Observational Proofs with Critical Contexts

Narjes Berregeb Adel Bouhoula Michaël Rusinowitch

LORIA - INRIA Lorraine
615, rue du Jardin Botanique - B.P. 101
54602 Villers-lès-Nancy Cedex, France.
E-mail:{berregeb,bouhoula,rusi}@loria.fr

Abstract. Observability concepts contribute to a better understanding of software correctness. In order to prove observational properties, the concept of *Context Induction* has been developed by Hennicker [10]. We propose in this paper to embed Context Induction in the implicit induction framework of [8]. The proof system we obtain applies to conditional specifications. It allows for many rewriting techniques and for the refutation of false observational conjectures. Under reasonable assumptions our method is refutationally complete, i.e. it can refute any conjecture which is not observationally valid. Moreover this proof system is operational: it has been implemented within the Spike prover and interesting computer experiments are reported.

1 Introduction

Observational concepts are fundamental in formal methods since for proving the correctness of a program with respect to a specification it is essential to be able to abstract away from internal implementation details. Data objects can be viewed as equal if they cannot be distinguished by experiments with observable result. The idea that the semantics of a specification must describe the *behaviour* of an abstract data type as viewed by an external user, is due to [9]. Though a lot of work has been devoted to the semantical aspects of observability (see [2] for a classification), few proof techniques have been studied [17,5,14,13], and even less have been implemented. In this paper we propose an automatic method for proving observational properties of conditional specifications. The method relies on computing families of well chosen contexts, called *critical contexts*, that "cover" in some sense all observable ones. These families are applied as induction schemes. Our inference system basically consists in extending terms by critical contexts and simplifying the results with a powerful rewriting machinery in order to generate new subgoals. An advantage of this approach is that it allows also for disproving false observational conjectures. The method is even refutationally complete for an interesting class of specifications. From a preliminary implementation on top of the Spike prover [8] computer experiments are reported. The given examples have been treated in a fully automatic way by the program.

2 Related works

Hennicker [10] has proposed an induction principle, called *context induction*, which is a proof principle for behavioural abstractions. A property is observationally valid if it is valid for all observable *experiments*. Such experiments are represented by *observable contexts*, which are context of observable sort over the signature of a specification where a distinguished subset of its sorts is specified as observable. Hence, a property is valid for all observable experiments if it is valid for all corresponding observable contexts. A context c is viewed as a particular term containing exactly one variable; therefore, the subterm ordering defines a noetherian relation on the set of observable contexts. Consequently, the principle of structural induction induces a proof principle for properties of contexts of observable sort, which is called *context induction*. This approach provides with a uniform proof method for the verification of behavioural properties. It has been implemented in the system ISAR [1]. However, in concrete examples, this verification is a non trivial task, and requires human guidance: the system often needs a generalization of the current induction assertion before each nested context induction, so that to achieve the proof.

Malcolm and Goguen [14] have proposed a proof technique which simplifies Hennicker proofs. The idea is to split the signature into *generators* and *defined functions*. Proving that two terms are behaviourally equivalent, comes to prove that they give the same result in all observable contexts built from defined functions, provided that the generators verify a congruence relation w.r.t. behavioural equivalence. This proof technique is an efficient optimization of Hennicker proofs.

Bidoit and Henniker [4] have investigated how a first order logic theorem prover can be used to prove properties in an observational framework. The method consists in computing automatically some special contexts called *crucial contexts*, and in enriching the specification so that to automatically prove observational properties. But this method was only developed for the proof of equations and for specifications where only one sort is not observable. Besides, it fails on several examples (cf. *Stack* example), where it is not possible to compute crucial contexts.

Bidoit and Hennicker [5] have also investigated characterization of behavioural theories that allows for proving behavioural theorems with standard proof techniques for first order logic. In particular they propose general conditions under which an infinite axiomatization of the observational equality can be transformed into a finitary one. However, in general there is no automatic procedure for generating such a finite axiomatization of the observational equality.

Puel [16] has adapted Huet-Hullot procedure for proof by consistency w.r.t. the final model. Lysne [13] extends Bachmair's method for proof by consistency to the final algebra framework. The proof technique is based on a special completion procedure whose idea is to consider, not only critical pairs emerging from positioning rewrite rules on equations, but also those emerging from positioning equations on to rewrite rules. This approach is restricted to equations and requires the ground convergence property of the axioms in order to be sound (in our case, ground convergence is needed only for refutational completeness).

3 Basic notions

We assume that the reader is familiar with the basic concepts of algebraic specifications [18], term rewriting and equational reasoning. A many sorted signature Σ is a pair (S, F) where S is a set of *sorts* and F is a set of function symbols. For short, a many sorted signature Σ will simply be denoted by F. We assume that we have a partition of F in two subsets, the first one, C, contains the *constructor symbols* and the second, D, is the set of *defined symbols*. Let X be a family of sorted variables and let $T(F, X)$ be the set of sorted terms. $var(t)$ stands for the set of all variables appearing in t. A term is *linear* if all its variables occur only once in it. If $var(t)$ is empty then t is a *ground* term. The set of all ground terms is $T(F)$. Let A be an arbitrary non-empty set, and let $F_A = \{f_A \mid f \in F\}$ such that if f is of arity n then f_A is a function from A^n to A. The pair (A, F) is called a *Σ-algebra*, and A the *carrier* of the algebra. For sake of simplicity, we will write A to denote the Σ-algebra when F and F_A are non-ambiguous.

