A Theory for Abstract Reduction Systems in PVS*

André Luiz Galdino†
Departamento de Matemática, Universidade Federal de Goiás
Campus de Catalão, Brazil
galdino@unb.br

and

Mauricio Ayala-Rincón
Instituto de Ciências Exatas, Universidade de Brasília
Brasília D.F., Brazil
ayala@unb.br

Abstract

A *theory* for Abstract Reduction Systems (ARS) in the proof assistant PVS (Prototype Verification System) called `ars` is described. Adequate specifications of basic definitions and notions of the theory of ARSs such as reduction, confluence and normal form are given and well-known results formalized. The formalizations include non trivial results of the theory of ARSs such as the correctness of the principle of Noetherian Induction, Newman's Lemma and its generalizations, and Commutation Lemmas, among others. Although term rewriting proving technologies have been provided in several specification languages and proof assistants, to our knowledge, before the development presented in this paper there was no complete formalization of an abstract reduction *theory* in PVS. This makes relevant the presented `ars` specification as the basis of a PVS *theory* called `trs` for the general treatment of Term Rewriting Systems.

Keywords: Abstract Reduction Systems, Term Rewriting Systems, Automated Theorem Proving, PVS.

1 Introduction

Concepts and properties related with Abstract Reduction Systems (ARS) and Term Rewriting Systems (TRS) have been specified in several proof assistants, e.g., RRL [9], ACL2 [18], Coq [8], Isabelle [14], Boyer-Moore [19], Otter [4] among others. Term rewriting proving technologies have been shown adequate in several mathematics and computer science fields including theorem proving as well as formal specification and design of computational processes and technologies (i.e., standard and non-standard software and hardware). In particular, we have developed a methodology for specifying reconfigurable hardware over FPGAs using the rewriting-logic programing environment ELAN [1]. These rewriting based hardware specifications are synthesized to commercial reconfigurable hardware by applying the system FELIX [10] and their correctness is verified over the proof assistant PVS after translating the rewriting specification to a corresponding logic theory with the system SAEPTUM [2]. The last mentioned step should be improved by making available a full theory of rewriting methods in PVS, that to our knowledge before our full PVS development for TRS reported in [5] was not available in this proof assistant.

With this motivation, this paper introduces a PVS *theory* called `ars` for dealing with properties of ARSs. Basic ARS notions are adequately specified in such a way that non elementary proof techniques such as Noetherian induction are straightforwardly applicable. To illustrate the adequateness of these specification

*This work was supported by the Brazilian Research Council and the District Federal Research Foundation under grants CNPq/DFG 490396/2007-0 and FAP-DF 8-004/2007. Authors are partially supported by the CNPq. The first author presented the work during the CLEI 2007 with financial support from FINATEC.

†Author on leave at the Universidade de Brasília as PhD student.

well-known results that are considered proof benchmarks such as Newman's, Yokouchi's and commutation Lemma are verified [6]. These specifications are built over PVS *theories* for sets and relations. In particular Noetherianity is based on the notion of well-founded relations and because of this, after introducing the notion of noetherian relation the principle of noetherian induction should be verified.

It should be stressed that *abstractness* is one of the distinctive features of `ars`; in fact, based on the PVS *theory* for binary relations, confluence properties of ARSs are formalized in an "almost geometric style" (eg [6]) as it was done in [13] for proof-checking the Church-Rosser theorem of the λ-calculus in Isabelle/HOL.

The introduced PVS *theory* `ars` was conceived as a first step in the development of a full TRS theory in PVS. The files of this theory are available at `www.mat.unb.br/~ayala/TCgroup`.

2 Brief Introduction to PVS

This section briefly describes the PVS prover used to specify the ARS theory. PVS consists of a specification language integrated with support tools and a proof assistant, 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 the tool, refer to the PVS System Guide [17], the PVS Prover Guide [20] and the PVS Language Reference [16] 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 is organized as a collection of *theories*, from which the most relevants are collectively referred as the *prelude* [15]. Each *theory* is composed essentially of *declarations*, which are used to introduce names for types, constants, variables, axioms and formulas, and `IMPORTINGs`, 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` *theory* below:

