Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 247 (2009) 67–83
www.elsevier.com/locate/entcs

A PVS *Theory* for Term Rewriting Systems

André L. Galdino[1,2]

Grupo de Teoria da Computação, Departamento de Matemática, Universidade de Brasília
Brasília D.F., Brazil &
Departamento de Matemática, Universidade Federal de Goiás
Catalão, Brazil

Mauricio Ayala-Rincón[1,3]

Grupo de Teoria da Computação, Departamento de Matemática, Universidade de Brasília
Brasília D.F., Brazil

Abstract

A *theory*, called `trs`, for Term Rewriting Systems in the theorem Prover PVS is described. This *theory* is built on the PVS libraries for finite sequences and sets and a previously developed PVS *theory* named `ars` for Abstract Reduction Systems which was built on the PVS libraries for sets. *Theories* for dealing with the structure of terms, for replacements and substitutions jointly with `ars` allow for adequate specifications of notions of term rewriting such as critical pairs and formalization of elaborated criteria from the theory of Term Rewriting Systems such as the Knuth-Bendix Critical Pair Theorem. On the other hand, `ars` specifies definitions and notions such as reduction, confluence and normal forms as well as non basic concepts such as Noetherianity.

Keywords: Abstract Reduction Systems, Term Rewriting Systems, Formalization of Theorems, PVS.

1 Introduction

The Prototype Verification System (PVS), developed at the SRI and widely used by industrial and academic parties, consists of a specification language built on higher-order logic, which supports modularity by means of parameterized *theories*, with a rich type-system and a prover which uses the sequent-style. A PVS *theory*, `ars`, built on the PVS prelude libraries for sets and binary relations that is useful for the treatment of properties of Abstract Reduction Systems (ARS) was reported in [14]. In `ars` notions such as reduction, derivation, normal form, confluence, local

[1] Authors partially supported by the Brazilian Research Council CNPq. Work supported by the Brazilian Research Council CNPq and the District Federal Research Foundation FAP-DF under grants CNPq/DFG 490396/2007-0 and FAP-DF 8-004/2007.

[2] Email: galdino@unb.br

[3] Email: ayala@unb.br

doi:10.1016/j.entcs.2009.07.049

confluence, joinability, noetherianity, etc., were adequately specified in such a way that proofs by noetherian induction are possible. The usefulness of `ars` was made evident by formalizing proofs of the well-known Church-Rosser criterion, Newman's and Yokouchi's Lemmas, among others [15].

In this work we present `trs`, a PVS *theory* for Term Rewriting Systems (TRS). To the best of our knowledge there is no other PVS `theory` for TRS. The *theory* `trs` is built on the PVS libraries for finite sequences and the *theory* `ars`. The development includes `theories` for dealing with the structure of terms, replacements and substitution. It includes specifications of elaborated notions of term rewriting such as critical pairs which makes possible mechanical proofs of non trivial criteria such as the Knuth-Bendix Critical Pair Theorem [19].

The novelty of this work in not to present mechanical proofs of theorems of the theory of TRS in PVS, which were done previously in other proof assistants. In fact, formalization of equational reasoning by rewriting started almost twenty five years ago with the development of the *Rewrite Rule Laboratory* RRL, the first successful tool for equational deduction via rewriting [18]. Also, specifications of λ-calculus, abstract reduction and term rewriting systems with formalizations of the Church-Rosser Theorem and Newman's Lemma have been presented in several proof assistants; eg, Coq [17], Isabelle [26], Isabelle/HOL [23], Boyer-Moore [29], Otter [7], among others. In particular, the first complete formalization of the Knuth-Bendix Critical Pair Theorem was presented in [27]; this formalization was given in a first-order language and developed in the prover ACL2. Instead presenting `trs` as "another collection of mechanical proofs of rewriting theorems", we would like to present `trs` as an adequate formalization of term rewriting theory in general and as the basis for the formal manipulation of (equational) specifications based on rewriting systems in PVS.

We believe `trs` enriches the power of PVS by allowing rewriting proof techniques inside this proof assistant. The motivation for doing this formalization is that rewriting systems have been applied to the specification and synthesis of reconfigurable hardware [4,22] and that the correction of these specifications can be carried out by translating these rewriting specifications into the language of PVS as logical theories (in [5] it is introduced a proved correct translation from ELAN rewriting specifications into PVS theories). In general, except for techniques for the treatment of termination, `trs` provides proof rewriting based techniques that are necessary in order to formalize the correctness of rewriting specifications in the proof assistant PVS.

The distinguishing features of `trs` are listed below.

- Abstractness is one of the relevant characteristics of `trs`; in fact, based on the PVS *theory* for binary relations, confluence properties of ARSs are formalized in an "almost geometric style", which allows for a "diagrammatic" treatment of reduction and rewriting properties as it is usual in the standard rewriting literature (eg [15]) as it was done in [23] for proof-checking the Church-Rosser theorem of the λ-calculus in Isabelle/HOL.
- Difficulties with the use of variable names such as the necessity of considering

terms modulo α-conversion are eliminated in [23] by using de Bruijn notation. But since it is inconvenient to represent TRSs with indices instead variable names, `trs` includes elaborated *sub-theories* for dealing with variables, terms, replacements and substitutions in the standard way: with variable names and renaming substitutions.

- Other distinctive feature of `trs` is the use of the elaborated theory of types of PVS to represent TRS objects such as binary relations (functions in an abstract type `T`: `[T->T]`), substitutions (the subtype of functions from variables to terms: `[V -> term]`, whose domain is finite), etc. In this way, the specification of higher-order theorems is straightforward. In fact, as we will illustrate, in contrast to the first-order formalization of the Critical Pair Theorem in ACL2 presented in [27], `trs` brings formalizations of higher-order rewriting theorems in a natural and clear manner over the higher-order specification language of PVS.