A *substitution* assigns terms of appropriate sorts to variables. The domain of η is defined by: $dom(\eta) = \{x \mid x\eta \neq x\}$. If t is a term, then $t\theta$ denotes the application of θ to t. If η applies every variable to a ground term, then η is a ground substitution. We denote by $\equiv$ the syntactic equivalence between objects. Let N^* be the set of sequences of positive integers. For any term t, $Pos(t) \subseteq N^*$ denotes its set of positions and the expression t/u denotes the *subterm of t at a position u*. We write $t[s]_u$ (resp. $t[s]$) to indicate that s is a subterm of t at position u (resp. at some position).The top position is written ε. Let $t(u)$ denote the symbol of t at position u. A position u in a term t is said to be *a strict position* if $t(u) = f \in F$. A position u in a term t such that $t(u) = x$ and $x \in X$, is a *linear variable position* if x occurs only once in t, otherwise, u is a *non linear variable position*. The depth of a term t is defined as follows: $|t| = 0$ if t is a constant or a variable, otherwise, $|f(t_1, \ldots, t_n)| = 1 + max_i|t_i|$. We denote by $\succ$ a transitive irreflexive relation on the set of terms, that is noetherian, monotonic ($s \succ t$ implies $w[s]_u \succ w[t]_u$), stable per instantiation ($s \succ t$ implies $s\sigma \succ t\sigma$) and satisfies the subterm property ($f(\cdots, t, \cdots) \succ t$). The multiset extension of $\succ$ will be denoted by $\gg$. An *equation* is a formula of the form $l = r$. A *conditional* equation is a formula of the following form: $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow l = r$. It will be written $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow l \rightarrow r$ and called a *conditional rule* if $\{l\sigma\} \gg \{r\sigma, a_1\sigma, b_1\sigma, \cdots, a_n\sigma, b_n\sigma\}$ for each substitution σ. The *precondition* of rule $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow l \rightarrow r$ is $\bigwedge_{i=1}^{n} a_i = b_i$. The term l is the *left-hand side* of the rule. A rewrite rule $c \Rightarrow l \rightarrow r$ is *left-linear* if l is linear. A set of conditional rules is called a *rewrite system*. A constructor is *free* if it is not the root of a left-hand side of a rule. Let t be a term in $T(C, X)$, t is called a *constructor* term. A rewrite system R is *left-linear* if every rule in R is left-linear. We define $depth(R)$ as the maximal depth of the strict positions in its left-hand sides. Let R be a set of conditional rules. Let t be a term and u a position in t. We write: $t[l\sigma]_u \rightarrow_R t[r\sigma]_u$ if there is a substitution σ and a conditional equation $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow l = r$ in R such that:

(i) $l\sigma \succ r\sigma$.

(ii) for all $i \in [1 \cdots n]$ there exists c_i such that $a_i\sigma \rightarrow_R^* c_i$ and $b_i\sigma \rightarrow_R^* c_i$.

(iii) $\{t[l\sigma]_u\} \gg \{a_1\sigma,\ b_1\sigma,\ \cdots,a_n\sigma,\ b_n\sigma\}$.
Rewriting is extended to literals and clauses as expected.

specification: STACK
sorts: nat, stack
observable sorts: nat
constructors:
0: $\rightarrow$nat;
s: nat $\rightarrow$nat;
Nil: $\rightarrow$stack;
$push$: nat $\times$ stack $\rightarrow$stack;
defined functions:
top: stack $\rightarrow$ nat;
pop: stack $\rightarrow$ stack;
axioms:
$top(Nil) = 0$
$top(push(i, s)) = i$
$pop(Nil) = Nil$
$pop(push(i, s)) = s$

Fig. 1. Stack specification

A term t is *irreducible* (or in *normal form*) if there is no term s such that $t \rightarrow_R s$. A term t is *ground reducible* iff all its ground instances are reducible. A symbol $f \in F$ is *completely defined* if all ground terms with root f are reducible to terms in $T(C)$. We say that R is *sufficiently complete* if all symbols in D are completely defined. A *clause* C is an expression of the form : $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow \bigvee_{j=1}^{m} a'_j = b'_j$. The clause C is *positive* if $n = 0$. The clause C is a *logical consequence* of E if C is valid in any model of E, denoted by $E \models C$. We say that C is *inductively valid* in E and denote it by $E \models_{Ind} C$ if for any ground substitution σ, (for all i, $E \models a_i\sigma = b_i\sigma$) implies (there exists j, $E \models a'_j\sigma = b'_j\sigma$). We say that two terms s and t are joinable, denoted by $s \downarrow_R t$, if $s \rightarrow_R^* v$ and $t \rightarrow_R^* v$ for some term v. The rewrite system R is *ground convergent* if the terms u and v are joinable whenever $u, v \in T(F)$ and $R \models u = v$.

