Derivation of Static Analysers of Functional Programs from Path Properties of a Natural Semantics

Valérie Gouranton, Daniel Le Métayer

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Derivation of static analysers of functional programs from path properties of a natural semantics

Valérie GOURANTON et Daniel LE MÉTAYER

N° 2607

Juillet 1995

PROGRAMME 2

ISSN 0249-6399

Derivation of static analysers of functional programs from path properties of a natural semantics

Valérie GOURANTON* et Daniel LE MÉTAYER**

Programme 2 — Calcul symbolique, programmation et génie logiciel
Projet Lande

Rapport de recherche n° 2607 — Juillet 1995 — 25 pages

Abstract: We advocate the use of operational semantics as a basis for specifying program analyses for functional languages. We put forward a methodology for defining a static analysis by successive refinements of the natural semantics of the language. We use paths as the abstract representation of proof trees and we provide a language for defining properties in terms of recurrence equations on paths. We show the specification of several standard properties on paths (neededness, absence, uniqueness, ...) and the mechanical derivation of the corresponding analyses.

Key-words: functional languages, operational semantics, neededness analysis, paths analysis, optimising compilers.

(Résumé : tsvp)

*gouranto@irisa.fr
**lemetaye@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Dérivation d'analyseurs de programmes fonctionnels à partir de propriétés de chemin d'une sémantique naturelle

Résumé : Nous préconisons l'utilisation d'une sémantique opérationnelle comme base de spécification d'analyses de programmes pour les langages fonctionnels. Nous mettons en avant une méthodologie pour définir une analyse statique par des raffinements successifs de la sémantique naturelle du langage. Nous utilisons les chemins comme une représentation abstraite d'arbres de preuve et nous fournissons un langage pour définir des propriétés en terme d'équations récursives sur les chemins. Nous montrons la spécification de plusieurs propriétés standards sur les chemins (nécessité, absence, unicité, ...) et la dérivation mécanique des analyses correspondantes.

Mots-clé : langages fonctionnels, sémantique opérationnelle, analyse de nécessité, analyse de chemin, optimisation de compilateurs.

1 Introduction

We first describe the general motivation of the work presented here before summarising the main results developed in the body of the paper.

1.1 Motivation

Because optimising compilers are crucial for the implementation of functional languages, the static analysis of functional programs has received considerable interest during the last decade. In particular a number of sophisticated frameworks and algorithms have been proposed for strictness analysis [3, 12], path analysis [1, 2, 17] and single-threading [9, 16]. The correctness and the accuracy of the analyses have been extensively studied and several techniques have been proposed recently to improve the efficiency of the analysers [10, 15].

Despite the fact that abstract interpretation was originally defined in an operational framework [5], most of the analyses for functional languages are based on abstract interpretations of denotational semantics. While this may be understood as a natural choice in the context of functional languages, this bias has unfortunate consequences. One significant shortcoming of the denotational approach is the fact that the connection with the optimisation which is supposed to be the initial motivation for the analysis may become less obvious. As an illustration, [4] and [8] use continuations to prove the correctness of strictness-based optimisations. This task is far from being trivial. An essentially operational property (neededness) is recast in the denotational framework (strictness) for the sake of the analysis, to be then used in an operational framework (abstract machine or compiler). Of course we can resort to continuations to recover evaluation order in a denotational setting but this may not be the most natural way to proceed. More importantly, this may preclude the application of the techniques developed in the functional community to other families of languages.

We focus on the practical advantages of operational semantics for program analysis in this paper. We put forward a methodology for defining a static analysis by successive refinements of the natural semantics of the language. We illustrate the methodology with the specification of several standard properties on paths (neededness, absence, uniqueness, ...) and the derivation of the corresponding analyses.

1.2 Outline

The framework we put forward involves two main steps for the refinement of the natural semantics of the language:

1. The definition of a function mapping proof trees in the natural semantics into abstract proof trees. Abstract proof trees extract from the sequents the pieces of information which are relevant to the class of analyses under consideration. We choose an abstraction in terms of paths (or sequences of variables) in this paper.