Initially, Section 2 gives the necessary background on PVS and specification of basic abstract reduction notions. Afterwards, Section 3 describes the elements used in the specification of the *theory* `trs` and Section 4 illustrates the usefulness of `trs` by showing how the Knuth-Bendix Critical Pair Theorem was formalized. Finally, before concluding, Section 5 presents related work.

The *theory* `trs` is available at `www.mat.unb.br/~ayala/publications.html`.

2 Specification of basic reduction notions in PVS

We suppose the reader is familiar with rewriting theory and its standard notations as presented in well-known textbooks (eg [6,8]).

2.1 PVS

PVS consists of a specification language integrated with support tools and a theorem prover, that provides an integrated environment for the development and analysis of formal specifications. Only the relevant aspects of PVS are explained here. For more details about this system, refer to the documentation available at `http://pvs.csl.sri.com`.

The *specification language* of PVS is built on higher-order logic, which supports modularity by means of parameterized *theories*, with a rich type-system, including the notions of subtypes and dependent types. It provides a large set of built-in constructs for expressing a variety of notions. The PVS specifications are organized as a collection of *theories*, from which the most relevant ones are collectively referred as the `prelude`. Each *theory* is composed essentially of *declarations*, which are used to introduce names for types, constants, variables, axioms and formulas, and `IMPORTING`s, which allow to import the visible names of another *theories*. Notice that parameterized *theories* are very convenient since the use of parameters allows more generic specifications, as we can see with the `ars` PVS *theory* below:

```
ars[T : TYPE] : THEORY
BEGIN
```

```
  IMPORTING   results_commutation[T],   modulo_equivalence[T],
              results_normal_form[T],   newman_yokouchi[T]
END ars
```

Within the `ars` *theory*, `T` is treated as a fixed uninterpreted type. So, when `ars` is used by another *theory* it must be instantiated. For example, the *theory* of `ars` of real numbers is just `ars[real]`.

A important step in PVS specifications is type-checking the *theory* that builds type-correctness conditions `TCC`s which are *proof obligations* that must be discharged before the *theory* can be considered type-checked. `TCC`s proofs may be postponed indefinitely, but the *theory* is considered *complete* only when all `TCC`s and formulas upon which the proofs are dependent have been completed.

The PVS *Prover* provides a variety of commands to construct the proofs of the different theorems. It is used interactively and it uses the sequent-style proof representation to display the current proof goal for the proof in progress. The prover maintains a proof tree for the current theorem being proved being the aim of the user to construct a proof tree that is complete, in the sense that all the leaves are recognized as `true`. Each node of the tree is a proof goal that results from the application of a prover command (*rule* or *strategy*) to its parent node.

2.2 Specification of basic abstract reduction notions

Figures 1 and 2 illustrate the hierarchy of *sub-theories* of the *theories* `ars` and `trs` respectively. Notice that `ars` makes part of `trs` and the Figure 1 is given separately for improving presentation only.

The complete `trs` development runs in PVS 4.2 and consists of 350 lemmas specified in 2745 lines (82K) and 50489 lines (3.4M) of proofs. PVS builds 124 TCCs whose proofs are included in the latter number. The number of lemmas corresponding to the *theory* `ars` is 65 from which 5 are TCCs only.

The *theory* `ars` imports the PVS library for sets (`sets_lemmas`) and over this it builds the closure of binary relations that are necessary for formalizing ARS theorems. Let consider a binary relation `R` over `T`, specified in PVS as `R: VAR pred[[T, T]]`. Its reflexive transitive closure, `RTC(R)`, is specified using the `iterate` function which allows us to obtain inductive proofs on the length of derivations:

```
RTC(R): reflexive_transitive = IUnion(LAMBDA n: iterate(R, n))
```

Formalizations of properties of the reflexive transitive closure are given as

```
R_subset_RTC: LEMMA subset?(R, RTC(R))

iterate_RTC: LEMMA FORALL n : subset?(iterate(R, n), RTC(R))

RTC_idempotent : LEMMA  RTC(RTC(R)) = RTC(R)

RTC_characterization : LEMMA   reflexive_transitive?(R) <=>
                               (R = RTC(R))
```

In the previous lemmas R is universally quantified. This applies for all unquantified variables in the lemmas and theorems to be presented in the remaining of the paper.

Other closure operators and their properties are formalized similarly: equivalence EC, symmetric SC, transitive TC, etc.