4 Observational semantics

The notion of *observation technique* have been introduced as a means for describing what is observed in a given algebra. An observational specification is then obtained by adding an observation technique to a standard algebraic specification. The observation technique we use in our method is based on sorts [1]. The semantics we choose is based on a relaxing of the satisfaction relation. The

[1] but it can be easily extended to observations based on operators

notion of *context* is fundamental in all approaches based on such observational semantics. An observational property is obtained by taking into account only observable information. Thus, to show that it is valid, one has to show its validity in all *observable contexts.*

Let $T(F,X)$ be a term algebra and let (S,F) be its signature. A *context* over F is a non-ground term $c \in T(F,X)$ with a distinguished occurrence of a variable called the *context variable* of c. To indicate the context variable z_s occuring in c, we often write $c[z_s]$ instead of c, where s is the sort of z_s. A context reduced to a variable z_s of sort s is called an *empty context* of sort s. The application of a context $c[z_s]$ to a term $t \in T(F,X)$ of sort s, denoted by $c[t]$, is defined by the substitution of z_s by t in $c[z_s]$. The context c is said to be *applicable* to t. By exception, $var(c)$ will denote the set of all variables occurring in c but the context variable of c. A context c is *ground* if $var(c) = \emptyset$. We denote by $|c|$ the depth of c. A *subcontext* (resp. *strict subcontext*) of c, is a subterm (resp. strict subterm) of c with the same contextual variable. A *clausal context* c for a clause C is a list of contexts $< c_1, \dots, c_n, c'_1, \dots, c'_m >$ such that for all $i \in [1..n]$: c_i is applicable to $a_i = b_i$, and for all $j \in [1..m]$: c'_j is applicable to $a'_j = b'_j$. The application of c to C, denoted by $c[C]$, gives the clause $\bigwedge_{i=1}^{n} c_i[a_i] = c_i[b_i] \Rightarrow \bigvee_{j=1}^{m} c'_j[a'_i] = c'_i[b'_i]$. Let $c[z_s]$ and $c'[z'_{s'}]$ be two contexts such that c is of sort s', let t be a term of sort s and σ be a substitution such that $z_s \notin dom(\sigma)$. We use the following notations: $c'[(c[t])]$=$(c'[c])[t]$=$c'[c[t]]$ and $(c[t])\sigma = (c\sigma)[t\sigma]$ =$c[t]\sigma$.

A specification SP is a triple (S,F,E) where (S,F) is a signature and E is a set of conditional equations. An *observational specification* SP_{obs} is a couple (SP, S_{obs}) such that $SP = (S,F,E)$ is a specification and $S_{obs} \subseteq S$ is the set of *observable sorts.* The Stack specification in Figure 1, is an observational specification, where $S_{obs} = \{nat\}$.

An observable term is a term whose sort belongs to S_{obs}. The set of observable contexts is denoted by C_{obs} . An equation $a = b$ is observable if a and b are observable. The precondition of a rule is observable if all its equations are observable. Consider the specification in Figure 1. There are infinitely many observable contexts: $top(z_{stack}), top(pop(z_{stack})), \dots,$ $top(pop(\dots(pop(z_{stack}))\dots)), \dots top(push(z_{stack})), top(push(i, pop(z_{stack}))), \dots$

The notion of observational validity is based on the idea that two objects in a given algebra are observationally equal if they cannot be distinguished by computations with observable results. These computations are formalized by contexts.

Let a and b be two terms. We say that a and b are observationally equal, and we denote it by $E \models_{Obs} a = b$ iff for all $c \in C_{obs}, E \models_{Ind} c[a] = c[b]$. Consider the stack specification in Figure 1. It is easy to see that $push(top(s), pop(s)) = s$ is not satisfied (in the classical sense). However, intuitively, it is observationally satisfied if we just *observe* the elements of the sequences $push(top(s), pop(s))$ and s. This can be formally shown by considering all observable contexts.

The next theorem gives a useful characterization of observational theorems (see e.g. [15]):

Theorem 1. *Suppose that all the preconditions of E are observable. Then $E \models_{Obs} \bigwedge_{i=1}^{n} a_i = b_i \Rightarrow \bigvee_{j=1}^{m} a'_j = b'_j$ iff for all ground substitutions σ, if (for all $i, E \models_{Obs} a_i\sigma = b_i\sigma$) then (there exists j such that $E \models_{Obs} a'_j\sigma = b'_j\sigma$).*

5 Induction schemes

Our purpose in this section is to introduce the ingredients allowing us to prove and disprove behavioural properties. This task amounts in general to check an infinite number of ground formulas for validity, since an infinite number of instances and an infinite number of contexts have to be considered for building these ground instances. This is where induction comes into play. Test substitutions will provide us with induction schemes for substitutions and critical contexts will provide us with induction schemes for contexts. In general, it is not possible to consider all the observable contexts. However, cover contexts are sufficient to prove behavioural theorems by reasoning on the ground irreducible observable contexts rather than on the whole set of observable contexts. In the following, we denote by R a conditional rewriting system with observable preconditions.

Definition 2 (cover set). A *cover set*, denoted by CS, for R, is a finite set of irreducible terms such that for all ground irreducible term s, there exist a term t in CS and a ground substitution σ such that $t\sigma \equiv s$.

We now introduce the notion of *cover context* that is used to schematize all contexts. Note that a *cover context* need not be observable, (unlike *crucial contexts* of [4]). The intuitive idea is to use *cover context* to extend the conjectures by the top in order to create redexes. Then the obtained formulas can be simplified by axioms and induction hypothesis.