2. The definition of specific properties expressed in terms of recurrence equations on abstract proof trees.

We provide a syntax for proof trees and we define their abstraction in terms of paths. Paths are constructed from basic elements (the empty path ^ and singleton paths $< x >$) and the concatenation operation @. They record information about the application of the rule for variables in the operational semantics. The properties of interest are defined in a language including user-defined basic properties, conjunction and "arrow properties" of functions. As an illustration, we show the definition of the property that a variable x is *needed* in a path:

$$\begin{array}{lcl} \overline{Need}^{\,x}(\hat{}) & = & \textbf{false} \\ \overline{Need}^{\,x}(<x>) & = & \textbf{true} \\ \overline{Need}^{\,x}(<\alpha>) & = & \textbf{false} \\ \overline{Need}^{\,x}(p_1 \mathbin{@} p_2) & = & \overline{Need}^{\,x}(p_1) \vee \overline{Need}^{\,x}(p_2) \end{array}$$

A generic inference system is constructed from the natural semantics and the path abstraction function. For instance, the generic system includes the following rule for the conditional (P is a side condition whose precise definition is not significant at this stage):

$$\textbf{COND} \quad \frac{\Gamma \vdash_g e_1 : \pi_1 \quad \Gamma \vdash_g e_2 : \pi_2 \quad \Gamma \vdash_g e_3 : \pi_3}{\Gamma \vdash_g \textbf{cond}(e_1\ e_2\ e_3) : \delta_i}$$

$$\text{if } P(\pi_1, \pi_2, \pi_3, \delta_i)$$

Specific systems can then be derived automatically from the generic system and the definitions of properties. For instance the following two rules are derived from the above generic rule for $Need^{\,x}$:

$$\textbf{COND}_1^1 \quad \frac{\Gamma \vdash_p e_1 : Need^{\,x}}{\Gamma \vdash_p \textbf{cond}(e_1\ e_2\ e_3) : Need^{\,x}}$$

$$\textbf{COND}_1^2 \quad \frac{\Gamma \vdash_p e_2 : Need^{\,x} \quad \Gamma \vdash_p e_3 : Need^{\,x}}{\Gamma \vdash_p \textbf{cond}(e_1\ e_2\ e_3) : Need^{\,x}}$$

The derived inference systems are associated with efficient inference algorithms exploiting the notion of "lazy type inference" introduced in [10]. The algorithms are expressed as syntax-directed and deterministic inference systems.

Apart from neededness analysis, we consider the following properties in this paper:

1. $Pred^{\,x,y}$ which is satisfied if any occurrence of y in a path is preceded by an occurrence of x. This property can be used to show that a variable has already been evaluated at a particular point in the program; it can be exploited to avoid tests in the implementation of lazy functional languages.

2. $Un^{\,x}$ which holds if x cannot appear more than once in a path. This information can be used to show that a variable can't be shared and so there is no need for updating the heap after its evaluation.

3. *Naft* ^{x,y} which means that there is no occurrence of y after an occurrence of x in a path. This property is useful in the context of *in-place updating* to detect the last occurrence of a variable.

4. *Abs* ^{x} which characterises paths without any occurrence of x. *Abs* ^{x} is used in the definition of the above properties.

For each of these properties, we provide its initial specification in the language of properties and we show the derived inference system.

In section 2 we introduce the natural semantics of the language, the path abstraction function and the instrumented path semantics. Section 3 defines the language of properties on paths and the generic inference system to assign properties to expressions. We establish the correctness of the system with respect to the instrumented semantics. In section 4 we provide a mechanical procedure for the specialisation of the generic inference system with respect to specific path properties. We give the specification of several path properties and we show the corresponding specialisations of the system. Section 5 discusses the lazy inference algorithm associated with the generic system. Section 6 is a review of related work and a section 7 discusses avenues for further research.