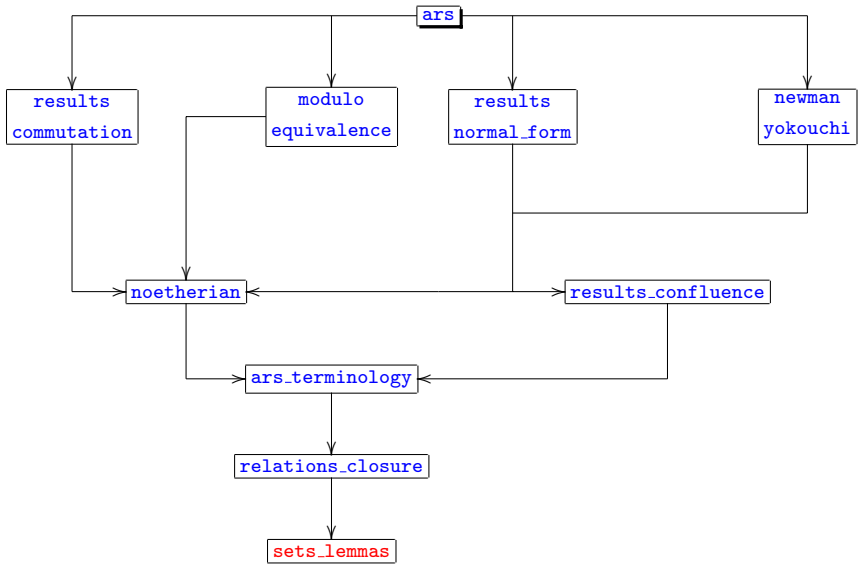


Fig. 1. Hierarchy of the ars *theory*

Basic abstract reduction notions such as *joinability*, *Church-Rosser* and *confluence* are defined in the PVS *sub-theory* ars_terminology as

```
ars_terminology[T : TYPE] : THEORY
BEGIN
  IMPORTING relations_closure[T]

    R : VAR PRED[[T, T]]
    x, y, z : VAR T
 ...
 joinable?(R)(x,y): bool = EXISTS z: RTC(R)(x,z) & RTC(R)(y, z)

 church_rosser?(R): bool = FORALL x, y:
                             EC(R)(x,y) =>
                             joinable?(R)(x,y)

 confluent?(R): bool = FORALL x, y, z:
                         RTC(R)(x,y) & RTC(R)(x,z) =>
                         joinable?(R)(y,z)
 ...
 END ars_terminology
```

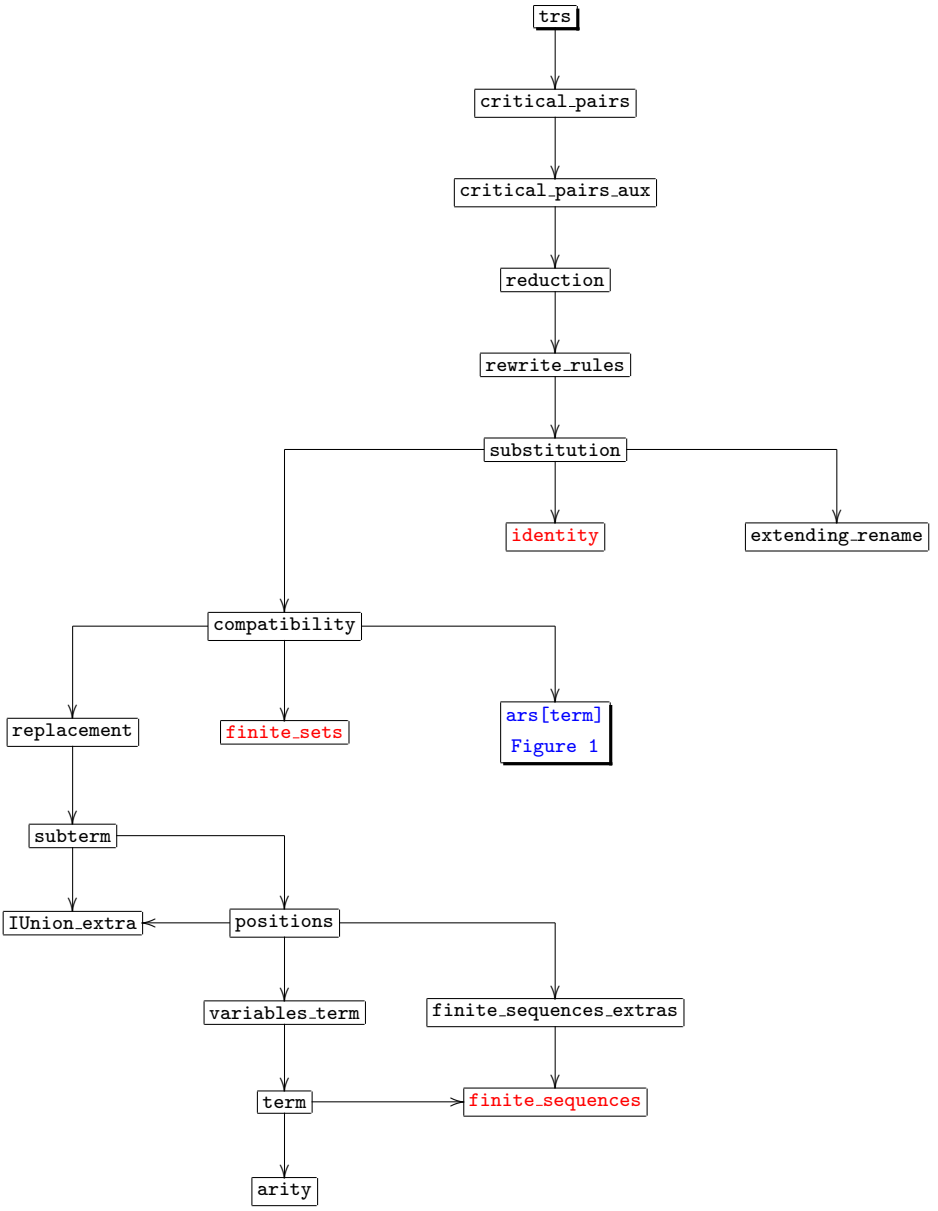


Fig. 2. Hierarchy of the `trs` *theory*

Basic abstract reduction results on confluence, for instance, are formalized (proved) in the PVS *sub-theory* `results_confluence`. The equivalence between Church-Rosser and confluence is specified as

```
CR_iff_Confluent: THEOREM church_rosser?(R) <=> confluent?(R)
```

In the *sub-theory* `noetherian` noetherian relations are specified based on the notion of well-founded relations and the principle of Noetherian induction is formalized (proved).

```
noetherian[T : TYPE] : THEORY
```

```
BEGIN

  IMPORTING ars_terminology[T],
            sets_aux@well_foundedness[T]

     P : VAR PRED[T]
     R : VAR PRED[[T, T]]
  x, y : VAR T

 noetherian?(R): bool = well_founded?(converse(R))
 ...
 noetherian_induction: LEMMA
             (FORALL (R: noetherian, P):
               (FORALL x:
                  (FORALL y: TC(R)(x, y) IMPLIES P(y))
                      IMPLIES P(x))
             IMPLIES
               (FORALL x: P(x)))

END noetherian
```