Definition 3. [cover context set] A *cover context set* CC is a set of contexts such that: for each ground irreducible context $c_{obs}[z_s] \in C_{obs}$, there exists $c[z_s] \in CC$ and a substitution θ such that $dom(\theta) = var(c)$ and $c\theta$ is a subcontext of c_{obs} .

A cover context set for the specification stack is $\{z_{nat}, top(z_{stack}), pop(z_{stack})\}$. The context $push(i, z_{stack})$ is not a cover context since $top(push(i, z_{stack}))$ and $pop(push(i, z_{stack}))$ are reducible. Note that usually there are infinitely many possible cover context sets. For instance,$\{z_{nat}, top(z_{stack}), top(pop(z_{stack})), pop(pop(z_{stack}))\}$ is also a cover context set.

In the following, we refine cover context sets so that to be able not only to prove behavioural properties, but also to disprove the non valid ones. We need first to introduce the following notions: A context c is *quasi ground reducible* if for all ground substitution τ such that $dom(\tau) = var(c)$, $c\tau$ is reducible. A term t is *strongly irreducible* if none of its non-variable subterms matches a left-hand side of a rule in R. A positive clause $C_{pos} \equiv \bigvee_{i=1}^{n} a_i = b_i$ is *strongly irreducible* if C_{pos} is not a tautology, and the maximal elements of $\{a_i, \; b_i\}$ w.r.t. $\prec$ are strongly irreducible by R.

Cover sets and cover context sets are fundamental for the correctness of our method. However, they cannot help us to disprove the non observationally valid clauses. For this purpose, we introduce a new notion of critical context sets and we use test sets defined in [7].

Definition 4 (test set, test substitution). A *test set* is a cover set which has the following additional properties: (i) the instance of a ground reducible term by a test substitution matches a left-hand side of R. (ii) if the instance of a positive clause C_{pos} by a test substitution σ is strongly irreducible, then $C_{pos}\sigma$ is not inductively valid w.r.t. R. A *test substitution* for a clause C instanciates all induction variables of C by terms taken from a given test set whose variables are renamed.

Definition 5 (critical context set, critical clausal context). A *critical context set* S is a cover context set such that for each positive clause C_{pos}, if $c[C_{pos}]\sigma$ is strongly irreducible where σ is a test substitution of C_{pos} and c is a clausal context of C_{pos}, then $C_{pos}\sigma$ is not observationally valid w.r.t. R. A *critical clausal context* for a clause C is a clausal context for C whose contexts belongs to S.

Test substitutions and critical context sets permit us to refute false conjectures by constructing a counterexample.

Definition 6 (provably inconsistent). Let R be a conditional rewriting system with observable preconditions. We say that $C \equiv \bigwedge_{i=1}^{n} a_i = b_i \Rightarrow \bigvee_{j=1}^{m} a'_j = b'_j$ is *provably inconsistent* if and only if there exists a test substitution σ and a clausal critical context c such that:

(i) for all i, $a_i\sigma = b_i\sigma$ is an inductive theorem w.r.t. R.
(ii) $c[\bigvee_{j=1}^{m} a'_j = b'_j]\sigma$ is strongly irreducible by R.

Provably inconsistent clauses are not observationally valid.

Theorem 7. *Let R be a conditional rewriting system with observable preconditions. Let C be a provably inconsistent clause. Then C is not observationally valid.*

5.1 Computation of test sets

The computation of test sets and test substitutions for conditional specifications is decidable if the axioms are sufficiently complete and the constructors are specified by a set of unconditional equations (see [12]). Unfortunately, no algorithm exists for the general case of conditional specifications. However, in [7], a procedure is described for computing test sets when the axioms are sufficiently complete over an arbitrary specification of constructors.

5.2 Computation of critical contexts

Let us first introduce the following lemma which gives us a useful characterization of critical context sets:

Lemma 8. *Let R be a conditional rewriting system. Let CC be a cover context set that has the following properties:*

(i) any non-observable context in CC has variables at depth greater than or equal to $depth(R)$.
(ii) for each context $c[z_s] \in CC$, there exists an observable context c_{obs} such that $c_{obs}[c]$ is strongly irreducible.

Then, CC is a critical context set for R.

Proof. Let C be a positive clause such that $c[C]\sigma$ is strongly irreducible, where σ is a test substitution of C and c is a critical clausal context of C. Let us prove that $C\sigma$ is not observationally valid. By (ii), there exists an observable clausal context c_{obs} such that $c_{obs}[c]$ is strongly irreducible. Now, using (i), we conclude that $c_{obs}[c[C]]\sigma$ is also strongly irreducible. Then, $c_{obs}[c[C]]\sigma$ is a provably inconsistent clause w.r.t. Definition 11 in [7]. By Theorem 12 in [7], $c_{obs}[c[C]]\sigma$ is not inductively valid. Thus, $R \not\models_{Obs} C\sigma$.

```
$CC_0 := \{c \in T(F,X) \mid |c| \leq depth(R),\ c \in C_{obs},\ c$ is not quasi ground reducible, and does not contain any observable strict subcontext$\}$

$T_0 := \{c \in T(F,X) \mid |c| = depth(R),\ c \notin C_{obs},\ c$ is not quasi ground reducible, $c$ does not contain any observable subcontext, and all variables (including the context one) in $c$ occur at $depth(R)\}$.

repeat
  $CC_{i+1} := CC_i \cup \{c \in T_i \mid \exists c_i \in CC_i$ such that $c_i[c]$ is not quasi ground reducible$\}$
  $T_{i+1} := T_i \setminus CC_{i+1}$
until $CC_{i+1} = CC_i$

output: $CC_i$
```

