Verification of Newman's and Yokouchi's Lemmas in PVS

André Luiz Galdino[1,2], **Mauricio Ayala-Rincón**[1]

[1]Grupo de Teoria da Computação
Instituto de Ciências Exatas – Universidade de Brasília
Brasília, Brazil

[2]Departamento de Matemática – Campus de Catalão – Universidade Federal de Goiás
Catalão, Brazil

{galdino,ayala}@unb.br

Abstract. *This paper shows how a previously specified theory for Abstract Reduction Systems (ARSs) in which noetherianity was defined by the notion of well-foundness over binary relations is used in order to prove results such as the well-known Newman's Lemma and the Yokouchi's Lemma. The former one known as the diamond lemma and the later which states a property of commutation between ARSs. The* `ars` *theory was specified in the Prototype Verification System (PVS) for which to the best of our knowledge there are no available theory for dealing with rewriting techniques in general. In addition to proof techniques available in PVS the verification of these lemmas implies an elaborated use of natural as well as noetherian induction.*

1. Introduction

A PVS theory built over 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 [6]. In that theory basic ARS notions such as reduction, derivation, normal form, confluence, local confluence, joinability, noetherianity, etc., were adequately specified in such a way that non elementary proof techniques such as noetherian induction are possible. In this paper we describe the use of this PVS theory for proving Newman's and Yokouchi's lemmas the later which is of interest because its proof is done by several applications of natural as well as noetherian induction.

The inductive proof of the Newman's Lemma given by Huet in [7] is a classical example of proofs in higher-order logic. Formal proofs of Newman's Lemma have been specified in several proof assistants, e.g., ACL2 [17], Coq [8], Isabelle [16], Boyer-Moore [18], Otter [5] among others.

The novelty of this work in not to present mechanical proofs of ARS theorems in PVS that were done previously in other proof assistants. Our intention is to give a first step in the direction of formalizing an elaborated and robust PVS theory for full Term Rewriting Systems (TRSs) which currently is not available in this proof assistant. The motivation for doing this work is the fact that rewriting has been applied to the specification and synthesis of reconfigurable hardware [2, 10] and that the correction of these specifications can be carried out by translating these rewriting specifications into the language of the PVS proof assistant as logic theories [3]. But robust proof rewriting based methods are necessary in order to deal efficiently with the correctness of these theories that come from rewriting based specifications.

Section 2 presents the necessary background on rewriting theory and analitical proofs of both lemmas. Sections 3 and 4 describe respectively the specification and verification of both lemmas in PVS, and, before concluding, Section 5 presents some new PVS strategies created to simplify the proofs. The PVS files of the `ars` theory as well as the proofs of Newman's, Yokouchi's and other lemmas are available at `www.mat.unb.br/∼ayala/publications.html`.

2. Background

We suppose the reader is familiar with rewriting concepts and standard notations such as presented either in [4] or in [19]. After briefly reviewing notations we start presenting Newman's and Yokouchi's lemmas an sketches of their analytical proofs.

We consider an abstract reduction relation as a binary relation R over a set T, denoted also as $\langle R, T\rangle$. The relation is identified as R, $\rightarrow\ R$ or simply $\rightarrow$. R^+ and R^* respectively denote the transitive and the reflexive transitive closure of R, denoted in arrow notation as $\rightarrow^+$ and $\rightarrow^*$, respectively. In the elegant arrow notation, the inverse relation R^{-1}, its transitive and its reflexive transitive closures are respectively denoted as $\leftarrow$, ${}^+\leftarrow$ and ${}^*\leftarrow$. The operator of composition is denoted as usual as $\circ$. An abstract reduction relation $\rightarrow$ over T is said to be: *confluent* whenever $({}^*\leftarrow \circ \rightarrow^*) \subseteq (\rightarrow^* \circ^* \leftarrow)$ and *locally confluent* whenever $(\leftarrow \circ \rightarrow) \subseteq (\rightarrow^* \circ^* \leftarrow)$. $\rightarrow$ satisfies the *diamond property* whenever $(\leftarrow \circ \rightarrow) \subseteq (\rightarrow \circ \leftarrow)$. Two elements of T, say x, y, are said to be *joinable* whenever $\exists u.x \rightarrow^* u\ {}^*\leftarrow y$. $\rightarrow$ is said to be *noetherian* whenever there is no infinite sequence of the form $x_1 \rightarrow x_2 \rightarrow \cdots$.

