrightarrow^{*}_{in}$
$seqt(2)|_l$
$seqt(3)|_l$
$seqt(m)|_l$
$\longrightarrow_{in}$
The infinite innermost reduction sequence from subterm $s|_l$.

Fig. 3 Proof intuition: building an infinite innermost derivation of an argument l as concatenation of a finite and an infinite non-root innermost derivation of terms.

5.2.2 Construction of Non-root Innermost Normal Forms for mint *terms*

Since a *mint* term s is noetherian regarding $\underset{s}{\longrightarrow}\ _{in_{>\lambda}}$, as previously shown, in an infinite derivation starting from s there exists an index where the first innermost reduction in the root position occurs. This result is formalized in lemma **`inf_inn_deriv_of_mint_has_min_root_reduction_index`**.

```
inf_inn_deriv_of_mint_has_min_root_reduction_index: LEMMA
 ∀(seq: sequence[term]):
   (↑in (seq(0)) ∧ ∀(i:nat): innermost_reduction?(E)(seq(i), seq(i+1))) ⟹
     ∃(j : nat) : seq(j) →λin seq(j+1) ∧
     ∀(k : nat) : seq(k) →λin seq(k+1) ⟹ k >= j
```

This lemma is formalized by providing as the first index required the minimum index of the infinite derivation where the reduction takes place at root position. The function minimum (**`min`**) of PVS, just as function **`choose`**, also requires a proof of non-emptiness of the set used as parameter. With the noetherianity provided by lemma **`mint_is_nr_inn_terminating`**, this non-emptiness constrain in obtained through an auxiliary result over noetherian relations restricted to an initial element that are subset of some non noetherian relation, which is given by lemma **`non_noetherian_and_noetherian_rest_subset`** in the **`restricted_reduction.pvs`** theory. This lemma provides then, an index of this infinite derivation whereas the given relation, i.e., $\underset{s}{\longrightarrow}\ _{in_{>\lambda}}$ does not hold.

Notice that, until this point, some infinite reduction sequence is being considered in the proof. However, the DPs are not extracted from the whole terms in this derivation. Instead, a *mint* term is innermost reduced until reaching an innermost normal form and then the rule applied to the root builds the DP. Thus, at this point, the extraction of the DP would be possible. But since the instance of this DPs is crucial for building an infinite chain, it is important to know that, not only the term that initiated the infinite derivation will be at some point reduced at root position, but which was exactly the term reached before such reduction.

In order to be able to extract the DP and substitution required to proceed with the proof, one obtains finally that every *mint* term non-root innermost derives into a term that has its arguments in normal form.

```
mint_reduces_to_int_nrnf_term : LEMMA
  ∀(s| ↑_in (s)) : ∃(t| ↑_in (t)) : s →*_{in>λ} t ∧ nf_{→>λ}(t)
```

The proof follows as an application of previous lemma, choosing the term at the index where the first reduction at root position takes place, since this term is in innermost normal form. Indeed, this term will be a normal instance of the *lhs* of some rule.

5.3 Existence of DPs

The term obtained in previous subsection is an innermost non terminating term such that it is also non-root innermost normalized. Such kind of non-root normalized terms should innermost reduce at root position, see $\xrightarrow{\lambda}_{in}$-reductions in Figure 2. These reductions from vertically to diagonally striped triangles give rise to the desired DPs. An important observation is that such terms reduce at root position with a rule and a normal substitution. The substitution should be normal since the terms are non-root innermost normal forms.

The following key auxiliary lemma provides the important result that such normal instances of *rhs*'s of rules applied as before and that have minimal innermost non terminating subterms give rise to dependency pairs. The innermost non terminality of the terms will guarantee the existence of such subterms.

```
normal_inst_of_rule_with_mint_on_rhs_gives_dp_alt : LEMMA
  ∀(e ∈ E, σ : (normal_sub?(E)), π ∈ Pos(rhs(e)σ)) :
    ↑_in (rhs(e)σ|_π) → dep_pair_alt?(E)(e, π)
```