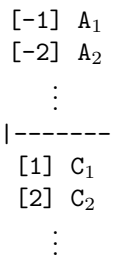
```
ars[T : TYPE] : THEORY
BEGIN

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

`T` is treated as a fixed uninterpreted type. Consequently, when the `ars` *theory* is invoked by another theory, `T` must be instantiated. For example, the *theory* of `ars` of set of `term` is just `ars[term]`. Notice that `ars` imports the *sub-theories* `results_commutation[T]`, `modulo_equivalence[T]`, `results_normal_form[T]` and `newman_yokouchi[T]`.

A important step in PVS specifications is type-checking the *theory*, which checks for semantic errors, such as undeclared names and ambiguous types. Type-checking may build new files or internal structures such as `TCCs` (type-correctness conditions). These `TCCs` represent *proof obligations* that must be discharged before the *theory* can be considered type-checked, and its proofs may be postponed indefinitely. Although, the *theory* is considered *complete* when all `TCCs` and formulas upon which the proof is 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. Each proof goal is a sequent consisting of two sequences of formulas called the antecedents (logically connected by conjunctions and numbered with negative integers) and the consequent (connected by disjunctions and numbered with positive integers) displayed as below:



3 PVS Strategies Used in the Proofs

Below we describe some PVS prover commands that are commonly used in our specification, and we define some strategies that reduce the size of the proofs. Many of these commands take parameters that control its behavior which are not discussed here. For additional details see [20].

1. `skolem:` This command chooses fresh constant names (universally quantified consequent), and proving "without loss of generality", or unconstrained arbitrary constant when one is known to exist (existentially quantified antecedent). In other words, `skolem` gives new constant names, e.g., for `x` it will give `x!1`, `x!2`, ... when applied repeatedly.

2. `skeep:` This command, from the *Field* library, is used to introduce Skolem constants by keeping the original names of the quantified variables. See [11] and [21].

3. `flatten:` This command is used to break an antecedent formula that is a conjunction or a consequent formula that is a disjunction into its components.

4. `assert:` This command is used to simplify the proof goal using decision procedures and rewriting.

5. `inst:` This command is used to instantiate a universally quantified antecedent or an existentially quantified consequent formula.

6. `case:` This command generates two subgoal, one where the given boolean expression is assumed to be `true` and the other where it is assumed to be `false`.

7. `expand:` This command expands and simplifies the definitions of the specified functions/predicates at the occurrences.

8. `lemma:` This command is used to put in a previously proved theorem as antecedent formula into the current proof goal instantiated as specified by the user.

Other useful rules can be found in [20], e.g., `replace`, `prop`, `split` and `decompose-equality`.

4 Specifying ARS in PVS

We briefly present the standard definitions of ARS and some properties [3] and then we present their specification in the `ars` *theory* which is organized in PVS *sub-theories* (see Figure 1).

An *Abstract Reduction System* (ARS) is a pair $(A, \rightarrow)$, where the *reduction* $\rightarrow$ is a binary relation on the set A, i.e., $\rightarrow \subseteq A \times A$. In this paper we consider some arbitrary but fixed ARS $(A, \rightarrow)$. We treated, in PVS, the set A as a fixed uninterpreted type `T`, and the reduction $\rightarrow$ as a binary relation `R` on `T` defined as predicate `PRED: TYPE = [[T,T] -> bool]`. So the relation `R(x,y)` means `x` reduces to `y`, and `y` is called a *reduct* of `x`.

To specify some of the central notions of ARS such as *confluence* and *termination*, first, it is necessary to adequately speficify several closure relations:

Abstract definition	PVS specification
$\rightarrow^0 := \{(x,x) \mid x \in A\}$	identity
$\rightarrow^{i+1} := \rightarrow^i \circ \rightarrow$	$(i+1)$-fold composition, $i \geq 1$
$\rightarrow^= := \rightarrow \cup \rightarrow^0$	reflexive closure (`RC`)
$\rightarrow^+ := \bigcup_{i>0} \rightarrow^i$	transitive closure (`TC`)
$\rightarrow^* := \rightarrow^+ \cup \rightarrow^0$	reflexive transitive closure (`RTC`)
$\leftarrow := \rightarrow^{-1}$	inverse (`converse`)
$\leftrightarrow := \rightarrow \cup \leftarrow$	symmetric closure (`SC`)
$\leftrightarrow^* := (\leftrightarrow)^*$	equivalence closure (`EC`)

`RC`, `TC`, `RTC`, `SC`, and `EC` were defined in the *sub-theory* `relations_closure` in the same way that Alfons Geser does in the PVS *theory* for closure operators (`closure_ops`). We just changed the names of the definitions and we proved some additional properties. For example, `RTC` is defined 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))
```

Then the additional properties are proved:

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

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

4.1 Confluence

For all $x, y, z \in A$ a relation $\rightarrow$ is called

1. *confluent* iff $y \,{}^*\!\leftarrow\, x \rightarrow^* z$ implies that y and z are *joinable*, i.e., iff there is a $r \in A$ such that $y \rightarrow^* r \,{}^*\!\leftarrow z$;
2. *Church-Rosser* iff $x \leftrightarrow^* y$ implies that x and y are joinable;
3. *semi-confluent* iff $y \leftarrow x \rightarrow^* z$ implies that y and z are joinable.