Lemma 1 (Newman's Lemma [13]) *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). On the one hand, that confluence implies local confluence follows by their definitions. On the other hand, suppose that R is noetherian and locally confluent. Confluence of R is proved by noetherian induction using the predicate

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

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 $x = y$ or $x = z$, y and z are joinable immediately. Otherwise we have $x \rightarrow y_1 \rightarrow^* y$ and $x \rightarrow z_1 \rightarrow^* z$ as shown in the Figure 1, where LC abbreviates the application of local confluence hypothesis, Ind of induction, and as usual dashed arrows stand for *existence*.

$$\begin{tikzcd}
x \arrow[r] \arrow[d] \arrow[dr, phantom, "LC"] & x_1 \arrow[r, "*"] \arrow[d, dashed, "*"] & z \arrow[dd, dashed, "*"] \\
y_1 \arrow[r, dashed, "*"] \arrow[d, "*"'] \arrow[dr, phantom, "Ind"] & u \arrow[d, dashed, "*"] \arrow[r, phantom, "Ind"] & {} \\
y \arrow[r, dashed, "*"] & v \arrow[r, dashed, "*"] & w
\end{tikzcd}$$

Figure 1. Proof of Newman's Lemma.

The existence of u follows by local confluence of R, the existence of v and w follows by induction hypothesis because $x \rightarrow^{+} y_1$ and $x \rightarrow^{+} z_1$, respectively. □

Lemma 2 (Yokouchi's Lemma [21]) *let R and S be two relations defined on the same set T, R being confluent and noetherian, and S having the diamond property. Suppose moreover that the following diagram holds:*

$$\begin{tikzcd}
x \arrow[r, "R"] \arrow[d, "S"'] \arrow[dr, phantom, "D"] & z \arrow[d, dashed, "R^*\circ S\circ R^*"] \\
y \arrow[r, dashed, "R^*"'] & u
\end{tikzcd}$$

Then the relation $R^* \circ S \circ R^*$ has the diamond property.

proof (Sketch). The proof starts by generalizing the diagram D of the lemma as the diagram D' in the Figure 2. This generalization is proved by noetherian induction using the predicate

$$P(x) := \forall y, z.\ xR^*z\ \wedge\ xSy\ \Rightarrow\ \exists u.(yR^*u\ \wedge\ zR^* \circ S \circ R^*u)$$

$$\begin{tikzcd}
x \arrow[r, "R^*"] \arrow[d, "S"'] \arrow[dr, phantom, "D'"] & z \arrow[d, dashed, "R^*\circ S\circ R^*"] \\
y \arrow[r, dashed, "R^*"'] & u
\end{tikzcd}$$

Figure 2. Generalization of Diagram D as D'

Then, to prove that $R^* \circ S \circ R^*$ has the diamond property, one also proceeds by noetherian induction but this time using the predicate

$$P'(x) := \forall y, z.\ xR^* \circ S \circ R^*y \wedge xR^* \circ S \circ R^*z\ \Rightarrow\ \exists u.(yR^* \circ S \circ R^*u \wedge zR^* \circ S \circ R^*u)$$

We conclude, by induction in the length of the derivation of the first R^* in $xR^* \circ S \circ R^*y$. In other words, we distinguish between the cases $xR \circ R^* \circ S \circ R^*y$ and $xS \circ R^*y$ as is shown in the Figure 3, where C and DP stand for use of confluence of R and diamond property of S hypotheses. □

3. Specification

The PVS theory `newman_yokouchi` presented in Table 1 specifies Newman's and Yokouchi's lemmas. This theory is parameterized as `newman_yokouchi[T]`, where (within the `newman_yokouchi` theory) `T` is treated as a fixed uninterpreted type. `R` and `S` denote rewriting relations over `T` and `x, y, z, w` and `u` elements of `T`. `=>`, `<=>` and `&` are abbreviations for `IMPLIES`, `IFF` and `AND`, respectively.