Using this formalization of noetherianity, the Newman's Lemma can be formalized (proved) elegantly as described in [15].

```
Newman_lemma: THEOREM FORALL R:
                       noetherian?(R)   =>
                            (confluent?(R) <=> local_confluent?(R))
```

3 Specification of term rewriting notions

The *theory* `trs` imports finite sequences and finite sets from the PVS libraries. Finite sequences are used to specify well-formed terms which are built from variables and function symbols with their associated arities. This is done by application of the PVS `DATATYPE` mechanism which is used to define recursive types.

```
term[variable: TYPE+, symbol: TYPE+] : DATATYPE
BEGIN

IMPORTING arity[symbol]

vars(v: variable): vars?
app(f:symbol,
    args:{args:finite_sequence[term] | args`length=arity(f)}): app?

END term
```

Notice that the well-formedness of terms, that is, the fact that function symbols are applied to the right number of arguments, is guaranteed by typing the arguments of each function symbol `f` as a finite sequence of length `arity(f)`. Also, finite sets and sequences are used to specify sets of subterms and sets of term positions. For instance, the (finite) set of positions of `t` where the variable `x` occurs is the finite set of finite sequences given as

```
Pos_var(t, x): set[positions?(t)] =
                                {p: positions?(t) | subtermOF(t,p)=x}
```

The *sub-theory* `replacement` formalizes the algebra of replacement of subterms of terms. `replaceTerm(t, s, p)` is the term which results from `s` replacing its subterm at position `p` by the term `t`. In standard rewriting notation this is written as $s[p \leftarrow t]$. Properties of this algebra of terms are easily proved. For instance,

Lemma 3.1 *Let s, t and r be terms, p be a position of s and q a position of t. Then*

$$s[p \leftarrow t][p.q \leftarrow r] = s[p \leftarrow t[q \leftarrow r]]$$

is formalized as

```
lemmaR4: LEMMA  positionsOF(s)(p) &
                positionsOF(t)(q)   =>
                        replaceTerm(r,replaceTerm(t,s,p),p o q) =
                        replaceTerm(replaceTerm(r,t,q),s,p)
```

The *sub-theory* `compatibility` formalizes the notion of a binary relation R that is compatible with the structure of terms, that is $R(r, s) \Rightarrow R(t[p \leftarrow r], t[p \leftarrow s])$:

```
comp_cont?(R): bool =
           (FORALL r, s:
               R(r,s)    =>
               R(replaceTerm(r, t, p), replaceTerm(s, t, p)))
```

Lemmas that state that the reflexive, transitive and equivalence closures of compatible relations are compatible as well are formalized too.

The *sub-theory* `substitution` specifies the algebra of substitutions. In this *sub-theory* notions such as domain, range, domain restriction, homeomorphic extension of substitutions and renaming substitutions are specified. The type of substitutions is built as functions from variables to terms `sig : [V -> term]`, whose domain is finite: `Sub?(sig): bool = is_finite(Dom(sig))` and `Sub: TYPE = (Sub?)`. The homeomorphic extension `ext(sig)` of a substitution `sig` is specified inductively over the structure of terms. In standard rewriting notation, the homeomorphic extension of a substitution σ from its domain of variables to the domain of terms is denoted as $\hat{\sigma}$, but to simplify notation, usually textbooks do not distinguish between a substitution σ and its extension $\hat{\sigma}$. In the formalization this distinction should be maintained carefully. For instance,

Lemma 3.2 *Let s and t be terms, p a position of s and σ a substitution. Then*

$$\sigma(s[p \leftarrow t]) = \sigma(s)[p \leftarrow \sigma(t)]$$

is formalized as

```
lemma6: ext(sigma)(replaceTerm(t,s,p)) =
        replaceTerm(ext(sigma)(t), ext(sigma)(s),p)
```

The *theory* `trs` does not include a *sub-theory* for first-order unification and the existence of most general unifiers is axiomatized.

In the *sub-theory* `rewrite_rules` term rewriting rules follow the usual restrictions:

```
rewrite_rule?(l,r): bool = (NOT vars?(l)) &
                           subset?(Vars(r), Vars(l))
```

The *sub-theory* `reduction` specifies the notion of reduction relation given as `reduction?(E)` and built from a term rewriting system, which is a set of rewriting rules `E`. Reduction relations are then proved to be closed under substitutions and compatible with operators (structure of terms):

```
subs_op: LEMMA close_subs?(reduction?(E)) &
               comp_op?(reduction?(E))
```

where a binary relation `R` closed under substitutions is specified as

```
close_subs?(R): bool = FORALL s, t, sigma:
                         R(s,t)  =>
                         R(ext(sigma)(s),ext(sigma)(t))
```

4 Formalizations (proofs) of term rewriting results

As illustration of formalizations of elaborated results from term rewriting theory we explain how the Knuth-Bendix Critical Pair Theorem was proved. We assume the reader familiar with the proof of this theorem (as presented in [16] or in well-known textbooks such as [6,8]). This theorem states that

Theorem 4.1 (Knuth-Bendix Critical Pair Theorem) *The reduction relation built from a term rewriting system is local confluent if, and only if all its critical pairs are joinable.*

As mentioned in the introduction, in the *theory* `trs` the use of variable names improves readability (in contrast to use of de Bruijn indices), but this implies additional work. In particular, this happens when specifying rewriting notions such as the one of critical pairs as presented in standard notation below.