The proof only requires to show that $rhs(e)|_\pi$ is defined. For this, initially it must be ensured that π is indeed a non variable position of $rhs(e)$. But σ is normal, thus, since the premise $\uparrow_{in} (rhs(e)\sigma|_\pi)$ implies innermost reducibility of $rhs(e)\sigma|_\pi$, if π were a variable position or a position introduced by this substitution, there would be a contradiction to its normality. This part of the result is formalized separately in lemma `reducible_position_of_normal_inst_is_app_pos_of_term` that states that reducible subterms of normal instances of terms appear only at non variable positions of the original term. Then, by the main result of the last subsection, i.e., lemma `mint_reduces_to_int_nrnf_term`, one has that $rhs(e)\sigma|_\pi \longrightarrow^*_{in_{>\lambda}} t$ for some term t such that $\uparrow_{in} (t)$ and $nf_{\longrightarrow_{>\lambda}}(t)$. Then, the term t has a defined symbol on its root. Thus, it only remains to prove that the root symbol of $rhs(e)\sigma|_\pi$ and t is the same, which is an auxiliary result formalized by induction on the length of the non-root (innermost) derivation in corollary `non_root_ir_preserves_root_symbol` for non-root innermost derivations.

5.4 Construction of Chained DPs

So far the existence of the elements needed to the proof was formalized. Now, one builds in fact the elements as in Figure 2. Initially, a *mint* term is non-root innermost normalized through the function below. The existential result given in subsection 5.2 allows the use of the PVS `choose` operator.

```
mint_to_int_nrnf$(E)(s : term| \uparrow_{in} (s))$ : term =
  choose$(\{t$ :term$|s \longrightarrow^{*}_{in_{>\lambda}} t) \wedge nf_{\longrightarrow_{>\lambda}}(t) \wedge \uparrow_{in} (t)\})$
```

Since this new non-root innermost normalized term is also innermost non terminating, there exist some rule and normal substitution allowing to innermost reduce this term at its root. Furthermore, the term obtained from this reduction will be also innermost non terminating, i.e., it will have a *mint* subterm at some position of the *rhs* of the used rule. This property is formalized in lemma **`reduced_nit_nrnf_has_mint`** specified as below.

```
reduced_nit_nrnf_has_mint : LEMMA
  $\forall(s$ :term$| \uparrow_{in} (E)(s))$ :
    $\exists(\sigma$ :Sub$, e$ :rewrite_rule $| e \in E, \pi \in Pos(rhs(e)))$ :
      $lhs(e)\sigma =$ mint_to_int_nrnf$(E)(s) \wedge \uparrow_{in} (rhs(e))\sigma|_{\pi})$
```

This lemma is formalized applying the existential results of Subsection 5.3 for obtaining the normal substitution σ and the rule e and, of Subsection 5.1 to obtain a position π such that $\uparrow_{in} (rhs(e))\sigma|_{\pi})$.

Lemma **`reduced_nit_nrnf_has_mint`** allows one to use `choose` to pick the rule and position leading to the DP and the substitution that will allow to chain the DP with the next DP originated from the *mint* term $rhs(e)\sigma|_{\pi}$ as specified in function **`dp_and_sub_from_int_nrnf`** below. Here is clear why this construction is facilitated by the use of the alternative definition of DPs that includes both the rule and the position.

```
dp_and_sub_from_int_nrnf$(E)(s$ :term$| \uparrow_{in} (s))$ : [dep_pair_alt$(E)$, Sub] =
 LET $sub\_e\_p$ =choose$(\{(\sigma$ :Sub$, e \in E, \pi \in Pos(rhs(e)))$ |
                            $lhs(e)\sigma =$ mint_to_int_nrnf$(E)(s), \uparrow_{in} (rhs(e))\sigma|_{\pi})\})$
 IN $((sub\_e\_p'2, sub\_e\_p'3), sub\_e\_p'1)$
```