These and other notions such as *local confluent*, *strongly confluent*, *diamond property*, *normal form*, *normalizing* and *commutation* are specified in the *sub-theory* `ars_terminology` as follow:

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

  IMPORTING relations_closure[T]

    R, R1, R2: VAR PRED[[T, T]]
   x, y, z, r: 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)

semi_confluent?(R): bool = FORALL x, y, z: R(x,y) & RTC(R)(x,z) => joinable?(R)(y,z)

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

commute?(R1,R2): bool = FORALL x, y, z: RTC(R1)(x,y) & RTC(R2)(x,z) =>
                                              EXISTS r: RTC(R2)(y,r) & RTC(R1)(z,r)

strong_commute?(R1,R2): bool = FORALL x, y, z: R1(x,y) & R2(x,z) =>
                                              EXISTS r: RC(R2)(y,r) & RTC(R1)(z,r)
...
END ars_terminology
```

Some basic results involving confluence are specified and proved in the `results_confluence` *sub-theory*. For example, the equivalence between Church-Rosser and confluence, and the *commutative union lemma* which tells us that for commutative relations union preserves confluence are specified as:

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

Commutative_Union_Lemma: LEMMA confluent?(R1) & confluent?(R2) &
                            commute?(R1,R2) => confluent?(union(R1, R2))
```

4.2 Termination

A relation $\rightarrow$ is called *terminating* or *noetherian* iff there is no infinite descending chain $a_0 \rightarrow a_1 \rightarrow \cdots$. In other words, $\rightarrow$ is *noetherian* iff $\leftarrow$ is well-founded.

As it is well-known many results involving termination are proved by *Noetherian induction*, that is: let P be some property of elements of A. Then to prove $P(x)$ for all $x \in A$, it suffices to prove $P(x)$ under the assumption that $P(y)$ holds for all successors $y \in A$ of x.

In the `noetherian` *sub-theory* below, we defined noetherian relation based on the notion of well-founded relation (it simplify proofs) and we proved the principle of Noetherian induction. To prove this principle we used the lemma `wf_induction`, with suitable substitutions, which expresses the principle of *well-founded induction* and can be found in the PVS prelude *theory* [15] as well as the notions of *well-founded* relations.

```
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
```

4.3 Modulo Equivalence

Considering a reduction relation `R`, together with an equivalence relation `Eq` we defined the notions of *reduction modulo equivalence* [7] and we proved the generalization of Newman's Lemma:

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

  IMPORTING noetherian[T]

  R, S: VAR  PRED[[T, T]]
    Eq: VAR equivalence
  x, y,
  z, w,
```

```
  u, v: VAR T
...
  joinable_m?(R, Eq)(x,y): bool = EXISTS u, v: RTC(R)(x,u) & Eq(u,v) & RTC(R)(y,v)

  local_confluent_m?(R, Eq): bool = FORALL x, y, z: R(x,y) & R(x,z) =>
                                                      joinable_m?(R, Eq)(y,z)

  confluent_m?(R, Eq): bool = FORALL x, y, z, w: RTC(R)(x,z)  &
                                                 Eq(x,y) &
                                                 RTC(R)(y,w) =>
                                                 joinable_m?(R, Eq)(z,w)

  locally_coherent?(R, Eq, S): bool = symmetric?(S) &
                                      FORALL  x, y, z: R(x,y) &
                                      S(x,z) => joinable_m?(R, Eq)(y,z)
...

van_oostrom94: LEMMA diamond_property?(RTC(R) o Eq) => confluent_m?(R, Eq)