Definition 4.2 [Critical Pair] Let $l_i \to r_i$, $i = 1, 2$ be rewriting rules whose variables have been renamed such that $\texttt{Vars}(l_1) \cap \texttt{Vars}(l_2) = \emptyset$. Let $p \in \texttt{positions?}(l_1)$ be such that $l_1|_p$ is not a variable and let $\sigma = mgu(l_1|_p, l_2)$. Then one says that

overlapping l_2 over l_1 at position p determines the *critical pair*

$$\langle\ \sigma(r_1),\ \ \sigma(l_1)[p \leftarrow \sigma(r_2)]\ \rangle$$

In the rewriting literature there is no explicit distinction between a set of rewriting rules (`E`) and the reduction relation (`reduction?(E)`). Informally, and only when necessary, as in the previous definition, some assumptions such as "suppose there are no variable names in common", "suppose it is a renaming with different variable names", etc. are given to avoid these problems. In `trs` this should be done explicitly by using renamings as in the formalization of critical pairs presented below. The set of critical pairs `CP?(E)` of a set of rewriting rules `E` is specified as:

```
CP?(E)(t1, t2): bool =
    EXISTS (sigma,
            rho,
            ((l1,r1) | member((l1,r1), E)),
            ((l2p,r2p) | member((l2p,r2p), E)),
            (p: positions?(l1))):
              LET (l2,r2) = (ext(rho)(l2p), ext(rho)(r2p)) IN
               disjoint?(Vars(l1),Vars(l2))                          &
               NOT vars?(subtermOF(l1, p))                           &
               mgu(sigma)(subtermOF(l1, p), l2)                      &
               t1 = ext(sigma)(r1)                                   &
               t2 = replaceTerm(ext(sigma)(r2), ext(sigma)(l1), p)
```

In this specification `rho` is a renaming substitution that guarantees that `(l1, r1)` and `(l2, r2)` are variants of rewriting rules without variables in common.

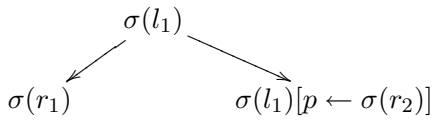
In the *sub-theory* `critical_pairs` (see Figure 2) the Knuth-Bendix Critical Pair Theorem is specified as:

```
CP_lemma: THEOREM  FORALL E:
          LET RRE = reduction?(E) IN
                              local_confluent?(RRE)
                                          <=>
            (FORALL t1, t2: CP?(E)(t1, t2) => joinable?(RRE)(t1,t2))
```

The *sub-theory* `critical_pairs` fully formalizes the proof of the Critical Pair Theorem (following the structure of the proof presented in [16]).

In the remaining of this section $\rightarrow$ denotes the reduction relation induced by the set of rules `E`, that is `reduction?(E)`.

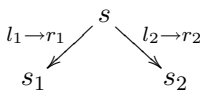
Necessity (`=>`): this is proved easily since all critical pairs are local divergences of the form



Then one concludes, by applying the hypothesis that the reduction relation is

locally confluent.

Sufficiency (<=): for the formalization of this part of the proof, let s be a term of divergence such that



that is, there are positions $p_i \in$ `positions?`(s), rules $l_i \to r_i \in$ `E`, and substitutions σ_i, such that $s|_{p_i} = \sigma_i(l_i)$ and $s_i = s[p_i \leftarrow \sigma_i(r_i)]$, for $i = 1, 2$.

One should prove that s_1 and s_2 are joinable. The proof is divided in three cases according to the manner in which the local divergence is generated: the case of divergence by reduction of terms at separate or parallel positions, that is, $p_1 \parallel p_2$ in standard notations; and the two cases of divergence by reduction of overlapping terms, the first, in which one has an instance of a critical pair, called a critical overlap and, the second, in which $\sigma_2(l_2)$ does not overlap with l_1 itself, called non-critical overlap.

Case 1 Suppose $p_1 \parallel p_2$. The formalization is obtained according to the following steps: firstly, by a lemma of persistence one obtains that $s_1 |_{p_2} = \sigma_2(l_2)$ and that $s_2 |_{p_1} = \sigma_1(l_1)$; secondly, by a lemma of commutativity, one obtains that $s_1[p_2 \leftarrow \sigma_2(r_2)] = s_2[p_1 \leftarrow \sigma_1(r_1)]$. Consequently, s_1 and s_2 are joinable.

Case 2 Suppose that p_1 and p_2 overlap, that is $p_1 \not\parallel p_2$. Then, either $p_1 \leq p_2$ or $p_2 \leq p_1$. Without lost of generality, one assumes that $p_2 \leq p_1$, that is $p_2 = p_1 p$, for some p possibly empty. The other case is proved symmetrically.

One starts by establishing the following properties:

(i) $\sigma_1(l_1|_p) = \sigma_2(l_2)$; and

(ii) by distributivity, $s_2|_{p_1} = \sigma_1(l_1)[p \leftarrow \sigma_2(r_2)]$.

In the sequel, one proves that there exists a term s_3 such that $\sigma_1(r_1) \to^* s_3$ and $s_2|_{p_1} \to^* s_3$, where $\to^*$ denotes `RTC(→)`. Then, by the compatibility of the relation $\to$, one concludes that s_1 and s_2 are joinable. For doing this, the following two sub-cases are considered.

Case 2a Critical overlap: $p \in$ `positions?`(l_1), $l_1|_p$ is not a variable and $\sigma_1(l_1|_p) = \sigma_2(l_2)$. The proof is obtained by application of the lemma `CP_lemma_aux1` presented below, which states that the divergence $\sigma_1(r_1)$ and $s_2|_{p_1}$ corresponds to an instance of a critical pair $\langle t_1, t_2 \rangle$.

```
CP_lemma_aux1: LEMMA
 FORALL E, ((l1, r1) | member((l1, r1), E)),
           ((l2, r2) | member((l2, r2), E)), (p: position):
 ( positionsOF(l1)(p)                                          &
   NOT vars?(subtermOF(l1, p))                                 &
   ext(sg1)(subtermOF(l1, p)) = ext(sg2)(l2) )
     =>
      EXISTS t1, t2, delta:
       CP?(E)(t1, t2)                                          &
```

```
ext(delta)(t1) = ext(sg1)(r1)                                    &
ext(delta)(t2) = replaceTerm(ext(sg2)(r2), ext(sg1)(l1), p)
```

Since by hypothesis $\langle t_1, t_2\rangle$ is joinable, there exists a term t_3 such that $t_1 \to^* t_3$ and $t_2 \to^* t_3$. Consequently, by the lemma, there exists δ such that $\delta(t_1) = \sigma_1(r_1)$ and $\delta(t_2) = s_2|_{p_1}$ and defining s_3 as $\delta(t_3)$, the result follows because $\to$ is closed under substitutions.