Newman's lemma specification is straightforward and based on predicates over rewriting relations. In the specification of the Yokouchi's lemma `RTC(R)` denotes the reflexive transitive closure of the relation `R`, i.e. `R`*. The composition of relations is

$$\begin{tikzcd}
x \arrow[r, "R^*"] \arrow[d, "R"'] \arrow[dr, phantom, "C"] & z_1 \arrow[r, "S"] \arrow[d, dashed, "R^*"] \arrow[dr, phantom, "D'"] & z_2 \arrow[r, "R^*"] \arrow[d, dashed, "R^*"] \arrow[dr, phantom, "C"] & z \arrow[d, dashed, "R^*"] \\
y_1 \arrow[r, dashed, "R^*"'] \arrow[d, "R^*\circ S\circ R^*"'] \arrow[drrr, phantom, "Ind"] & u_1 \arrow[r, dashed, "R^*\circ S\circ R^*"'] & u_2 \arrow[r, dashed, "R^*"'] & u_3 \arrow[d, dashed, "R^*\circ S\circ R^*"'] \\
y \arrow[rrr, dashed, "R^*\circ S\circ R^*"'] & & & u
\end{tikzcd}
\qquad
\begin{tikzcd}
x \arrow[r, "S"] \arrow[d, "S"'] \arrow[dr, phantom, "DP"] & z_1 \arrow[r, "R^*"] \arrow[d, dashed, "S"] \arrow[dr, phantom, "D'"] & z \arrow[d, dashed, "R^*\circ S\circ R^*"] \\
y_1 \arrow[r, dashed, "S"'] \arrow[d, "R^*"'] \arrow[dr, phantom, "D'"] & u_1 \arrow[r, dashed, "R^*"'] \arrow[d, dashed, "R^*"] \arrow[dr, phantom, "C"] & u_2 \arrow[d, dashed, "R^*"] \\
y \arrow[r, dashed, "R^*\circ S\circ R^*"'] & u_3 \arrow[r, dashed, "R^*"'] & u
\end{tikzcd}$$

Figure 3. Cases $xR \circ R^* \circ S \circ R^*y$ and $xS \circ R^*y$

denoted as `o`. The first and third lemmas of this theory correspond to the lemmas 1 and 2 respectively. The second lemma, `Yokouchi_lemma_ax1`, corresponds to the generalization D' of the diagram D presented in Figure 2.

The theories `results_confluence[T]` and `noetherian[T]`, also components of the whole `ars` theory, are imported by the `newman_yokouchi` theory. The former containing results about confluence and the later the definition of noetherianity formulated in terms of the notion of well-founded relation as it will be explained.

Specifications of rewriting properties are exemplified by the following ones.

```
joinable?(R)(x,y): bool = EXISTS z: RTC(R)(x,z) & RTC(R)(y, z)

local_confluent?(R): bool =
 FORALL x, y, z: R(x,y) & 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)

diamond_property?(R): bool =
  FORALL x, y, z: R(x,y) & R(x,z) => EXISTS r: R(y,r) & R(z,r)
```

In order to make easy the use of natural induction, closures of relations are built as unions of iterations of their compositions. For instance, the reflexive transitive closure operator `RTC` of a relation `R` is specified as the union of the iterations `iterate(R, i)`, for all `i` ≥ 0, where `iterate(R,i)` specifies the relation $\underbrace{\mathtt{R} \circ \mathtt{R} \cdots \circ \mathtt{R}}_{\mathtt{i}\text{ times}}$ available in the PVS prelude libraries. The PVS prelude operators `iterate` and `IUnion` are specified as

```
  iterate(p, i)(x): RECURSIVE T =
    IF i = 0 THEN x ELSE p(iterate(p, i-1)(x)) ENDIF
    MEASURE i

  IUnion(A): set[T] = {x | EXISTS i: A(i)(x)}
```

and using them `RTC` can be specified as

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

Notice that the type of `RTC` is `reflexive_transitive`. This places in the basis of the system of types the intrinsic characteristics of the `RTC` construction creating