newman_lemma_general: THEOREM noetherian?(R) =>
                              (local_confluent_m?(R, Eq) &
                               locally_coherent?(R, Eq, Eq) <=>
                                                        confluent_m?(R, Eq))
END modulo_equivalence
```

5 Organization of the ars PVS Theory, Proof Examples and Proof Summary

5.1 Organization of the theory ars

Below we show the organization of the PVS *sub-theories* which compound the `ars` *theory* and we give a brief description of each one (see Figure 1).

1. `relations_closure:` This *sub-theory* contains the definitions of closure of a relation and some properties.

2. `ars_terminology:` This *sub-theory* contains some terminology of ARS such as unique normal form, reducible and sucessor, and notions of confluence and commutation.

3. `results_confluence:` This *sub-theory* contains some results about confluence such as *strong confluent implies semi-confluent.*

4. `results_commutation:` This *sub-theory* contains some results about commutation such as Commutation Lemma.

5. `results_normal_form:` This *sub-theory* contains some results involving normal form such as *a relation is normalizing and confluent iff every element has a unique normal form.*

6. `noetherian:` This *sub-theory* contains the definition of *convergent reduction* and noetherian relation and the Noetherian induction lemma.

7. `newman_yokouchi:` This *sub-theory* contains the specification of Newman's Lemma and Yokouchi's Lemma.

8. `modulo_equivalence:` This *sub-theory* contains the notions of reduction modulo equivalence and, for example, the proof of the generalization of Newman's Lemma.

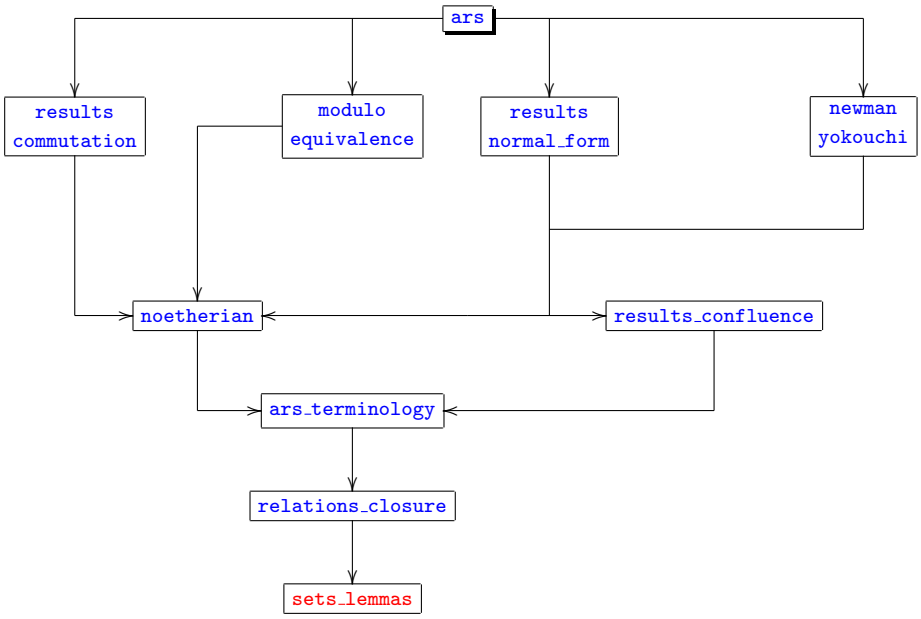


Figure 1: Hierarchy of the `ars` *theory*

5.2 Proof Examples

The PVS proofs are available as part of the `ars` *theory* at `www.mat.unb.br/~ayala/TCgroup` and detailed explanations of the PVS proofs of Newmann's and Yokouchi's lemmas are available in [6].

To prove the commutation lemma:

```
Commutation_Lemma: THEOREM strong_commute?(R1,R2) => commute?(R1,R2)
```

we use the sequence of commands `skeep`, `expand`, `skolem`, `lemma`, `inst`, and `assert`. The command `lemma` is used to invoke the following lemma:

```
commute_and_iterate_two: LEMMA FORALL (n,m: nat): strong_commute?(R1,R2)
                         & iterate(R1, n)(x,y) & iterate(R2, m)(x,z) =>
                                   EXISTS r: RTC(R2)(y,r) & RTC(R1)(z,r)
```

This lemma is proved by induction on `m` by applying the command `(induct "m")`, and by invoking the lemma `commute_and_iterate_one`, presented below, which is proved by induction too.