Fig. 2. Computation of Critical Contexts

Now, let us present our method for constructing such critical contexts. The idea of our procedure is the following: starting from the non quasi ground reducible observable contexts of depth smaller than or equal to $depth(R)$, we construct all contexts that can be embedded in one of those observable contexts, to give a non quasi ground reducible and observable context. It can be proved, by reduction to ground reducibility, that quasi ground reducibility is decidable too for equational systems and semi-decidable for conditional rewrite systems [11]. The following remark is also useful: given a context $c[z_s]$ of the form $f(t_1, \ldots, t_n)$ where f is a completely defined function and for all $i \in [1..n]$, t_i is a constructor term. If z_s does not appear at an induction position of f, then $c[z_s]$ is quasi ground reducible.

Theorem 9. *Let R be a rewriting system and CC be the result of the application of the procedure given in Figure 2. Then:*

- *CC is a cover context set for R.*
- *if R is equational and left-linear then CC is a critical context set for R.*

Proof. It is relatively easy to show that CC is a cover context set for R. Now, assume that R is equational and leftlinear and let us prove that CC is also a critical context set for R. By construction, any non-observable context in CC has variables at depth greater than or equal to $depth(R)$. Now, since R is equational, any non quasi ground reducible context is necessarily strongly irreducible. On the other hand, R is left-linear and the variables of non-observable context occur at $depth(R)$, then for each context $c[z_s] \in CC$, there exists i such that $c \in CC_i$, we can show that there exists an observable context c_{obs} such that $c_{obs}[c]$ is strongly irreducible. The proof is done by induction on i.

Example 10. Consider the Stack specification in Figure 1. We have $depth(R) = 1$, then:
$CC_0 = \{z_{nat}, top(z_{stack})\}$,
$T_0 = \{pop(z_{stack}), push(i, z_{stack})\}$.
$CC_1 = \{z_{nat}, top(z_{stack})\} \cup \{pop(z_{stack})\}$,
$CC = CC_1$ is a critical context set for R.

Example 11. Consider the List specification in Figure 3. We have: $depth(R) = 1$, then:
$CC_0 = \{z_{nat}, z_{bool}, in(x, z_{list})\}$,
$T_0 = \{union(z_{list}, x), insert(x, z_{list})\}$,
$CC_1 = CC_0 \cup \{union(z_{list}, x)\}$.
$CC = CC_1$ is a cover context set for R. In fact, $union(x, z_{list})$ is quasi ground reducible and $in(y, union(z_{list}, x))$ is not quasi ground reducible since $in(0, union(z_{list}, Nil))$ is irreducible, but $in(y, insert(x, z_{list}))$ is quasi ground reducible.

It is possible to compute critical context sets in the case where R is a conditional rewriting system. It is sufficient to apply our procedure given in Figure 2 to compute a cover context set CC, and then to check that for each non observable context $c \in CC$, there exists an observable context c_{obs} such that $c_{obs}[c]$ is strongly irreducible. In Example 11, we have $in(x, (union(z_{list}, y)$ is strongly irreducible, then we conclude that $CC = \{z_{nat}, z_{bool}, in(x, z_{list}), union(z_{list}, x)\}$ is a critical context set for R.

specification: LIST
sorts: nat, bool, list
observable sorts: nat, bool
constructors:
0: $\rightarrow$nat;
s: nat $\rightarrow$nat;
Nil: $\rightarrow$list;
insert: nat $\times$ list $\rightarrow$list;
True: $\rightarrow$bool;
False: $\rightarrow$bool;
defined functions:
union: list $\times$ list $\rightarrow$ list;
in: nat $\times$ list $\rightarrow$ bool;
eq: nat $\times$ nat $\rightarrow$ bool;
axioms:
$union(Nil, l) = l$
$union(insert(x, l), l1) = insert(x, union(l, l1))$
$in(x, Nil) = False$
$eq(x, y) = True \Rightarrow in(x, insert(y, l)) = True$
$eq(x, y) = False \Rightarrow in(x, insert(y, l)) = in(x, l)$
$eq(0, 0) = True$
$eq(0, s(x)) = False$
$eq(s(x), 0) = False$
$eq(s(x), s(y)) = eq(x, y)$

Fig. 3. List specification

6 Inference system

The inference system we use (see Figure 4) is based on a set of transition rules applied to (E, H), where E is the set of conjectures to prove and H is the set of induction hypotheses. The initial set of conditional rules R is oriented with a well founded ordering. An *I-derivation* is a sequence of states: $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \dots (E_n, H_n) \vdash_I \dots$. We say that an I-derivation is *fair* if the set of persistent clauses $(\cup_i \cap_{j \geq i} E_j)$ is empty. Context induction is performed implicitly by the **Generation** rule. An equation is selected in a clause and it is extended by critical contexts and test sets. These extensions are rewritten by R either by conditional rewriting or by case analysis. The resulting conjectures are collected in $\bigcup_{c,\sigma} E_{c,\sigma}$. **Case Simplification** illustrates the case reasoning: it simplifies a conjecture with conditional rules provided that the disjunction of their preconditions [2] is inductively valid in R.