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

  IMPORTING results_confluence[T], noetherian[T]

     R, S : VAR PRED[[T, T]]

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

 Yokouchi_lemma_ax1: LEMMA
  (noetherian?(R) & confluent?(R) &
   (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))))
    =>
     (FORALL x,y,z: (S(x,y) & RTC(R)(x,z)) =>
      (EXISTS (w:T): RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w)))

 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))

END newman_yokouchi
```

Table 1. newman_yokouchi **PVS theory**

the necessary proofs obligations to be proved (PVS Type Correctness Conditions - TCCs). This type is specified as follows.

```
  reflexive_transitive?(R): bool = reflexive?(R) & transitive?(R)

  reflexive_transitive: TYPE = (reflexive_transitive?)
```

Noetherianity is defined in terms of well foundness.

```
 noetherian?(R): bool = well_founded?(converse(R))

 noetherian: TYPE = (noetherian?)
```

Noetherian induction is then verified using the lemma `wf_induction`, which expresses the principle of *well-founded induction* and can be found in the PVS prelude theory [14] as well as the notion of well-founded relation.

```
  wf_induction: LEMMA
    (FORALL (p: pred[T]):
      (FORALL (x: T):
        (FORALL (y: T): y<x => p(y))
           => p(x))
    =>
      (FORALL (x:T): p(x)))
```

The lemma `noetherian_induction` presented below corresponds to the principle of noetherian induction.

```
 noetherian_induction: LEMMA
             (FORALL (R: noetherian, P: PRED[T]):
               (FORALL x:
                  (FORALL y: TC(R)(x, y) => P(y))
                      => P(x))
            =>
              (FORALL x: P(x)))
```

This lemma is specified using the transitive closure operator `TC`, that is defined similarly to `RTC` by using `IUnion` and `iterate`. Notice that the relation `R` is of type `noetherian`. Its application depends on an adequate instantiation of the predicate `P`.

4. Verification

A complete proof summary of the `ars` theory including the theory `newman_yokouchi` that was obtained by using the PVS tool ProofLite [11] is available in [6] and also can be generated by downloading the files as reported in the introduction.

The verification of Newman's and Yokouchi's Lemmas consists of 1857 lines (168247 bytes) of proofs. To prove the Newman's Lemma we used 114 proof steps, and to prove the Yokouchi's Lemma we used 225 proof steps. Here we just present the relevant (small) fragment of the proof tree.

In addition to the standard proof techniques available in PVS we have used some strategies available in the PVS packages Field [12] and Manip [20] for algebraic manipulation.

4.1. Verification of Newman's Lemma

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))
```

Notice that the rewriting relation `R` is correctly universally quantified since it is declared as a variable in the theory.

The first command `(skeep)` introduces Skolem constants for the universally quantified variables in the theorem.

```
Rerunning step: (skeep)

Newman_lemma :

{-1}  noetherian?(R)
  |-------
{1}   (confluent?(R) <=> local_confluent?(R))
```

Applying the conjunctive splitting command `(split)` to the goal yields two subgoals. The first subgoal, `Newman_lemma.1`, is to demonstrate that confluence implies locally confluence, which is easily proved. The second subgoal, `Newman_lemma.2`, is the truly interesting one.

```
Rerunning step: (split) Splitting
conjunctions, this yields 2 subgoals:

Newman_lemma.1 :

[-1]  noetherian?(R)
  |-------
{1}   confluent?(R) => local_confluent?(R)

Newman_lemma.2 :

[-1]  noetherian?(R)
  |-------
{1}   local_confluent?(R) => confluent?(R)
```