```
commute_and_iterate_one: LEMMA FORALL (n: nat): strong_commute?(R1,R2)
                         & iterate(R1, n)(x,y) & R2(x,z) =>
                                      EXISTS r: RTC(R1)(z,r) & RC(R2)(y,r)
```

Now, we present details of the proof of Newman's lemma that since the inductive proof given by Huet in [7] is considered a classical benchmark for proof in higher-order logic as discussed for instance in [4].

Lemma 1 (Newman's Lemma [12]) *Let R be a noetherian relation defined on the set T. Then R is confluent if, and only if it is locally confluent.*

proof (Sketch). The $\Rightarrow$-direction follows immediately by definition. $\Leftarrow$-direction is proved by noetherian induction using the predicate

$$P(x) = \forall y, z.\ y\ {}^*\!\leftarrow x \rightarrow^* z \implies y \text{ and } z \text{ joinable}$$

Obviously R is confluent if $P(x)$ holds for all x. Noetherian induction require us to show $P(x)$ under the assumption $P(t)$ for all t such that $x \rightarrow^+ t$. To prove $P(x)$, we analyze the divergence $y\ {}^*\!\leftarrow x \rightarrow^* z$. If

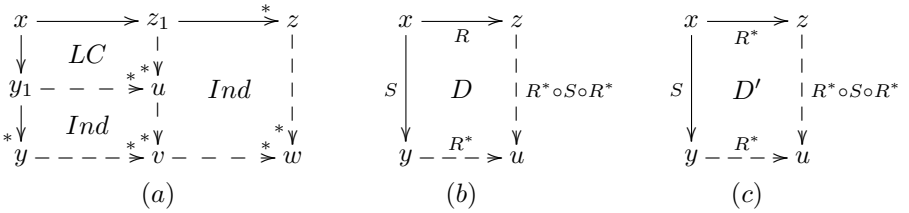


Figure 2: Proof of Newman's Lemma, Diagram D and Generalization of D as D'

$x = y$ or $x = z$, y and z are joinable immediately. Otherwise we have $x \to y_1 \to^* y$ and $x \to z_1 \to^* z$ as shown in the Figure 2(a), where as usual dashed arrows stand for *existence*. The existence of u follows by local confluence (LC) of R, the existence of v and w follows by induction hypothesis (Ind). □

The formalizations of Newman's Lemma use 114 proof steps and here only the relevant fragment of the proof trees, focusing on the application of noetherian induction, are presented. The Newman's Lemma is specified in the `newman_yokouchi` *sub-theory* as:

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

When the PVS prover is invoked the proof tree starts off with a root node (sequent) having no antecedent and as succedent the theorem to be proved.

```
Newman_lemma :
  |-------
{1}   FORALL (R: PRED[[T,T]]):
        noetherian?(R) => (confluent?(R) <=> local_confluent?(R))
```

The reduction relation `R` is correctly universally quantified, since it was declared as a variable in the *theory* `newman_yokouchi` (see Figure 1). After skolemization by applying the proof command `(skeep)`, the conjunctive splitting command `(split)` is applied to the goal obtaining two subgoals. The first subgoal, `Newman_lemma.1`, is to demonstrate that confluence implies local confluence, which is easily formalized. The second subgoal, `Newman_lemma.2`, that is to demonstrate that local confluence implies confluence (under noetherianity hypothesis), is the truly interesting one.

For proving this subgoal, after disjuntive simplification with `(flatten)`, one introduces the noetherian induction scheme `noetherian_induction` and instantiates its predicate `P` as:

```
(LAMBDA (a:T): (FORALL (b,c:T): RTC(R)(a,b) & RTC(R)(a,c) => joinable?(R)(b,c)))
```

Then, the subgoals `Newman_lemma.2.1` and `2.2` presented below are obtained by applying the command `(split)`, that splits the implication of the instantiated noetherian induction scheme. The first subgoal is easily verified by expanding the definition of the predicate `confluent?`, skolemization and adequate instantiation of the variables of the antecedent {`-1`}.

```
Newman_lemma.2.1 :
{-1}  FORALL (x:T):  FORALL (b,c:T):
         RTC(R)(x,b) & RTC(R)(x,c) => joinable?(R)(b,c)
[-2]  local_confluent?(R)   [-3]  noetherian?(R)
  |-------
[1]   confluent?(R)

Newman_lemma.2.2 :
[-1]  local_confluent?(R)   [-2]  noetherian?(R)
  |-------
{1}   FORALL (x:T): (FORALL (y:T): TC(R)(x,y) => (FORALL (b,c: T):
              RTC(R)(y,b) & RTC(R)(y,c) => joinable?(R)(b,c)))
      => (FORALL (b,c:T): RTC(R)(x,b) & RTC(R)(x,c)=> joinable?(R)(b,c))
```

To prove the latter subgoal, one needs to show `P(x)` under the assumption `P(y)` for all `y` such that `x` $\rightarrow^+$ `y`. After skolemization, expansion of the definition of `RTC` and hidding unnecessary formulas one obtains the following sequent.

