A Formalization of Newman's and Yokouchi's Lemmas in a Higher-Order Language

ANDRÉ LUIZ GALDINO
Departamentos de Matemática, Universidade de Brasília and
Universidade Federal de Goiás, Campus de Catalão, Brazil
and
MAURICIO AYALA-RINCÓN
Instituto de Ciências Exatas, Universidade de Brasília, Brazil

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

keywords: Abstract Reduction Systems, Rewriting Systems, Higher-Order Specification Languages, 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 over the PVS prelude libraries for sets and binary relations that is useful for the treatment of Abstract Reduction Systems (ARS) was reported in [8]. In the *theory* `ars` 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 usefulness of `ars` by describing proofs of Newman's and Yokouchi's lemmas. The former proof is a well-known classical application of noetherian induction and the latter is of interest because it is based on several applications of natural and noetherian induction. The inductive proof of the Newman's Lemma given by Huet in [10] is a classical example of proofs in higher-order logic.

The novelty of this work in not to present mechanical proofs of ARS theorems

E-mails: `galdino@catalao.ufg.br, ayala@unb.br`
The authors are partially supported by the Brazilian Research Council, CNPq.

in PVS that were done previously in other proof assistants. In fact, well-known formalizations of Newman's Lemma have been specified in several proof assistants, e.g., ACL2 [21], Coq [11] (an earlier reference reporting a formalization in this proof assistant is [7]), Isabelle [20], Boyer-Moore [23], Otter [3], among others. `ars` is presented as the basis for the formalization of an elaborated and robust PVS *theory* for full Term Rewriting Systems (TRSs), called `trs`, in which more elaborated theorems such as the well-known Knuth-Bendix critical pair criterion were formalized [9]. In [17], Nipkow treated concepts such as confluence and commutation, and formalized in Isabelle/HOL 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. The libraries *CoLoR* [5] and *Coccinelle* [6] 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 neither in `ars` nor in `trs`. In [22], 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.

The main motivation for doing this work of formalization of the theory of (ARSs and) TRSs is the fact that rewriting has been applied to the specification and synthesis of reconfigurable hardware [13] 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 (in fact, [1] introduces a proved correct translation from ELAN rewriting specifications into PVS *theories*). And robust proof rewriting based methods are necessary to deal efficiently with the correctness of these *theories* that come from rewriting based specifications.

Two characteristics of the `ars` *theory* can be remarked: firstly, the use of an higher-order language for the treatment of higher-order properties of reduction and rewriting, makes it natural the specification of higher-order properties of reduction relations such as Newman's lemma; secondly, proofs follow the "almost geometrical style" based in diagrams used in the standard rewriting literature.

Section 2 presents proofs of both lemmas. Sections 3 and 4 describe respectively the specification and verification of both lemmas in PVS. The theory `ars` together with proofs of Newman's, Yokouchi's, among other interesting lemmas is available at `www.mat.unb.br/∼ayala/publications.html`.

2. BACKGROUND: MATHEMATICAL PROOFS

We suppose the reader is familiar with rewriting concepts and standard notations as presented in [2] or [4].

An abstract reduction relation is 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 2.1. *(Newman's Lemma [16]) 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 necessity follows immediately by definition. The sufficiency is proved by noetherian induction using the predicate

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

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(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). $\square$

$$\begin{tikzcd}
x \arrow[r] \arrow[d] \arrow[dr, phantom, "LC"] & z_1 \arrow[r, "*"] \arrow[d, dashed, "*"] \arrow[ddr, phantom, "Ind"] & z \arrow[dd, dashed, "*"] \\
y_1 \arrow[r, dashed, "*"] \arrow[d, "*"'] \arrow[dr, phantom, "Ind"] & u \arrow[d, dashed, "*"] & \\
y \arrow[r, dashed, "*"] & v \arrow[r, dashed, "*"] & w
\end{tikzcd}$$

(a)

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

(b)

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

(c)

Fig. 1. Proof of Newman's Lemma, Diagram D and Generalization of D as D'

Lemma 2.2. *(Yokouchi's Lemma [24]) 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 diagram D as shown in the Figure 1(b) holds. 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 1(c). This generalization is proved by noetherian induction using the predicate

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

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

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

One concludes, 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 2, where C and DP stand for use of confluence of R and diamond property of S hypotheses, respectively. $\square$

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

Fig. 2. Cases $xR \circ R^* \circ S \circ R^*y$ and $xS \circ R^*y$

3. SPECIFICATION

Figure 3 illustrates the hierarchy of *subtheories* of the PVS *theory* `ars`. The *theory* `ars` uses standard PVS libraries as Field and Manip [15] and is composed essentially of `IMPORTING`s of four *subtheories* as can be seen in Table I. These *subtheories* include formalizations of theorems related with the properties of commutation, reduction modulo equivalence, normalization and Newman's and Yokohuchi's lemmas, respectively.