For proving the later subgoal, after disjuntive simplification with `flatten`, one introduces the noetherian induction scheme `noetherian_induction` as an antecedent formula. This gives the sequent:

```
Rerunning step: (lemma "noetherian_induction")

Newman_lemma.2 :

{-1}  FORALL (R: noetherian[T], P: PRED[T]):
        (FORALL (x: T):
         (FORALL (y: T): TC(R)(x, y) => P(y)) => P(x))
         => (FORALL (x: T): P(x))
[-2]  local_confluent?(R)
[-3]  noetherian?(R)
  |-------
[1]   confluent?(R)
```

Then, the predicate of the induction scheme is adequately instantiated using the command `inst` with the predicate:

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

This gives the following sequent.

```
Newman_lemma.2 :

{-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)))
       =>
         (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)
```

The subgoals `Newman_lemma.2.1` and `Newman_lemma.2.2` presented below are then obtained by applying the command `(split)` that splits the implication of the first antecedent. The first subgoal is easily verified by expanding the definition of the predicate `confluent?` by applying the command `(expand "confluent?")` and then Skolemnization.

```
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)
```

The second subgoal requires to prove `P(x)` under the assumption `P(y)` for all `y` such that $x \rightarrow^{+} y$:

```
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))
[2]   confluent?(R)
```

Since formulas in the succedent are connected by disjunction, one needs only to prove that the antecedents imply the first succedent. Thus the second succedent can be hidden by applying the command `(hide 2)` and then by applying the sequence of defined strategies (skeep2), expand-closure "RTC" -2) and (expand-closure "RTC" -3), and (skolem2 "i" "j") 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, we analyze 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., $x \to x1 \to^* b$ and $x \to x2 \to^* c$ corresponds to the following sequent obtained after some simplifications. Compare with the diagram of Figure 1 (replacing some symbols).

```
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 (b_1, c_1: T):
            RTC(R)(y, b_1) & RTC(R)(y, c_1) =>
             joinable?(R)(b_1, c_1))
[-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?, instantiating with x, x1 and x2 using (inst), and applying the defined strategy (join-skolem "u"). Invoking the lemma R_subset_TC, which states that a relation is contained in its transitive closure, one proves that $x \to^+ x1$ and $x \to^+ x2$. So, the existence of v and w follows by induction hypothesis that is instantiating [-3] conveniently, and the lemma follows.

4.2. Verification of Yokouchi's Lemma

The verification starts with the sequent:

```
Yokouchi_lemma :

   |-------
{1}   FORALL (R, S: PRED[[T, T]]):
        (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))
```

After Skolemnization and propositional flattening, one introduces the auxiliary lemma `Yokouchi_lemma_ax1` corresponding to the generalization D' in Figure 2. Then one obtains the new goal:

```
Rerunning step: (lemma "Yokouchi_lemma_ax1")

Yokouchi_lemma :

{-1}  FORALL (R, S: PRED[[T, T]]):
        (noetherian?(R) & confluent?(R) &
           (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))))
       =>
        (FORALL x, y, z: (S(x,y) & RTC(R)(x,z)) =>
          (EXISTS (w: T):
             RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w)))
[-2]  noetherian?(R)
[-3]  confluent?(R)
[-4]  diamond_property?(S)
[-5]  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))
   |-------
[1]   diamond_property?(RTC(R) o S o RTC(R))
```

Notice that the antecedents `[-2]`, `[-3]` and `[-5]` correspond to all the hypotheses of {`-1`}. Then, after a suitable instantiation of {`-1`} and propositional simplification with the command `(prop)` one obtains the goal:

```
Rerunning step: (prop)
```

```
Yokouchi_lemma :

{-1}  FORALL x, y, z:
        (S(x,y) & RTC(R)(x,z)) =>
          (EXISTS (w: T):
             RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w))
[-2]  noetherian?(R)
[-3]  confluent?(R)
[-4]  diamond_property?(S)
[-5]  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))
  |-------
[1]   diamond_property?(RTC(R) o S o RTC(R))
```

Then, the definition of `diamond_property?` is expanded and the noetherian induction scheme introduced instantiating its predicate as:

```
LAMBDA (a:T):(FORALL (b,c:T):
     (RTC(R) o S o RTC(R))(a,c) & (RTC(R) o S o RTC(R))(a,b)
   => (EXISTS (d:T):
   (RTC(R) o S o RTC(R))(b,d) AND (RTC(R) o S o RTC(R))(c,d)))
```

After splitting conjunctions, one obtains the following two subgoals:

```
Yokouchi_lemma.1 :

{-1}  FORALL (x: T):
       FORALL (b, c: T):
        (RTC(R) o S o RTC(R))(x,c) & (RTC(R) o S o RTC(R))(x,b)
      =>
       (EXISTS (d: T):
        (RTC(R) o S o RTC(R))(b,d) & (RTC(R) o S o RTC(R))(c,d))
[-2]  FORALL x, y, z: (S(x,y) & RTC(R)(x,z)) =>
       (EXISTS (w: T): RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w))
[-3]  noetherian?(R)
[-4]  confluent?(R)
[-5]  FORALL (x: T), (y:T), (z: T):
       S(x,y) & S(x,z) => (EXISTS (r: T): S(y,r) & S(z,r))
[-6]  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))
[-7]  (RTC(R) o S o RTC(R))(x,y)
[-8]  (RTC(R) o S o RTC(R))(x,z)
  |-------
[1]   EXISTS (r: T):
        (RTC(R) o S o RTC(R))(y,r) & (RTC(R) o S o RTC(R))(z,r)
```

and

```
Yokouchi_lemma.2 :

[-1]  FORALL x, y, z:
       (S(x,y) & RTC(R)(x,z)) =>
        (EXISTS (w: T): RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w))
[-2]  noetherian?(R)
[-3]  confluent?(R)
[-4]  FORALL (x: T), (y: T), (z: T):
       S(x,y) & S(x,z) => (EXISTS (r: T): S(y,r) & S(z,r))
[-5]  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))
[-6]  (RTC(R) o S o RTC(R))(x,y)
[-7]  (RTC(R) o S o RTC(R))(x,z)
  |-------
{1}   FORALL (x: T):
      (FORALL (y: T):
        TC(R)(x,y) =>
         (FORALL (b, c: T):
          (RTC(R) o S o RTC(R))(y,c) & (RTC(R) o S o RTC(R))(y,b)
          =>
           (EXISTS (d: T):
            (RTC(R) o S o RTC(R))(b,d) &
             (RTC(R) o S o RTC(R))(c,d))))
      =>
       (FORALL (b, c: T):
        (RTC(R) o S o RTC(R))(x,c) & (RTC(R) o S o RTC(R))(x,b)
       =>
        (EXISTS (d: T):
        (RTC(R) o S o RTC(R))(b,d) & (RTC(R) o S o RTC(R))(c,d)))
[2]   EXISTS (r: T):
        (RTC(R) o S o RTC(R))(y,r) & (RTC(R) o S o RTC(R))(z,r)
```

The subgoal `Yokouchi_lemma.1` is easily verified instantiating adequately the antecedent `[-1]` and asserting. The subgoal `Yokouchi_lemma.2` is proved following the scheme in Figure 3 as detailed below.

1. First step: introduce Skolem constants and consider the cases $\mathtt{x} = z_1$ and/or $\mathtt{x} = y_1$. The goal below corresponds to the case "and" (second diagram of the Figure 3 renaming some variables).

```
Yokouchi_lemma.2.1.1 :