Whenever this function has as input a term that is an instance of the *rhs* of a DP that is in non-root innermost normal form, the resulting DP and substitution will be chained with the DP and substitution used to build the input term. This result is specified in lemma **`next_inst_dp_is_inn_chained_and_mnt`** given below, where the desired alternative DPs are transformed into standard DPs in order to allow the analysis through predicate **`inn_chained_dp?`**:

```
next_inst_dp_is_inn_chained_and_mnt : LEMMA
  $\forall(E)($ $dp$ : dep_pair_alt$(E)$,
        $\sigma$ :Sub | $\uparrow_{in} (rhs(dp'1)\sigma|_{dp'2}))$ $\wedge$ $nf_{\longrightarrow_{>\lambda}}(lhs(dp'1)\sigma)$ ) :
     LET $std\_dp = (lhs(dp'1),rhs(dp'1)|_{dp'2})$ ,
         $next\_dp\_sub$ = dp_and_sub_from_int_nrnf$(E)(rhs(dp'1)\sigma|_{dp'2})$ ,
         $next\_std\_dp = (lhs(next\_dp\_sub'1'1),rhs(next\_dp\_sub'1'1)|_{next\_dp\_sub'1'2})$,
         $\sigma' = next\_dp\_sub'2$ IN
       inn_chained_dp?$(E)(std\_dp, next\_std\_dp)(\sigma, \sigma')$ $\wedge$ $\uparrow_{in} ((next\_std\_dp'2)\sigma')$
```

The formalization of this lemma is quite simple in its core, however, since transformations between the standard and alternative notions of DPs are used, the proof of some typing conditions are required in order to ensure type correctness. Once circumvented the typing issues, one must only guarantee the innermost chained property for the input DP and substitution and the resulting DP and substitution created and that the instantiated subterm of the *rhs* of the new DP is a *mint* term. Notice that the latter property is a direct result of the type of the PVS `choose` operator used in function **`dp_and_sub_from_int_nrnf`**; indeed, this property was included (and formalized) as part of this lemma just to avoid the necessity to repeatedly ensure non-emptiness of the used set providing more efficiency since this result is used several times throughout the rest of the formalization. To guarantee that the DPs are chained is also straightforward, since **`dp_and_sub_from_int_nrnf`** is defined over **`mint_to_int_nrnf`**, which gives a term with type as a non-root innermost normal form of the *mint* input, i.e., exactly the definition given by predicate **`inn_chainned_dp?`**; using notation of the lemma: $rhs(dp'1)|_{dp'2}\sigma \longrightarrow^*_{in_{>\lambda}} lhs(next\_dp\_sub'1'1)\sigma'$.

This result allows the specification of a function using predicate subtyping, a very interesting feature available in PVS. Using this feature elaborated predicate types can be assigned to the outputs of functions, and type checking will automatically generate the *type checking conditions* (TCCs) to ensure well-definedness of the function. Although used in other functions through the formalization, the most interesting application of this feature happens in the next function that outputs a pair for an input pair of DP and substitution, and where the type of the output uses the predicates **`in_chained_dp?`** and $\uparrow_{in}$. The generated TCCs are not proved automatically; however, to ensure that the type predicates hold, typing provided in the previous lemma are applied.

```
next_dp_and_sub($E$)( $dp$ : dep_pair_alt($E$),
                              $\sigma$ : Sub | $\uparrow_{in}$ $(rhs(dp'1)\sigma|_{dp'2})$ $\wedge\, nf_{\longrightarrow_{>\lambda}}(lhs(dp'1)\sigma)$ ) :
  { ($next\_dp$ :  dep_pair_alt($E$),
    $next\_\sigma$ : Sub)  |  inn_chained_dp?($E$)($dp, next\_dp$)($\sigma, next\_\sigma$) $\wedge$
                        $\uparrow_{in}$ $(rhs(next\_dp'1)next\_\sigma|_{next\_dp'2})$) } =
         dp_and_sub_from_int_nrnf($E$)$(rhs(dp'1)\sigma)|_{dp'2}$)
```

Applying **`dp_and_sub_from_int_nrnf`** (as in the body of the above function) to a *mint* term built from a pair of DP and substitution, one provides as output a pair of DP and substitution with the specified subtyping predicates, guaranteeing that the input and output are chained.

5.5 Construction of the Infinite Innermost Dependency Chain

With the possibility of creating new DPs and substitutions from *mint* terms, it is possible to build, inductively, an infinite DP chain from any innermost non terminating term. However, PVS syntax makes this construction a little bit tricky, since its functional language only allows directly construction of lambda-style or recursive functions. A lambda-style function to create such infinite chain is not possible, since the construction of every pair of DP and substitution depends on the previous one in the chain. But a direct construction of a recursive function is also problematic since the use of the `choose` operator in several steps of this construction makes it difficult to guarantee its determinism and then its functionality.