In general the critical overlap case is proved in textbooks (eg [6]) by assuming that the rewriting rules $l_i \to r_i$ are renamed such that $\texttt{Vars}(l_1) \cap \texttt{Vars}(l_2) = \emptyset$. This assumption implicitly suggests the supposition that $Dom(\sigma_1) \cap Dom(\sigma_2) = \emptyset$ holds and that, consequently, the substitution $\sigma_3 = \sigma_1 \cup \sigma_2$ is well-defined. Thus, σ_3 is a unifier of the terms $l_1|_p$ and l_2. From these implicit assumptions, it is possible to conclude that the terms of the divergence are an instance of a critical pair. Although, for obtaining a mechanical proof these implicit assumptions are not possible. In the presented proof it was necessary to formalize the additional lemma `CP_lemma_aux1a` that states that such renaming exists. Observe that the condition $\texttt{Vars}(l_1) \cap \texttt{Vars}(l_2) = \emptyset$ is obtained renaming a unique rule.

```
CP_lemma_aux1a: LEMMA
 FORALL E, ((l1, r1) | member((l1, r1), E)),
           ((l2, r2) | member((l2, r2), E)), (p: position):
 ( positionsOF(l1)(p)                                            &
   NOT vars?(subtermOF(l1, p))                                   &
   ext(sg1)(subtermOF(l1, p)) = ext(sg2)(l2) )
       =>
        EXISTS alpha, rho:
         disjoint?(Vars(l1), Vars(ext(rho)(l2)))                 &
         ext(sg1)(subtermOF(l1, p)) = ext(comp(alpha, rho))(l2)
```

Case 2b Non-critical overlap: $p = q_1 q_2$, for q_2 possibly empty, such that q_1 is a position of variable in l_1 and $\sigma_2(l_2) = \sigma_1(l_1|_{q_1})|_{q_2}$.

Although this is the more difficult case of the proof, in textbooks it is presented diagrammatically without the necessary analytical details. The difficulties arise because the rewriting rules are not necessarily linear. Thus, several occurrences of the variable $l_1|_{q_1}$ are possible in both sides of the rule $l_1 \to r_1$, which makes difficult the proof of joinability. The formalization of this case uses thirteen auxiliary lemmas specified in the *sub-theory* `critical_pairs_aux`. The following lemma as presented in [16] has a central role.

Lemma 4.3 *Let $\to$ be a relation compatible with the structure of terms, x be a variable, and σ_1 and σ_2 be substitutions such that:*

$$\begin{aligned} \sigma_1(x) &\to \sigma_2(x) \text{ and} \\ \sigma_1(y) &= \sigma_2(y), \text{ for all } y \neq x. \end{aligned}$$

Let t be an arbitrary term, and $p_1, \ldots, p_n \in \texttt{positions?}(t)$ be all the occurrences of x in t. Define $t_0 = \sigma_1(t)$ and $t_i = t_{i-1}[p_i \leftarrow \sigma_2(x)]$, for $1 \leq i \leq n$. Then $t_i \to^{n-i} \sigma_2(t)$, for $0 \leq i \leq n$. In particular, $\sigma_1(t) \to^n \sigma_2(t)$.

The formalization of this lemma requires two additional constructors called `replace_pos` and `RSigma` that are specified as presented below. `replace_pos` receives three arguments: two terms `t` and `s` and `fssp`, that is a sequence of parallel positions (`SSP`) of `s`. Recursively, it substitutes all subterms at these positions of `s` by `t`.

```
replace_pos(t, s, (fssp:SPP(s)) ): RECURSIVE term =
   IF length(fssp) = 0 THEN  s
   ELSE  replace_pos(t,replaceTerm(t, s, fssp(0)), rest(fssp))
   ENDIF
 MEASURE length(fssp)
```

`RSigma` is a boolean operator that holds for relations `R`, substitutions `sg1` and `sg2` and variable `x`, whenever the hypothesis of Lemma 4.3 holds, that is, except for `x`, `sg1` and `sg2` have identical images and `sg1(x)` reduces via `R` into `sg2(x)`.

```
RSigma(R, sg1, sg2, x): bool =
   FORALL (y: (V)): IF y /= x  THEN sg1(y) = sg2(y)
   ELSE R(sg1(x), sg2(x))
   ENDIF
```

Then, Lemma 4.3 can be formalized as

```
CP_lemma_aux2: LEMMA
  FORALL R, t, x, sg1, sg2:
   LET Posv = Pos_var(t, x), seqv = set2seq(Posv) IN
    comp_cont?(R) & RSigma(R, sg1, sg2, x)
     =>
      (FORALL (i: below[length(seqv)]):
        RTC(R)(replace_pos(ext(sg2)(x),ext(sg1)(t), #(seqv(i))),
               ext(sg2)(t)))                                    &
      RTC(R)(ext(sg1)(t), ext(sg2)(t))
```

In the specification of `CP_lemma_aux2`, `Pos_var(t,x)` is the set of all different positions of the variable `x` occurring in the term `t`, as given in the Section 3. This set is transformed into a sequence of positions with the operator `set2seq`. The operator `#(_)` constructs a unitary sequence with its argument.

The proof of the Knuth-Bendix Critical Pair Theorem required the formalization of sixteen specific auxiliary lemmas without taking into account general lemmas of the *theory* `trs`. The formalization of the theorem required 933 proof commands without taking into account commands used in the proof of the sixteen auxiliary lemmas.