{-1}  j = 0
[-2]  i = 0
[-3]  FORALL (y: T):
       TC(R)(x!1,y) =>
        (FORALL (b, c: T):
         (RTC(R) o S o RTC(R))(y,c) &
         (RTC(R) o S o RTC(R))(y,b)
         =>
          (EXISTS (d: T):
                (RTC(R) o S o RTC(R))(b,d) &
                (RTC(R) o S o RTC(R))(c,d)))
{-4}  iterate(R,0)(x!1,z1)
[-5]  S(z1,z2)
[-6]  RTC(R)(z2,c!1)
[-7]  iterate(R,0)(x!1,y1)
[-8]  S(y1,y2)
[-9]  RTC(R)(y2,b!1)
[-10] FORALL x, y, z:
        (S(x,y) & RTC(R)(x,z)) =>
         (EXISTS (w: T): RTC(R)(y,w) &
         (RTC(R) o S o RTC(R))(z,w))
[-11] noetherian?(R)
[-12] confluent?(R)
[-13] FORALL (x: T), (y: T), (z: T):
        S(x,y) & S(x,z) => (EXISTS (r: T): S(y,r) & S(z,r))
[-14] 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))
  |-------
[1]   EXISTS (d: T):
        (RTC(R) o S o RTC(R))(b!1,d) &
        (RTC(R) o S o RTC(R))(c!1,d)
```

2. Second step: invoke the lemma `iterate_RTC` which states that for all `n`, `iterate(R,n)` $\subseteq$ `RTC(R)`; expand the definitions of composition of relations, `confluent?` and `joinable?`; hide irrelevant formulas; and then, by applying disjunctive simplification one obtains the goal:

```
Yokouchi_lemma.2.2.1.1.2 :

{-1}  EXISTS (y_1: T):
        (EXISTS (y: T): RTC(R)(w4,y) &
                        S(y,y_1)) & RTC(R)(y_1,d)
[-2]  RTC(R)(c!1,w4)
  |-------
{1}   EXISTS (y_1: T):
        (EXISTS (y: T): RTC(R)(c!1,y) &
                        S(y,y_1)) & RTC(R)(y_1,d)
```

3. Third step: conclude applying the auxiliary lemma `Yokouchi_lemma_ax1`, the confluence of `R` and induction hypotheses.

5. PVS Defined Strategies Used in the Proofs

PVS provides a simple language to combine sequences of commonly used proof steps into strategies [1]. These strategies can then be used as prover commands. To facilitate and reduce the size of our proofs some commonly used sequences of proof commands were turned into strategies. Two simple ones are discussed below.

In some proofs, it was necessary to firstly, `expand` the definition of *joinable*; afterward, to introduce `skolem` constants and finally, to apply disjunctive simplification (`flatten`). The strategy `join-skolem` accomplishes this.

```
(defstep join-skolem (var1 fnum)
  (then (expand "joinable?" fnum) (skolem * var1) (flatten))
  "Expanding joinable?, Skolemizing, and
   Applying disjunctive simplification.")
```

Another useful defined strategy is `expand-closure`, that expands the definition of closure relation according to input `closure` which can be either `rtc` (Reflexive Transitive Closure) or `ec` (Equivalence Closure) or `rc` (Reflexive Closure) or `sc` (Symmetric Closure). This strategy uses another one called `expand-um` which expands the definitions of `union` and `member`.