Table I. `ars` PVS *theory*

```
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. The *theory* `ars` imports the PVS library for sets (see `sets_lemmas` in Figure 3) and over this it builds the closure of binary relations and its properties (in the *subtheory* `relations_closure`) that are necessary for formalizing ARS terminology (in the *subtheory* `ars_terminology`) and theorems. Let consider a binary relation `R` over `T`, specified in PVS as `R: VAR pred[[T, T]]`. 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, denoted as `RTC`, of a relation `R` (i.e., `R`*) is specified as the union of the iterations `iterate(R,i)`, for all `i` $\geq$ 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 `orders` library. The operators `iterate` and `IUnion`, available in the PVS prelude *theory* are specified as

```
iterate(R,i): RECURSIVE PRED[[T,T]] =
           IF i = 0 THEN =[T] ELSE iterate(R,i - 1) o R ENDIF
           MEASURE i
IUnion(A): set[T] = {x | EXISTS i: A(i)(x)}
```

where the composition of relations is denoted as `o`. Then `RTC` can be specified as

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

Notice that the type of `RTC` is `reflexive_transitive`. From the intrinsic characteristics of the `RTC` construction, the system of types of PVS will create the necessary proofs obligations to be proved (PVS *Type Correctness Conditions - TCCs*). The predicate `reflexive_transitive?` is used to specify the associated type as follows.

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

where `&` is an abbreviation for `AND`. Notice that the suffix "?" is used for discriminating between predicates and types.

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. `R` denotes a reduction relation over `T` and `n` a natural. `=>`, `<=>` are abbreviations for `IMPLIES` and `IFF`, respectively.

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

The PVS *theory* `newman_yokouchi` presented in Table II 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 reduction relations over `T` and `x, y, z, w` and `u` elements of `T`.

Newman's Lemma specification is straightforward and based on predicates over reduction relations. The second lemma, `Yokouchi_lemma_ax1`, corresponds to the generalization D' of the diagram D presented in Figure 1.

The *subtheories* `results_confluence[T]` and `noetherian[T]`, also components of the whole `ars` *theory* (see Figure 3), are imported by the `newman_yokouchi` *theory*. The former contains results about confluence and the latter the definition

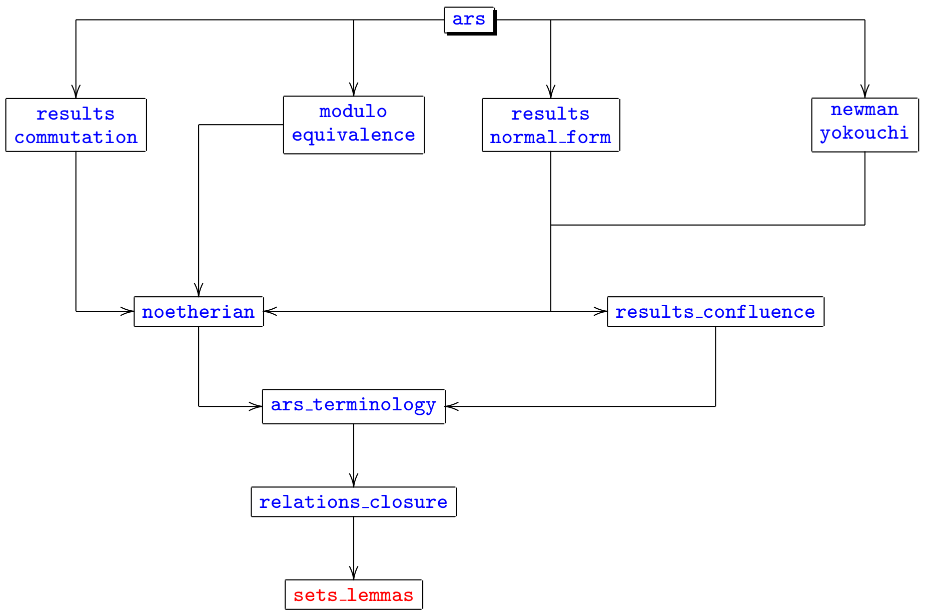

Fig. 3. Hierarchy of the `ars` *theory*

Table II. `newman_yokouchi` PVS *theory*

```
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
```

of noetherianity formulated in terms of the notion of well-foundness 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)
```

Noetherianity is formalized in terms of well foundness, and noetherian induction is then verified using the lemma `wf_induction`, which expresses the principle of *well-founded induction*. The notion of well-founded relations and this principle are available in the prelude *theory* [18].

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

  wf_induction: LEMMA
    (FORALL (p: pred[T]):  (FORALL x: (FORALL y:
       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, y: (TC(R)(x,y) => P(y))  => P(x))  =>   (FORALL x: P(x)))
```

This lemma uses the transitive closure operator `TC`, that is specified similarly to `RTC`. Its application depends on an adequate instantiation of the predicate `P`.

4. VERIFICATION

The verification of Newman's and Yokouchi's lemmas consists of 1857 lines (168247 bytes) of proofs. The formalizations of Newman's and Yokouchi's lemmas use 114 and 225 proof steps, respectively. Here the relevant fragment of the proof trees, focusing on the application of noetherian induction, are presented.

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))
```

The reduction relation `R` is correctly universally quantified, since it was declared as a variable in the *theory* `newman_yokouchi` (see Table II). 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 1(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.