A simple solution for this problem is to use the recursion theorem to provide the existence of a function from naturals to pairs of a DP and a substitution such that each pair generates the next pair in the chain according to the function `next_dp_and_sub` implying then that contiguous images are chained.

The recursion theorem is specified as below. It states that for all predicates X over a set T, initial element a in X and function f over elements of X, there exists a function u from naturals to X such that the images of u are given by the sequence $a, f(a), \ldots, f^n(a), \ldots$.

```
recursion_theorem : THEOREM
  ∀(X :set[T], a ∈ X, f : [(X)− > (X)]) :
    ∃(u : [nat− > (X)]) : u(0) = a ∧ ∀(n :nat) : u(n + 1) = f(u(n))
```

To use this theorem, the predicate is instantiated with pairs of DP and substitution of the type of the parameters of the function `next_dp_and_sub`, i.e.,

$$(\ dp : \mathrm{dep\_pair\_alt}(E), \sigma : \mathrm{Sub} \mid \ \uparrow_{in} (rhs(dp'1)\sigma|_{dp'2}) \ \wedge nf_{\longrightarrow_{>\lambda}}(lhs(dp'1)\sigma) \ )$$

The fist element of the sequence a is instantiated as the pair of DP and substitution, obtained from the initial term starting any infinite innermost derivation, according to the techniques given is subsections 5.1, 5.2 and 5.3. As expected, the function from pairs to pairs is chosen as `next_dp_and_sub`. The recursion theorem guarantees just the existence of a total function from naturals to the sequence inductively built using function `next_dp_and_sub` starting from the initial pair. But the choice of this function assures by its predicate subtyping that each pair of consecutive pairs are in fact chained.

As a consequence of all that, the sufficiency lemma below is obtained.

```
dp_termination_implies_noetherian : LEMMA
  ∀(E) : inn_dp_termination?(E) ⟹noetherian?(⟶_in)
```

6 Related Work

There are several methods of semi-decision to address the analysis of termination, among them, the well-known *Ranking functions* implemented in PVS as termination TCCs, as mentioned in the introduction. A more recent criterion to verify termination of functional programs is the so called *size-change principle* (SCP, for short) [LJBA01]. This principle does not require decreasingness after each recursive call, but strict decreasingness (using a measure regarding some well-founded order) for each possible infinite “cycle” of recursive calls; thus, if such a measure exists, infinite computations are not possible since they will imply infinite decreasingness (over a well-founded order). The SCP and DPs criterion are compared in [TG05] taking into account termination, innermost termination and evaluation of functional specifications. One approach of the SCP is given by the technology of *calling contexts graphs* (CCG, for short) [MV06], which implements the SCP by representing all possible executions of a functional program as paths in a graph in which nodes are labeled by the different occurrences of function calls in it. More precisely, each node corresponds to a so called *calling context* that consists of the formal parameters of a function in which a function call is specified, the actual parameters of the function call, and the conditions that lead to the execution of the function call. Possible computations are then characterized as sequences of calling contexts related with paths in that graph, and termination is analyzed regarding the behavior of measurements on the possible circuits in the CCG. The CCGs technology has the advantage of allowing combinations of a finite family of measures at each node of a possible circuit, simplifying in this manner the formulation of a single and complex measure that works (decreases) for all possible circuits. These combinations are also implemented in the so called Matrix Weighted Graphs developed by Avelar in [Ave14]. All these technologies to verify termination are implemented and formalized to be equivalent in the PVS library `PVS0` mentioned in the introduction. The innermost DP termination criterion is formally related with noetherianity of the relation of chained DPs, which is relevant to verify termination of TRSs providing well-founded orderings on DPs, and this technology is related in PVS0 with CCGs for formalization of equivalence between termination criteria for functional programs.