```
(defstep expand-closure (closure fnum)
  (if (equal closure ‘rtc)
      (then (expand "RTC" fnum ) (expand "IUnion" fnum))
      (if (equal closure ‘ec)
          (then (expand "EC" fnum)
                (expand "RTC" fnum)
                (expand "IUnion" fnum))
          (if (equal closure ‘rc)
              (then (expand "RC" fnum) (expand-um fnum))
              (if (equal closure ‘sc)
                  (then (expand "SC" fnum) (expand-um fnum))
                  (skip)))))
  "Expanding the definition of ~A.")
```

In the description of the verification of Newman’s and Yokouchi’s lemmas we have mentioned the application of the defined strategies (`expand-closure`, `join-skolem`, `skeep2`, `skolem2`), which are available together with the files of the `ars` theory and proofs at `www.mat.unb.br/∼ayala/publications.html`.

6. Conclusions and Future Work

This paper illustrates that the `ars` PVS theory previously presented in [6] is in fact adequate for expressing and verifying (well-known) non elementary results of the theory of ARSs. The verifications of Newman’s and Yokouchi’s lemmas were described focusing on the steps related with the applications of noetherian induction. Also it should be stressed here that although this work does not advance the state of the art in the formalization of mathematics since specifications of ARSs and even of TRSs are available since the development of the Rewriting Rule Laboratory (RRL) in the 1980’s [9], it is of practical interest since the availability of rewriting proving technologies are essential in any modern proof assistant and to the best of our knowledge neither rewriting theories nor rewriting libraries are available in PVS currently.

As current work we are developing a more elaborated PVS theory for dealing with TRSs that is of interest to verify the correction of concrete rewriting based specifications of computational objects such as reconfigurable hardware as mentioned in the introduction. By this extension rewriting strategies and new tactic-based proving techniques will be available in PVS in a natural manner. For this purpose, the type of `terms` built over a signature of function symbols is specified as an abstract data type ([15]) with the type of function symbols and the type of variables as its parameters. In the `trs` theory the type `terms` is the actual parameter of the theory `ars[T]`. From this point, term `positions` are given as usual by finite sequences of naturals, and useful operations on terms such as `subterm` at a given position and `replacement` of a subterm at a given position by using recursive declarations; `substitutions` are functions from variables into `terms`. All `ars` definitions and results hold for the reduction relation induced over `terms` by an specific TRS which is specified as a binary relation over `terms`. The induced reduction relation is given by closing the rewriting one under substitutions and structure of terms (signature operations) as it is formalized in the standard rewriting literature [4, 19].

References

[1] M. Archer, B. Di Vito, and C. Muñoz. Developing user strategies in PVS: A tutorial. In *Proceedings of Design and Application of Strategies/Tactics in Higher Order Logics STRATA'03*, NASA/CP-2003-212448, NASA LaRC,Hampton VA 23681-2199, USA, September 2003.

[2] M. Ayala-Rincón, 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 Trans. Design Autom. Electr. Syst.*, 11(2):251–281, 2006.

[3] M. Ayala-Rincón 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*, pages 125–130. ACM Press, 2006.

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

[5] M. Bezem 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(86-100), 2003.

[6] A.L. Galdino and M. Ayala-Rincón. A Theory for Abstract Rewriting Systems in PVS. Available at `www.mat.unb.br/∼ayala/publications.html`. Departamento de Matemática, Universidade de Brasília, 2007.

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

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

[9] D. Kapur 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*, volume 355 of *LNCS*. Springer, April 1989.

[10] C. Morra, J. Becker, M. Ayala-Rincón, and R. W. Hartenstein. FELIX: Using Rewriting-Logic for Generating Functionally Equivalent Implementations. In *15th Int. Con-*

ference on Field Programmable Logic and Applications - FPL 2005, pages 25–30. IEEE CS, 2005.

[11] C. Muñoz. Batch proving and proof scripting in PVS. Report NIA Report No. 2007-03, NASA/CR-2007-214546, NIA-NASA Langley, National Institute of Aerospace, Hampton, VA, February 2007.

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

[13] M. H. A. Newman. On theories with a combinatorial definition of ”equivalence”. *Ann. of Math.*, 43(2):223–243, 1942.

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

[15] Sam Owre and Natarajan Shankar. Abstract datatypes in PVS. Technical Report SRI-CSL-93-9R, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993. Extensively revised June 1997; Also available as NASA Contractor Report CR-97-206264.

[16] O. Rasmussen. The church-rosser theorem in isabelle: A proof porting experiment. Technical Report 364, University of Cambridge, Computer Laboratory, March 1995.

[17] J. L. Ruiz-Reina, J. A. Alonso, M. J. Hidalgo, and F.J. Martín-Mateos. Formalizing Rewriting in the ACL2 Theorem Prover. *Lecture Notes in Computer Science*, 1930, 2001.

[18] N. Shankar. A Mechanical Proof of the Church-Rosser theorem. *Journal of the ACM*, 35:475–522, 1988.

[19] C. J. van Rijsbergen, editor. *Term Rewriting Systems*. Cambridge University Press, 2003.

[20] B.L. Di Vito. *Manip User’s Guide, Version 1.1*. NASA Langley Research Center, Hampton, Virginia, February, 18 2003.

[21] H. Yokouchi. Church-Rosser Theorem for a Rewriting System on Categorical Combinators. *Theoretical Computer Science*, 65(3):271–290, 1989.