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 skolemization and propositional flattening, one introduces the auxiliary lemma `Yokouchi_lemma_ax1` corresponding to the generalization D' in Figure 1. Then one obtains the new goal:

```
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 the hypotheses of `{-1}`. Then, after a suitable instantiation of `{-1}`, propositional simplification and expansion of the definition `diamond_property?`, one introduces the noetherian induction scheme instantiating its predicate `P` 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)))
```

Then, 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))
...
[-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)))
```

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 2 as detailed below.

(1) First step: introduce Skolem constants and consider the cases $\mathtt{x} = z_1$ and/or $\mathtt{x} = y_1$.
(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, apply disjunctive simplification.
(3) Third step: conclude applying the confluence of `R`, the lemma `Yokouchi_lemma_ax1` and induction hypothesis.

5. CONCLUSIONS AND FUTURE WORK

This work illustrates that the `ars` *theory*, previously presented in [8], is in fact adequate for formalizing (well-known) non elementary higher-order results of the theory of ARSs. The formalizations follow the diagrammatic style of analytic presentations of proofs in the standard theory of ARSs. The formalizations of Newman's and Yokouchi's lemmas were described focusing on the key proof 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 [12], it is doubtless of practical interest. In fact, the availability of rewriting proving technologies are essential in any modern proof assistant and to the best of our knowledge before the development of the `ars` *theory* neither rewriting *theories* nor rewriting libraries were available in PVS.

A *theory* `trs`, which is an extension of `ars`, for dealing with TRSs was also developed and presented in [9]. Sophisticated theorems of the theory of TRS such as the Knuth-Bendix Critical Pairs theorem were formalized. This *theory* is of interest to verify the correction of concrete rewriting based specifications of computational objects such as reconfigurable hardware as mentioned in the introduction. Rewriting based specifications algebraic operators synthesized by the system FELIX over commercial reconfigurable architectures as introduced in [13] (see also [14]) can be translated into the language of PVS to be verified. Inside the *theory* `trs` rewriting strategies and new tactic-based proving techniques are available in PVS in a natural manner now. For this purpose, the type `term` built over a signature of function symbols was specified as an abstract data type [19] with the type of function symbols and the type of variables as its parameters. In the `trs` *theory* the type `term` 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 `term`. All `ars` definitions and results hold for the reduction relation induced over `term` by an specific TRS which is specified as a binary relation over `term`. 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 [2, 4].

References

[1] 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 - SBCCI06*, pages 125–130. ACM Press, 2006.

[2] F. Baader and T. Nipkow. *Term Rewriting and* All That. CUP, 1998.

[3] M. Bezem and T. Coquand. Neman's Lemma - a Case Study in proof automation and geometric logic. *Bull. of the EATCS*, 79(86-100), 2003.

[4] 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.

[5] 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.

[6] 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.

[7] T. Coquand and G. Huet, Concepts mathématiques et informatiques formalisés dans le calcul des constructions, INRIA Raports de Recherche number 463, 1985.

[8] A.L. Galdino and M. Ayala-Rincón. A Theory for Abstract Rewriting Systems in PVS. In *33th Latin American Conference on Informatics - CLEI07.* Available: `www.mat.unb.br/∼ayala/publications.html`.

[9] A.L. Galdino and M. Ayala-Rincón. A PVS Theory for Term Rewriting Systems. Presented in and to appear in ENTCS proc. of Third Workshop on Logical and Semantic Frameworks, with Applications - LSFA, 2008. Available: `www.mat.unb.br/∼ayala/publications.html`.

[10] G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *J.ACM*, 27(4):797–821, 1980.

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

[12] D. Kapur and H. Zhang. An overview of Rewrite Rule Laboratory (RRL), in: N. Dershowitz, editor, *Proc. 3rd Int. Conf. on Rewriting techniques and Applications*, Lecture Notes in Computer Science **355** (1989), pp. 559–563 .

[13] C. Morra, 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*, pages 25–30. IEEE CS, 2005.

[14] C. Morra, Cardoso, J.M.P. and J. Becker. Using Rewriting Logic to Match Patterns of Instructions from a Compiler Intermediate Form to Coarse-Grained Processing Elements IN *IEEE International Parallel and Distributed Processing Symposium - IPDPS 2007*, pages 1–8, 2007.

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

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

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

[18] S. Owre and N. Shankar. The PVS Prelude Library. Technical report, SRI-CSL-03-01, Computer Science Laboratory, SRI International, Menlo Park, CA, 2003.

[19] S. Owre and N. 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.

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

[21] J. L. Ruiz-Reina, J. A. Alonso, M. J. Hidalgo, and F.J. Martín-Mateos. Formalizing Rewriting in the ACL2 Theorem Prover, in: J. A. Campbell and E. Roanes-Lozano, editors, *Proc. Artificial Intelligence and Symbolic Computation, International Conference AISC 2000, Revised Papers*, Lecture Notes in Computer Science **1930** (2001), pp. 92–106.

[22] 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.

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

[24] H. Yokouchi and T. Hikita A rewriting system for categorical combinators with multiple arguments *SIAM Journal on Computing*, 19(1):78–97, 1990.