```
Newman_lemma.2.2 :
[-1]  FORALL (y:T): TC(R)(x,y) => (FORALL (b,c:T):
            RTC(R)(y,b) & RTC(R)(y,c) => joinable?(R)(b,c))
{-2}  iterate(R,i)(x,b)  {-3}  iterate(R,j)(x,c)
[-4]  local_confluent?(R)  [-5]  noetherian?(R)
  |-------
[1]   joinable?(R)(b, c)
```

To prove this goal, one analyzes the cases `x = b` or `x = c` or `b` $\neq$ `x` $\neq$ `c`. To contemplate these cases one uses the command `(case-replace "i = 0")` which replaces `i` by `0` in the current subgoal and generates a second subgoal for the case `x = b`. Similarly, the case `x = c` is proved. The case `x` $\neq$ `b` and `x` $\neq$ `c`, i.e., $x \rightarrow x1 \rightarrow^* b$ and $x \rightarrow x2 \rightarrow^* c$ corresponds to the following sequent obtained after some simplifications. Compare with the diagram of Figure 2(a) (replacing some variable symbols: `u, v` and `w`).

```
Newman_lemma.2.2.2.2.1.1 :

[-1]  RTC(R)(x1,b)  [-2]  RTC(R)(x2,c)
[-3]  FORALL (y:T): TC(R)(x,y) => (FORALL (b1,c1:T):
            RTC(R)(y,b1) & RTC(R)(y,c1) => joinable?(R)(b1,c1))
[-4]  R(x,x1)  [-5]  R(x,x2)  [-6]  RTC(R)(x1,u)  [-7]  RTC(R)(x2,u)
[-8]  noetherian?(R)
  |-------
[1]   j = 0   [2]   i = 0   [3]   joinable?(R)(b,c)
```

Firstly, make a copy of the formula `-3` by using `(copy -3)`.

The existence of `u` follows by expanding `local_confluent?` (instantiated with variables `x`, `x1` and `x2`), `joinable?`, by introducing `skolem` constants (`u`) and by applying disjunctive simplification `flatten`. Then one applies the lemma `R_subset_TC`, which states that a relation is contained in its transitive closure and one proves that `x` $\rightarrow^+$ `x1` and `x` $\rightarrow^+$ `x2`. Thus, the existence of `v` and `w` follows by induction hypothesis, that is by instantiating `[-3]` conveniently, and the lemma follows.

Another interesting example of results available in the `newman_yokouchi` *sub-theory* and obtained by Noetherian induction is the Yokouchi lemma which is specified as:

```
Yokouchi_lemma: THEOREM ( noetherian?(R) & confluent?(R) &
         diamond_property?(S) & (FORALL x, y, z: (S(x,y) & R(x,z)) =>
           (EXISTS (u: T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))) )
                              => diamond_property?(RTC(R) o S o RTC(R))
```

5.3 Proof Summary

We present below a proof summary for all *sub-theories* involved with `ars` *theory* that is obtained by using the PVS prover invocation command `ALT-x pri`. The complete `ars` development runs in PVS 4.2 and consists of 65 lemmas, from which 5 are TCCs only, specified in 791 lines (48K) and 8384 lines (640K) of proofs.

```
Proof summary for theory results_commutation
   Local_Comu_and_Noeth.................proved - complete   [shostak](0.53 s)
   commute_and_iterate_one..............proved - complete   [shostak](0.19 s)
   commute_and_iterate_two..............proved - complete   [shostak](0.17 s)
   Comutation_Lemma.....................proved - complete   [shostak](0.07 s)
   Theory totals: 4 formulas, 4 attempted, 4 succeeded (0.96 s)

Proof summary for theory modulo_equivalence
   van_oostrom94........................proved - complete   [shostak](0.15 s)
   newman_lemma_general.................proved - complete   [shostak](0.81 s)
   Theory totals: 2 formulas, 2 attempted, 2 succeeded (0.96 s)
```

```
Proof summary for theory results_normal_form
   NF_doesnot_rewrite.....................proved - complete   [shostak](0.08 s)
   NF_implies_RTC.........................proved - complete   [shostak](0.04 s)
   NFs_implies_Equal......................proved - complete   [shostak](0.04 s)
   Norm_and_Confl_implies_UNF.............proved - complete   [shostak](0.06 s)
   Normalizing_and_Confl..................proved - complete   [shostak](0.13 s)
   Normal_Confl_iff_UNF...................proved - complete   [shostak](0.06 s)
   Noetherian_implies_normalizing........proved - complete   [shostak](0.07 s)
   Convergent_UNF.........................proved - complete   [shostak](0.02 s)
   Noet_and_Confl_iff_UNF.................proved - complete   [shostak](0.03 s)
   Convergent_iff_eqNF....................proved - complete   [shostak](0.10 s)
   Theory totals: 10 formulas, 10 attempted, 10 succeeded (0.63 s)