Finally, it is important to remark that parts of the formalization of the Critical Pair Theorem are useful for mechanical proofs of other relevant non trivial TRS results such as confluence of orthogonal rewriting systems.

5 Related work

This section complements the discussion on related work started in the introduction.

In [17] Huet formalized properties involving confluence for the λ-calculus in Coq, in particular, for β-reduction. The main result is a formalization of the Prism Theorem (see theorem 5 in [30]). In [26], Rasmussen presented a translation to Isabelle of the treatment developed by Huet in Coq. In [24], Nipkow treated concepts such as confluence and commutation, and formalized in Isabelle/HOL [26] some results such as the theorems of the commutative union and the Church-Rosser theorems for β-, η- and $\beta \cup \eta$-reduction in the λ-calculus free of types. In [29], Shankar using the Boyer-Moore prover [10], formalized the Church-Rosser theorem for the λ-calculus. This formalization uses de Bruijn indices and the proof of the theorem is based on the approach of Tait-Martin-Löf, that is, in the notion of parallel reduction. In [25], Pfenning presented a formalization in LCF of the λ-calculus free of types, in which the Church-Rosser property is proved. Also, a formalization in PVS of the Church-Rosser theorem for a version of the λ-calculus *call-by-value* is presented by Ford and Mason in [13].

In [21] McKinna and Pollack presented a *survey* about concepts and results of the λ-calculus with pure types formalized in LEGO. Also, in [1] it was formalized in LEGO, by Altenkirch, the system F of Girard with the principal objective of verifying that such system is strongly normalizing. Another calculi formalized in Coq, with main objective to verify that they are strongly normalizing, are: the calculus of construction [11, 2], the λ-calculus typed with co-products [3] and the simple typed λ-calculus *à la* Church with constants [20].

The libraries *CoLoR* [9] and *Coccinelle* [12] developed in Coq, by Blanqui *et al* and Contejean *et al*, respectively, focused on formalizations of termination criteria by reduction orders, that was not considered in `trs`.

In [28], Saïbi presented specifications in Coq of concepts of the theory of rewriting, such as closure of relations and local confluence, and formalizations of some rewriting properties such as Newman's and Yokouchi's Lemmas. In addition, without proving the Knuth-Bendix theorem, critical pairs were analyzed for the calculus of explicit substitutions $\lambda_{\sigma\Uparrow}$. The Critical Pair Theorem is axiomatically assumed and applied in order to verify that this calculus is locally confluent.

Differently from the previously mentioned works, the *theories* `ars` and `trs` pretend to be more general trying to include all the elements that are necessary to formalize any property and result of the theory of rewriting, without focusing any rewriting system or rewriting calculus in particular.

In [27], Ruiz-Reina *et al* presented a first-order formalization in ACL2 of concepts and results from the theories of ARS and TRS. The work in [27] pretends, as the one presented here, to be a general formalization of the theory of rewriting. But in contrast to this work, `ars` and `trs` were developed in a natural manner using the higher-order language of PVS to represent the higher-order objects of the theory of rewriting. In particular, this straightforward and elegant representation of second-order objects such as reduction relations makes it possible the diagrammatic

treatment of properties such as confluence and commutativity as it is desirable and usual in the treatment of the theory of abstract reduction relations.

Also, in [27], Ruiz-Reina *et al* reported the first known complete formalization of the Knuth-Bendix Critical Pair Theorem. To the best of our knowledge, after Ruiz-Reina *et al* work no other formalization of this theorem was reported. Thus, the formalization of the Knuth-Bendix Critical Pair Theorem presented here should be the first one specified in higher-order language.

One of the main characteristics of the development presented in this work is the use of variable names instead variables as indices. Some of the works cited previously such as [29] and [17] used de Bruijn indices avoiding in this way the necessity of variable renamings. Other works such as [21] and [13] used variable names in their formalizations. Although, the use of indices is considered highly elegant and convenient, in particular de Bruijn notation is considered to be very adequate for implementations of the λ-calculus, its use results inconvenient for representing rewriting systems in general. The variable names approach adopted in the *theories* `trs` and `ars` allows representation of mathematical elements as they are presented in papers and textbooks.

6 Conclusions and Future Work

The PVS *theory* `trs` specifies adequately basic notions of the theory of TRSs. The *theory* `trs` is built on a *theory* for ARSs, `ars`, that was built on the PVS library for binary relations. The main distinctive features of `trs` are to give easy, almost geometrical, representations of abstract reduction properties and to present higher-order theorems in a natural way in the higher-order specification language of PVS.

Our intention specifying the `trs` *theory* was not to include exhaustively all well-known results of term rewriting theory, but instead to give the essential mechanisms for expressing and mechanically proving all these results. Adequability of our specification is made evident by presenting elegant formal proofs of well-known properties of ARSs such as Newman's and Yokouchi's Lemmas and of TRSs such as the Knuth-Bendix Critical Pair Theorem.

As future work `trs` should be used to check properties of concrete computational objects which are specified and synthesized by term rewriting systems by methodologies as the ones presented in [4] and [22], respectively. Also, formalizations of termination criteria will be proposed to enlarge the power of the development.

References

[1] Altenkirch, T., *A Formalization of the Strong Normalization Proof for System F in LEGO*, in: M. Bezem and J. F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*, Lecture Notes in Computer Science **664** (1993), pp. 13–28.

[2] Altenkirch, T., *Proving Strong Normalization of CC by Modifying Realizability Semantics*, in: H. P. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, Lecture Notes in Computer Science **806** (1994), pp. 3–18.

[3] Altenkirch, T., P. Dybjer, M. Hofmann and P. Scott, *Normalization by Evaluation for Typed Lambda Calculus with Coproducts*, in: J. Halpern, editor, *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science* (2001), pp. 303–310.