Formalizations of the theorem of soundness and completeness of DPs (for short DP theorem) are available in several proof assistants. In [BK11], Blanqui and Koprowski described a formalization of the DP theorem for the standard reduction relation that is part of the CoLoR library developed in Coq for certifying proofs of termination. The formalized result is the DP theorem for the standard reduction relation, and not for the innermost termination. The proof in [BK11], as the current formalization, uses the non-root reduction relation (internal reduction) and the reduction at root position relation (head reduction), but instead of building infinite chains from infinite derivations, it assumes a well-founded relation over the set of chained DPs to conclude noetherianity of the standard reduction relation. Also, the library Coccinelle [CCF$^+$07] includes a formalization in Coq of DP theorem that defines a relation between instances of *lhs* of DPs and proves the equivalence between well-foundedness of this relation and well-foundedness of the reduction relation of a given TRS. To chain DPs instances of the lists of arguments of *lhs*'s and *rhs*'s of DPs, which are headed by the same function symbol, are related by the reflexive-transitive closure of the rewriting relation (avoiding in this way the use of tuple symbols). The formalization also considers a refinement of the notion of DPs, which avoids DPs generated by a rule, where the *rhs* of the DP appears also as a subterm of the *lhs* of the rule.

Such a formalization of DP theorem for the standard reduction relation, also is present in the proof assistant Isabelle, as part of the library for rewriting IsaFoR briefly described in [Thi10]. In this formalization the original signature of the TRS is extended with new tuple symbols for substituting the defined symbols (see comments after Definition 1 of DPs), which implies the analysis of additional properties of the new term rewriting system induced over the extended signature and also properties relating this new rewriting system with the original one. The proof, as in the current formalization, builds an infinite chain from an infinite derivation and vice-versa. The interesting features from this work are that it uses the same refinement of DPs as the formalization in Coccinelle and that it was done for a full definition of "Q-restricted" rewriting providing in this manner a general result that has as corollaries both the DP theorem for the standard and the innermost reduction relations, the former given explicitly. Essentially, for TRSs E and Q, the Q-restricted relation, denoted as $\xrightarrow{Q}_E$, is defined as the relation such that $s \xrightarrow{Q}_E t$ iff $s \longrightarrow_E t$ at some position π such that proper subterms of $s|_\pi$ are normal regarding Q; so $\xrightarrow{\emptyset}_E$ and $\xrightarrow{E}_E$ correspond respectively to the standard and the innermost reduction relations [GTSK05]. This formalization is used to provide a sound environment to certify concrete termination proofs in an automatic way by the tool CeTA [TS09].

7 Discussion and Future Work

A formalization in PVS of the soundness and completeness of the Dependency Pairs criterion for innermost termination of TRSs was presented. The formalization follows the lines of reasoning of proofs given in papers such as [AG00]. In particular, it builds infinite innermost chains of DPs from infinite innermost derivations and vice-versa.

The kernel of the formalization consists of 64 lemmas from which 40 are TCCs. This is available in the specification and formalization files `dependency_pairs.pvs` and `.prf` that have size 21KB and 871KB, respectively. For achieving the formalization, the `TRS` library of PVS, was extended with theories `innermost_reduction` and `restricted_reduction`, which include 37 lemmas from which 17 are TCCs. Both these theories add 10KB of specification and 451 KB of proofs. The proof of necessity (in theory `dependency_pairs`) required 11% of the whole size of the formalization file, while sufficiency required 80%. The remaining 9% of the formalization file deals with basic properties of DPs, and a lemma relating innermost DP termination with noetherianity of the innermost chain relation. From the total size used in the proof of sufficiency, the formalization was split approximately into 11%, 59%, 6%, 20% and 4% for the tasks presented in Section 5: Existence of *mint* Subterms (Subsection 5.1), Non-root Innermost Normalization of *mint* Terms (5.2), Existence of DPs (5.3), Construction of chained DPs (5.4), Construction of the Infinite Innermost Dependency Chain (5.5), respectively. As expected from the discussion in Section 5, the formalizations of normalization of *mint* terms and constructions of chained DPs were the most elaborated and the ones that required more space.