Definition 12 (Case Analysis). Let R be a set of conditional rules and let $l \vee r$ be a clause. **Case Analysis**$(l[g\sigma]_u, r) = \{P_1\sigma \Rightarrow l[d_1\sigma]_u \vee r; \cdots; P_n\sigma \Rightarrow l[d_n\sigma]_u \vee r\}$ **if** $\forall i \in [1 \cdots n] : P_i \Rightarrow g \rightarrow d_i \in R$ **and** $R \models_{Ind} P_1\sigma \vee \cdots \vee P_n\sigma$.

[2] Recall that the preconditions of the axioms in R are assumed to be observable

The rule **Context subsumption** appeared to be very useful for manipulating non orientable conjectures.

An I-derivation *fails* when there exists a conjecture such that no rule can be applied to it. An I-derivation *succeeds* if all conjectures are proved.

Theorem 13 (correctness of successful I-derivations). *Let* $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \ldots$ *be a fair I-derivation. If it succeeds then* $R \models_{Obs} E_0$.

Proof. The proof is done by contradiction. Suppose $R \not\models_{Obs} E_0$ and let $C \in \cup_i E_i$ be a minimal counterexample w.r.t. a well founded ordering on clauses extending $\succ_p$. We can easily show, as in [8], that no inference rule can be applied to C. Hence C persists in the derivation contradicting the fairness hypothesis.

Theorem 14 (correctness of disproof). *Let* $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ *be an I-derivation. If there exists j such that* **Disproof** *is applied to* (E_j, H_j)*, then* $R \not\models_{Obs} E_0$.

Proof. If there exists j such that **Disproof** is applied to (E_j, H_j), then by Theorem 7, we conclude that $R \not\models_{Obs} E_j$. Now, to prove that $R \not\models_{Obs} E_0$, it is sufficient to prove the following claim: Let $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ be an I-derivation step. If $\forall i \leq j, R \models_{Obs} E_i$ then $R \models_{Obs} E_{j+1}$. If $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by a simplification rule, then the equations which are used for simplification occur in some E_k $(k \leq j)$ and therefore are observationally valid in R by assumption. Hence, E_{j+1} is observationally valid too in R. If $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by **Generation** on $C \in E_j$, every auxiliary equation which is used for rewriting an instance of C by a critical context c and a test substitution σ, is either in R or E_k $(k \leq j)$ and hence E_{j+1} is valid in R.

Now we consider boolean specifications. To be more specific, we assume there exists an observable sort *bool* with two free constructors $\{true,\ false\}$. The sort *bool* will be observable. Every rule in R is of type: $\bigwedge_{i=1}^{n} p_i = p'_i \Rightarrow s \rightarrow t$ where for all i in $[1 \cdots n]$, $p'_i \in \{true,\ false\}$. Conjectures will be *boolean clauses*, i.e. clauses whose negative literals are of type $\neg(p = p')$ where $p' \in \{true, false\}$. Let $f \in D$, a completely defined symbol in R. Then f is *strongly complete [7] w.r.t* R if for all the rules $p_i \Rightarrow f(t_1, \ldots, t_n) \rightarrow r_i$ whose left-hand sides are identical up to a renaming μ_i, we have $R \models_{Ind} \bigvee_{i=1}^{n} p_i \mu_i$. We say that R is *strongly complete* if for all $f \in D$, f is strongly complete w.r.t R.

Theorem 15 (refutational completeness). *Let R be a conditional rewrite system. Assume that R is ground convergent and strongly complete. Let E_0 be a set of boolean clauses. Then* $R \not\models_{Obs} E_0$ *iff all fair derivations issued from* $(E_0, \emptyset)$ *fail.*

Proof. The proof follows the line of the corresponding Theorem 6.5 in [6] that was given for the initial semantics.

Generation: $\dfrac{(E \cup \{l \vee r\}, H)}{(E \cup (\bigcup_{c,\sigma} E_{c,\sigma}), H \cup \{C\})}$

if $\begin{cases} \text{for all test substitution } \sigma \text{ and for all critical context } c\text{:} \\ \quad \textbf{either } c[l]\sigma \vee r\sigma \text{ is a tautology, then } E_{c,\sigma} = \emptyset \\ \quad \textbf{or } c[l]\sigma \rightarrow_R l'\text{, then } E_{c,\sigma} = \{l' \vee r\sigma\} \\ \quad \textbf{otherwise } E_{c,\sigma} = \textbf{Case Analysis}(c[l]\sigma, r\sigma) \end{cases}$

Case Simplification: $\dfrac{(E \cup \{l \vee r\}, H)}{(E \cup E', H)}$

if $E' = \textbf{Case Analysis}(l, r)$

Simplification: $\dfrac{(E \cup \{(a = b) \vee r\}, H)}{(E \cup \{(a' = b) \vee r\}, H)}$

if $\begin{cases} a \rightarrow_R a'\text{, or} \\ a[v\lambda] \rightarrow_{H \cup E} a[w\lambda] \text{ by } v = w \text{ where } v \succ w \text{ and } (v\lambda \prec a \text{ or } w\lambda \prec b) \end{cases}$

Subsumption: $\dfrac{(E \cup \{C\}, H)}{(E, H)}$

if C is subsumed by a clause of $R \cup H \cup E$

Context Subsumption: $\dfrac{(E \cup \{C\}, H)}{(E, H)}$

if $\begin{cases} \text{there exists a clause } C' \in R \cup H \cup E \text{ and a clausal context } c \\ \text{such that } c[C'] \text{ subsumes } C \end{cases}$

Delete: $\dfrac{(E \cup \{C\}, H)}{(E, H)}$

if C is a tautology

Disproof: $\dfrac{(E \cup \{C\}, H)}{\mathit{Disproof}}$

if C is provably inconsistent

Fig. 4. Inference System I

7 Computer experiments

We have implemented these results in the Spike prover, written in Caml Light.