Proof summary for theory newman_yokouchi
   Newman_lemma...........................proved - complete   [shostak](0.37 s)
   Yokouchi_lemma_ax1.....................proved - complete   [shostak](0.51 s)
   Yokouchi_lemma.........................proved - complete   [shostak](0.62 s)
   Theory totals: 3 formulas, 3 attempted, 3 succeeded (1.50 s)

Proof summary for theory results_confluence
   Joinable_implies_Equiv.................proved - complete   [shostak](0.08 s)
   reduct_transitive......................proved - complete   [shostak](0.08 s)
   semi_and_iterate.......................proved - complete   [shostak](0.20 s)
   Confl_implies_Semi.....................proved - complete   [shostak](0.07 s)
   Semi_implies_CR........................proved - complete   [shostak](0.07 s)
   CR_iff_Confluent.......................proved - complete   [shostak](0.10 s)
   strong_and_iterate.....................proved - complete   [shostak](0.14 s)
   Str_Confl_implies_Semi_Confl..........proved - complete   [shostak](0.09 s)
   Strong_Confl_implies_Confl............proved - complete   [shostak](0.06 s)
   DP_implies_StC.........................proved - complete   [shostak](0.07 s)
   R1_Confl_iff_R2_Confl..................proved - complete   [shostak](0.13 s)
   R1_equal_R2............................proved - complete   [shostak](0.08 s)
   R2_Str_Confl_implies_R1_Confl.........proved - complete   [shostak](0.07 s)
   Confluence_Commute.....................proved - complete   [shostak](0.12 s)
   R1_R2_RTC_R1_R2........................proved - complete   [shostak](0.19 s)
   Commutative_Union_Lemma................proved - complete   [shostak](0.07 s)
   Theory totals: 16 formulas, 16 attempted, 16 succeeded (1.62 s)

Proof summary for theory noetherian
   R_is_Noet_iff_TC_is....................proved - complete   [shostak](0.08 s)
   noetherian_induction...................proved - complete   [shostak](0.05 s)
   Theory totals: 2 formulas, 2 attempted, 2 succeeded (0.13 s)

Proof summary for theory ars_terminology
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory relations_closure
   RC_TCC1................................proved - complete   [shostak](0.03 s)
   change_to_RC...........................proved - complete   [shostak](0.02 s)
   R_subset_RC............................proved - complete   [shostak](0.03 s)
   RC_idempotent..........................proved - complete   [shostak](0.15 s)
   RC_characterization....................proved - complete   [shostak](0.07 s)
   SC_TCC1................................proved - complete   [shostak](0.05 s)
   change_to_SC...........................proved - complete   [shostak](0.01 s)
   R_subset_SC............................proved - complete   [shostak](0.02 s)
   SC_idempotent..........................proved - complete   [shostak](0.15 s)
   SC_characterization....................proved - complete   [shostak](0.08 s)
   TC_TCC1................................proved - complete   [shostak](0.07 s)
   change_to_TC...........................proved - complete   [shostak](0.02 s)
```

```
    R_subset_TC.........................proved - complete   [shostak](0.03 s)
    TC_converse.........................proved - complete   [shostak](0.12 s)
    TC_idempotent.......................proved - complete   [shostak](0.14 s)
    TC_characterization.................proved - complete   [shostak](0.08 s)
    RTC_TCC1............................proved - complete   [shostak](0.06 s)
    change_to_RTC.......................proved - complete   [shostak](0.02 s)
    R_subset_RTC........................proved - complete   [shostak](0.03 s)
    iterate_RTC.........................proved - complete   [shostak](0.03 s)
    RTC_idempotent......................proved - complete   [shostak](0.15 s)
    RTC_characterization................proved - complete   [shostak](0.08 s)
    EC_TCC1.............................proved - complete   [shostak](0.15 s)
    change_to_EC........................proved - complete   [shostak](0.03 s)
    R_subset_EC.........................proved - complete   [shostak](0.08 s)
    RTC_subset_EC.......................proved - complete   [shostak](0.15 s)
    EC_idempotent.......................proved - complete   [shostak](0.15 s)
    EC_characterization.................proved - complete   [shostak](0.07 s)
    Theory totals: 28 formulas, 28 attempted, 28 succeeded (2.07 s)