In order to formalize the DP theorem for the standard rewriting relation, a similar reasoning than the one used for innermost reduction can be followed, but the involved properties cannot be reused to prove each other. In fact, notice for instance that necessity for the standard reduction, i.e., `noetherian?(reduction?(E))` implies `dp_termination?(E)`, cannot be applied to infer `inn_dp_termination?(E)`, if one has `noetherian?(innermost_reduction?(E))`. The required properties should be developed explicitly for the standard reduction relation, as done so far for the formalization of necessity theorem. An important difference happens in the formalization of sufficiency, when one is building an infinite chain from an infinite derivation. Specifically, for the innermost case, *mint* terms are normalized regarding the non-root innermost relation, giving rise to a term that has an innermost reduction redex at its root (vertically striped small triangles in Fig. 2), while for the standard case, the unique guarantee it that *mnt* terms reduce at non-root positions into a term that can be reduced at its root position. This small difference implies a few adjusts in order to apply the rules on root position leading to the DPs that will produce the chain. The existence of DPs from such (non necessarily non-root normalized) terms follows from an argumentation based on the fact that the *mnt* term starting the non-root derivation is non-root terminating; thus, when a given rule is applied at root position of some of its non terminating descendants, the substitution allowing the application of such rule may not have non-root non-terminating redexes. Other than that, the `chained_dp?` property also follows directly from the type of the chosen descendant term of a *mnt* term where the first root reduction takes place.

References

AG97. Thomas Arts and Jürgen Giesl. Automatically proving termination where simplification orderings fail. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 1997.

AG98. Thomas Arts and Jürgen Giesl. Modularity of termination using dependency pairs. In Tobias Nipkow, editor, *Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 1998.

AG00. Thomas Arts and Jürgen Giesl. Termination of term rewriting using Dependency Pairs. *Theoretical Computer Science*, 236:133–178, 2000.

Ave14. Andréia Borges Avelar. *Formalização da Automação da Terminação Através de Grafos com Matrizes de Medida.* PhD thesis, Universidade de Brasília, 2014.

BK11. Frédéric Blanqui and Adam Koprowski. Color: a coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Math. Struct. in Comp. Science*, 21:827–859, 2011.

BN98. Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

CCF$^+$07. Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. In *International Symposium on Frontiers of Combining Systems*, pages 148–162. Springer, 2007.

GAR08. André Luiz Galdino and Mauricio Ayala-Rincón. A Formalization of Newman's and Yokouchi's Lemmas in a Higher-Order Language. *Journal of Formalized Reasoning*, 1, 2008.

GAR10. André Luiz Galdino and Mauricio Ayala-Rincón. A Formalization of the Knuth–Bendix(–Huet) Critical Pair Theorem. *Journal of Automated Reasoning*, 45(3):301–325, 2010.

GTSK05. Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The Dependency Pair Framework: Combining techniques for automated termination proofs. In Franz Baader Andrei and Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning: 11th International Conference, LPAR 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331. Springer, 2005.

LJBA01. Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The Size-change Principle for Program Termination. In *ACM SIGPLAN Notices*, pages 81–92. ACM, 2001.

MV06. Panagiotis Manolios and Daron Vroon. Termination Analysis with Calling Context Graphs. In *Proceedings of the 18th International Conference on Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2006.

RMAR$^+$18. Thiago Mendonça Ferreira Ramos, César Muñoz, Mauricio Ayala-Rincón, Mariano Moscato, Aaron Dutle, and Anthony Narkawicz. Formalization of the undecidability of the Halting Problem for a functional language. In *Workshop on Logic, Language, Information, and Computation WoLLIC*, volume 10944 of *LNCS*. Springer Nature, 2018. In press.

ROGAR17. Ana Cristina Rocha-Oliveira, André Luiz Galdino, and Mauricio Ayala-Rincón. Confluence of Orthogonal Term Rewriting Systems in the Prototype Verification System. *Journal of Automated Reasoning*, 58(2):231–251, 2017.

TG03. René Thiemann and Jürgen Giesl. Size-change Termination for Term Rewriting. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2003.

TG05. René Thiemann and Jürgen Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Appl. Algebra Eng. Commun. Comput.*, 16(4):229–270, 2005.

Thi10. Signature extensions preserve termination - an alternative proof via dependency pairs. In *19th EACSL Annual Conferences on Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 514–528, 2010.

TS09. René Thiemann and Christian Sternagel. Certification of Termination Proofs Using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.

Tur36. Alan M. Turing. On computable numbers with an application to the "Entscheidungsproblem". *Proceeding of the London Mathematical Society*, 1936.

Tur49. Alan Turing. Checking a large routine. In *Report of a Conference High Speed Automatic Calculating-Machines*, pages 67–69. University Mathematical Laboratory, 1949.