Example 16 (Stacks). We proved automatically that $push(top(S), pop(S)) = S$ is a behavioural property of the stack specification (see Figure 1). Note that this example fails with the approach of [4], since it is not possible to compute automatically a set of *crucial contexts*: if two stacks have the same top they are not necessarily equal. In the approach of [10], we have to introduce an auxiliary function *iterated_pop* : $nat \times stack \rightarrow stack$ such that $iterated\_pop(n, s)$ iterates n times *pop*. This is easy because *pop* is unary. The function *iterated_pop* is defined by:

$$iterated\_pop(0, s) = s,\ iterated\_pop(n + 1, s) = iterated\_pop(n, pop(s))$$

Then, we have to prove the property for all contexts of the form $top(iterated\_pop(x, c[z_{stack}]))$. However, this schematization of contexts could be more complicated in case of a function of arity greater than two. So, this process seems to be not easy to automatize in general. In the approach of [14], this problem remains too.

Now, let us describe our proof. The prover computes first a test set for R and the induction positions of functions, which are necessary for inductive proofs. It also computes a critical context. These computation are done only once and before the beginning of the proof.

```
test set of R:
 -> elem = {0, s(x1)}
 -> stack = {Nil ; push(x1,x2)}

critical contexts of R:
 -> stack = {pop(x1)}
 -> elem = {x1, top(x1)}

induction positions of functions:
 -> top : [[1]]
 -> pop : [[1]]

E0 = {push(top(x1),pop(x1)) = x1}

Application of generation on:
    push(top(x1),pop(x1)) = x1 :
 1) Nil = pop(Nil) ;
 2) x2 = pop(push(x1,x2)) ;
 3) 0 = top(Nil) ;
 4) x1 = top(push(x1,x2))

E1 = {Nil = pop(Nil) ;
      x2 = pop(push(x1,x2)) ;
      0 = top(Nil) ;
      x1 = top(push(x1,x2))}
H1 = {push(top(x1),pop(x1)) = x1}

Delete  Nil = pop(Nil)
 it is subsumed by:pop(Nil) = Nil of R

Delete  x2 = pop(push(x1,x2))
 it is subsumed by:pop(push(x1,x2)) = x2 of R

Delete  0 = top(Nil)
 it is subsumed by:top(Nil) = 0 of R

Delete  x1 = top(push(x1,x2))
 it is subsumed by:top(push(x1,x2)) = x1 of R
```

```
E2 = {}
H2 = {push(top(x1),pop(x1)) = x1}

The initial conjectures are observationally valid in R
```

Example 17 (Lists). Consider now the specification *list* in Figure 3. The theorem $insert(x1, insert(x1, x2)) = insert(x1, x2)$ is automatically proved.

```
test set of R:
 -> nat = {0 ; s(x1)}
 -> list = {Nil ; insert(x1,x2)}
 -> bool = {False ; True}

critical contexts of R:
 -> bool = {x1, in(x1,x2)}
 -> list = {x1, union(x1,x2)}

induction positions of functions:
 -> union : [[1]]
 -> in : [[2]]
 -> eq : [[1];[2]]

E0 = {insert(x1,insert(x1,x2)) = insert(x1,x2)}

Application of generation on:
    insert(x1,insert(x1,x2)) = insert(x1,x2) :
 1) eq(x3,x1) = True => True = in(x3,insert(x1,x2)) ;
 2) eq(x3,x1) = False => in(x3,insert(x1,x2)) = in(x3,insert(x1,x2)) ;
 3) eq(x3,x1) = False, eq(x3,x1) = True ;
 4) insert(x1,insert(x1,union(x2,x4))) = union(insert(x1,x2),x4)

Delete  eq(x3,x1) = False => in(x3,insert(x1,x2)) = in(x3,insert(x1,x2))

Delete  eq(x3,x1) = True => True = in(x3,insert(x1,x2))
 it is subsumed by:eq(x1,x2) = True => in(x1,insert(x2,x3)) = True of R

E1 = {eq(x3,x1) = False, eq(x3,x1) = True ;
      insert(x1,insert(x1,union(x2,x4))) = union(insert(x1,x2),x4)}
H1 = {insert(x1,insert(x1,x2)) = insert(x1,x2)}

Simplification of:
    insert(x1,insert(x1,union(x2,x4))) = union(insert(x1,x2),x4) by H1:
    insert(x1,union(x2,x4)) = union(insert(x1,x2),x4)

E2 = {eq(x3,x1) = False, eq(x3,x1) = True ;
      insert(x1,union(x2,x4)) = union(insert(x1,x2),x4)}
H2 = {insert(x1,insert(x1,x2)) = insert(x1,x2)}

Simplification of:
    insert(x1,union(x2,x4)) = union(insert(x1,x2),x4) by R:
    insert(x1,union(x2,x4)) = insert(x1,union(x2,x4))
```

```
E3 = {eq(x3,x1) = False, eq(x3,x1) = True ;
      insert(x1,union(x2,x4)) = insert(x1,union(x2,x4))}
H3 = {insert(x1,insert(x1,x2)) = insert(x1,x2)}

Delete  insert(x1,union(x2,x4)) = insert(x1,union(x2,x4))

Application of generation on:
    eq(x3,x1) = False, eq(x3,x1) = True :
 1) eq(0,0) = True, True = False ;
 2) eq(s(x1),0) = True, False = False ;
 3) eq(0,s(x1)) = True, False = False ;
 4) eq(s(x2),s(x1)) = True, eq(x2,x1) = False

Delete  eq(s(x1),0) = True, False = False

Delete  eq(0,s(x1)) = True, False = False

Delete  eq(0,0) = True, True = False
 it is subsumed by:eq(x1,x1) = True of R

Simplification of:
    eq(s(x2),s(x1)) = True, eq(x2,x1) = False by R:
    eq(x2,x1) = True, eq(x2,x1) = False

E4 = {eq(x2,x1) = True, eq(x2,x1) = False}
H4 = {eq(x3,x1) = False, eq(x3,x1) = True ;
      insert(x1,insert(x1,x2)) = insert(x1,x2)}

Delete  eq(x2,x1) = True, eq(x2,x1) = False
 it is subsumed by:eq(x3,x1) = False, eq(x3,x1) = True of H4

E5 = {}
H5 = {eq(x3,x1) = False, eq(x3,x1) = True ;

      insert(x1,insert(x1,x2)) = insert(x1,x2)}

The initial conjectures are observationally valid in R
```