 Proof summary for theory ars
    Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Grand Totals: 65 proofs, 65 attempted, 65 succeeded (7.87 s)
```

6 Conclusions and Future Work

The PVS *theory* `ars` specifies adequately basic notions of the theory of Abstract Reduction Systems. On the one hand `ars` is built over the PVS *theory* for binary relations being the closures specified in terms of "iteration" of the binary relations. In this way inductive proofs on the length of derivations are possible. On the other hand, the notion of noetherianity is specified in terms of the notion of well-founded relations which allows us to adequately formulate and verify the principle of noetherian induction necessary for proving several properties of ARSs.

Our intention specifying the `ars` *theory* was not to exhaustively include proofs of all well-known results of the theory of ARSs, but instead to give the essential mechanisms for expressing and mechanically proving all these results. Adequability of our specification is made evident by the presentation of elegant proofs of well-known results over ARSs such as the Newman's and Yokouchi's lemmas [6]. Also it should be stressed here that although `ars` does not advance the state of the art in the formalization of mathematics since specifications of Abstract Reductions Systems and even of Term Rewriting Systems are available since the development of the Rewriting Rule Laboratory (RRL) in the 1980s [9], it is of practical interest since the availability of rewriting proving technologies are essential in a modern proof assistants as PVS.

`ars` was extended to a more elaborated PVS *theory* for full Term Rewriting Systems, called `trs` [5], that is of interest to verify the correction of concrete rewriting based specifications of computational objects as mentioned in the introduction. The PVS *theory* `trs` consists of 166 lemmas specified in 71980 lines (108K) and 42105 lines (2.8M) of proofs. Among the results included in `trs` we mention the first higher-order complete formalization of the Knuth-Bendix Critical Pair Theorem.

References

[1] Ayala-Rincón, M., Llanos, C. H., Jacobi, R. P., and Hartenstein, R. W. Prototyping time- and space-efficient computations of algebraic operations over dynamically reconfigurable systems modeled by rewriting-logic. *ACM Trans. Design Autom. Electr. Syst. 11*, 2 (2006), 251–281.

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

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

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

[5] Galdino, A. L., and Ayala-Rincón, M. A PVS *Theory* for Term Rewriting Systems. In *Proceedings of the Third Workshop on Logical and Semantic Frameworks, with Applications - LSFA 2008* (2008), E. Pimentel and M. Benevides, Eds., Electronic Notes in Theoretical Computer Science, Elsevier. Accepted. Available: `www.mat.unb.br/~ayala/publications.html`.

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

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

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

[9] Kapur, D., and Zhang, H. An overview of Rewrite Rule Laboratory (RRL). In *Proc. Third Int. Conf. on Rewriting techniques and Applications, Chapel-Hill, NC* (April 1989), N. Dershowitz, Ed., vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag.

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

[11] Muñoz, C., and Mayero, M. Real Automation in the Field. ICASE Interim Report 39 NASA/CR-2001-211271, NASA Langley Research Center, NASA Langley Research Center, December 2001.

[12] Newman, M. H. A. On theories with a combinatorial definition of equivalence. *Annals of Mathematics 43(2)* (1942), 223–243.

[13] Nipkow, T. More Church-Rosser Proofs. *Journal of Automated Reasoning 26*, 1 (2001), 51–66.

[14] Nipkow, T., Paulson, L. C., and Wenzel, M. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, vol. 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[15] Owre, S., and Shankar, N. The PVS Prelude Library. Tech. rep., SRI-CSL-03-01, Computer Science Laboratory, SRI International, Menlo Park, CA, March 2003. Available: `http://pvs.csl.sri.com/`.

[16] Owre, S., Shankar, N., Rushby, J. M., and Stringer-Calvert, D. W. J. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999. Available: `http://pvs.csl.sri.com/`.

[17] Owre, S., Shankar, N., Rushby, J. M., and Stringer-Calvert, D. W. J. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999. Available: `http://pvs.csl.sri.com/`.

[18] Ruiz-Reina, J. L., Alonso, J.-A., Hidalgo, M.-J., and Martín-Mateos, F.-J. Formalizing Rewriting in the ACL2 Theorem Prover. In *AISC'00: Revised Papers from the International Conference on Artificial Intelligence and Symbolic Computation* (London, UK, 2001), vol. 1930 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 92–106.

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

[20] Shankar, N., Owre, S., Rushby, J. M., and Stringer-Calvert, D. W. J. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999. Available: `http://pvs.csl.sri.com/`.

[21] Vito, B. D. *Manip User's Guide, Version 1.1*. NASA Langley Research Center, Hampton, Virginia, February, 18 2003.