[4] Ayala-Rincón, M., C. H. Llanos, R. P. Jacobi and R. W. Hartenstein, *Prototyping time- and space-efficient computations of algebraic operations over dynamically reconfigurable systems modeled by rewriting-logic*, ACM Transactions on Design Automation of Electronic Systems **11** (2006), pp. 251–281.

[5] Ayala-Rincón, M. and T. M. Sant'Ana, *SAEPTUM: Verification of ELAN Hardware Specifications using the Proof Assistant PVS*, in: *19th Symp. on Integrated Circuits and System Design* (2006), pp. 125–130.

[6] Baader, F. and T. Nipkow, "Term Rewriting and *All That*," Cambridge University Press, 1998.

[7] Bezem, M. and T. Coquand, *Neman's Lemma - a Case Study in proof automation and geometric logic*, Bull. of the European Association for Theoretical Computer Science **79** (2003), pp. 86–100.

[8] Bezem, M., J. W. Klop and R. de Vrijer, editors, "Term Rewriting Systems by TeReSe," Number 55 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2003.

[9] Blanqui, F., S. Coupet-Grimal, W. Delobel, S. Hinderer and A. Koprowski, *CoLoR, a Coq Library on Rewriting and termination*, in: *8th International Workshop on Termination (WST '06)*, 2006.

[10] Boyer, R. S. and J. S. Moore, "A computational logic handbook," Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[11] Bruno, B., "Auto-validation d'un système de preuves avec familles inductives," Thèse de doctorat, Université Paris 7 (1999).

[12] Contejean, E., P. Courtieu, J. Forest, O. Pons and X. Urbain, *Certification of automated termination proofs*, in: B. Konev and F. Wolter, editors, *6th International Symposium on Frontiers of Combining Systems (FroCos 07)*, Lecture Notes in Artificial Intelligence **4720** (2007), pp. 148–162.

[13] Ford, Jonathan M. and Mason, Ian A., *Operational Techniques in PVS – A Preliminary Evaluation.*, in: *Proceedings of the Australasian Theory Symposium, CATS'01*, 2001.

[14] Galdino, A. L. and M. Ayala-Rincón, *A Theory for Abstract Rewriting Systems in PVS*, in: *XXXIII Conferencia Latinoamericana de Informática - CLEI'07*, 2007, p. 12 pages, proceedings in CD. Available `www.mat.unb.br/∼ayala/publications.html`.

[15] Galdino, A. L. and M. Ayala-Rincón, *Verification of Newman's and Yokouchi's Lemmas in PVS*, in: A. Beckmann, C. Dimitracopoulos and B. Löwe, editors, *Local Proceedings of Logic and Theory of Algorithms, Fourth Conference on Computability in Europe - CiE 2008* (2008), pp. 137–146, available: `www.mat.unb.br/∼ayala/publications.html`.

[16] Huet, G., *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems*, Journal of the Association for Computing Machinery **27(4)** (1980), pp. 797–821.

[17] Huet, G., *Residual Theory in λ-calculus: A Formal development*, Jornal of Functional Programming **4(3)** (1994), pp. 371–394.

[18] Kapur, D. and H. Zhang, *An overview of Rewrite Rule Laboratory (RRL)*, in: N. Dershowitz, editor, *Proc. Third Int. Conf. on Rewriting techniques and Applications, Chapel-Hill, NC*, Lecture Notes in Computer Science **355** (1989), pp. 559–563.

[19] Knuth, D. E. and P. B. Bendix, *Simple Word Problems in Universal Algebra*, Computational Problems in Abstract Algebra (1970), pp. 263–297.

[20] Koprowski, A., *A Formalization of the Simply Typed Lambda in Coq* (2006), available: `http://citeseer.ist.psu.edu/742197.html`.

[21] McKinna, J. and R. Pollack, *Some Lambda Calculus and Type Theory Formalized*, Journal of Automated Reasoning **23** (1999), pp. 373–409.

[22] Morra, C., J. Becker, M. Ayala-Rincón and R. W. Hartenstein, *FELIX: Using Rewriting-Logic for Generating Functionally Equivalent Implementations*, in: *15th Int. Conference on Field Programmable Logic and Applications - FPL 2005* (2005), pp. 25–30.

[23] Nipkow, T., *More Church-Rosser Proofs (in Isabelle/HOL)*, in: M. McRobbie and J. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction (CADE-13)*, Lecture Notes in Artificial Intelligence **1104** (1996), pp. 733–747.

[24] Nipkow, T., *More Church-Rosser Proofs*, Journal of Automated Reasoning **26** (2001), pp. 51–66.

[25] Pfenning, F., *A Proof of the Church-Rosser Theorem and its Representation in a Logical Framework* (1992), a preliminary version is available as Carnegie Mellon Technical Report CMU-CS-92-186. available: http://citeseer.ist.psu.edu/pfenning92proof.html.

[26] Rasmussen, O., *The Church-Rosser Theorem in Isabelle: A Proof Porting Experiment*, Technical Report UCAM-CL-TR-364, Computer Laboratory, University of Cambridge (1995).

[27] Ruiz-Reina, J.-L., J.-A. Alonso, M.-J. Hidalgo and F.-J. Martín-Mateos, *Formal Proofs About Rewriting Using ACL2*, Annals of Mathematics and Artificial Intelligence **36** (2002), pp. 239–262.

[28] Saïbi, A., *Formalization of a lamda-Calculus with Explicit Substitutions in Coq*, in: *TYPES'94: Selected papers from the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science **996** (1995), pp. 183–202.

[29] Shankar, N., *A Mechanical Proof of the Church-Rosser theorem*, Journal of the Association for Computing Machinery **35** (1988), pp. 475–522.

[30] van Oostrom, V., *Development Closed Critical Pairs*, in: *Selected Papers from the Second International Workshop on Higher-Order Algebra, Logic, and Term Rewriting - HOA'95*, Lecture Notes in Computer Science **1074** (1996), pp. 185–200.