In the same way we have proved the following conjectures:

$$insert(x, insert(y,l)) = insert(y, insert(x,l)) \quad and \quad union(l,l') = union(l',l)$$

8 Conclusion

We have presented an automatic procedure for proving observational properties in conditional specifications. The method relies on the construction of a set of *critical contexts* which enables to prove or disprove conjectures. Under reasonable hypotheses, we have shown that the procedure is refutational complete: each non observationally valid conjecture will be detected after a finite time.

A cover context w.r.t. our definition 3 garantees the soundness of our procedure. However, cover contexts computed by our procedure may contain unecessary contexts, as in Example 17 where $union(z_{list}, x)$ is useless for observations. We plan to refine our notion of cover and critical contexts in order to select only the needed contexts.

We also plan to extend the observation technique to terms and formulas.

References

1. B. Bauer and R. Hennicker. Proving the correctness of algebraic implementations by the ISAR system. In *DISCO'93*, volume 722 of *Lecture Notes in Computer Science*, pages 2–16. Springer-Verlag, 1993.
2. G. Bernot, M. Bidoit, and T. Knapik. Behavioural approaches to algebraic specifications: A comparative study. *Acta Informatica*, 31(7):651–671, 1994.
3. N. Berregeb, A. Bouhoula, and M. Rusinowitch. Observational proofs by implicit context induction. Technical Report 3151, INRIA, 1997.
4. M. Bidoit and R. Hennicker. How to prove observational theorems with LP. In U. Martin and J. Wing, editors, *Proc. of First International Workshop on Larch*. Springer-Verlag, 1992.
5. M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165(1):3–55, 1996.
6. A. Bouhoula. Using Induction and Rewriting to Verify and Complete Parameterized Specifications. *Theoretical Computer Science*, 170(1-2):245–276, 1996.
7. A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.
8. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.
9. J. Guttag. *The specification and Application to Programming of Abstract Data Types*. PhD Thesis, University of Toronto, 1975.
10. R. Hennicker. Context induction: a proof principle for behavioural abstractions and algebraic implementations. *Formal Aspects of Computing*, 3(4):326–345, 1991.
11. S. Kaplan and M. Choquer. On the decidability of quasi-reducibility. *Bulletin of European Association for Theoretical Computer Science*, 28:32–34, February 1986.
12. E. Kounalis. Testing for the ground (co-)reducibility property in term-rewriting systems. *Theoretical Computer Science*, 106:87–117, 1992.
13. O. Lysne. Extending Bachmair's method for proof by consistency to the final algebra. *Information Processing Letters*, 51:303–310, 1994.
14. G. Malcolm and J. Goguen. Proving correctness of refinement and implementation. Technical Monograph PRG-114, Oxford University Computing Laboratory, November 1994.
15. P. Padawitz. *Computing in Horn Clause Theories*. Springer-Verlag, 1988.
16. L. Puel. Proofs in the final algebra. IXth Colloquium on Trees in Algebra and Programming. Bordeaux, France, March 1984.
17. D.T. Sanella and A. Tarlecki. Towards formal development of ml programs: foundations and methodology. In J. Diaz and F. Orejas, editors, *TAPSOFT'89*, volume 352 of *Lecture Notes in Computer Science*, pages 375–389. Springer-Verlag, 1989.
18. M. Wirsing. Algebraic specifications. In J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, and D. Perrin, editors, *Handbook of Theoretical Computer Science*, volume B, chapter 13. Elsevier Science Publishers B. V. (North-Holland) and The MIT press, 1990.