2 Natural semantics and path semantics

We consider a non-strict simply typed functional language whose syntax and semantics are defined in Figure 1 and Figure 2. Variables b, e and v range respectively over B, E and V. The environment $\Gamma + [x : (\Gamma', e)]$ is like Γ except that x is attached the value (Γ', e) and $\Gamma[x : (\Gamma', e)]$ denotes an environment associating x with the value (Γ', e). In order to describe the abstraction, we define the syntax of proof trees and paths.

$$
\begin{array}{lcl}
PT & ::= & [A, S] \mid [SR, S, PT] \mid [MR, S, PT_1, PT_2] \\
S & ::= & [Env, E, V] \\
A & ::= & \text{VAL} \mid \lambda \\
SR & ::= & \text{VAR} \mid \text{FIX} \\
MR & ::= & \text{APP} \mid \text{OP} \mid \text{COND}_t \mid \text{COND}_f \\
P & ::= & \hat{} \mid <x> \mid P_1 \,@\, P_2
\end{array}
$$

In the definition of PT (Proof Trees), A, SR and MR represent respectively axioms, single-premise rules and the multiple-premises rules. S is the category of sequents.

DEFINITION **2.1** *We write Prooftree*$(\vdash_k, s, pt)$ *if pt is a proof tree of the sequent s in the inference system* $\vdash_k$.

We can now define the function A_p which abstracts the path information from a proof tree (following our convention, lower-case variables range in the corresponding upper-case domains):

DEFINITION **2.2**

$$A_p \ : \ PT \rightarrow P$$

$$\begin{array}{lcl} A_p\ [a,s] & = & \wedge \\ A_p\ [VAR,[\Gamma,x,v],pt] & = & < x > @(A_p\ pt) \\ A_p\ [FIX,s,pt] & = & A_p\ pt \\ A_p\ [mr,s,pt_1,pt_2] & = & (A_p\ pt_1)@(A_p\ pt_2) \end{array}$$

The instrumented semantics derived from this abstraction is shown in Figure 3. The result of an evaluation in the path semantics is a pair $< value, path >$. The following correspondence property can easily be proven by recurrence on the structure of proof trees:

THEOREM **2.3**

$$\Gamma \vdash_s e \rightarrow v \quad \Leftrightarrow \quad \Gamma \vdash_i e \rightarrow <v, A_p(pt)>$$

where $Prooftree(\vdash_s, [\Gamma, e, v], pt)$

Expressions
$E ::= x \mid B \mid \lambda x.E \mid E_1\ \mathbf{Op}\ E_2 \mid \mathbf{cond}(E_1\ E_2\ E_3) \mid E_1 E_2 \mid \mathbf{fix}\ \lambda f.\lambda x.E$
$B ::= \mathbf{true} \mid \mathbf{false} \mid n$

Environments
$Env ::= \emptyset \mid Env_1 + [x : (Env_2, E)]$

Terminal values
$V ::= B \mid (Env, \lambda x.E) \mid V_1\ \mathbf{Op}\ V_2$

Figure 1: Syntax of the language

3 The generic inference system

As we mentioned in the introduction, we are not interested in path analysis *per se* in this paper. We rather see the path semantics as an intermediate step towards the definition of an analyser. The next step consists in expressing in terms of paths the set of properties of interest. In this section, we present the language of properties and a generic inference system. The next section shows the definition in this language of several specific properties and the specialised inference systems they yield. The syntax of properties is presented in Figure 4. The language looks very much like a type system with conjunction and basic types δ_i, **True** and **False**. Simple properties apply to paths and arrow properties characterise functional behaviours. The basic properties δ_i are defined by the user in terms of their characteristic function $\overline{\delta_i}$ as shown in Figure 4. The characteristic function must be defined for each form of path (empty path $\wedge$, singleton path $< x >$ or concatenation $p_1 @ p_2$). The x_i are the variables which are significant for the defined property (like x in $Need^{\,x}$) and α stands for other variables. The definition of $Need^{\,x}$ in the introduction is an illustration of this syntax.

$$[\text{VAL}]\quad \Gamma \vdash_s b \to b$$

$$[\lambda]\quad \Gamma \vdash_s \lambda x.e \to (\Gamma, \lambda z.e[z/x]) \quad \textbf{where } z \textbf{ is a fresh variable}$$

$$[\text{VAR}]\quad \frac{\Gamma' \vdash_s e \to v}{\Gamma[x : (\Gamma', e)] \vdash_s x \to v} \qquad [\text{OP}]\quad \frac{\Gamma \vdash_s e_1 \to v_1 \quad \Gamma \vdash_s e_2 \to v_2}{\Gamma \vdash_s e_1 \ \mathbf{Op}\ e_2 \to v_1 \ \mathbf{Op}\ v_2}$$

$$[\text{COND}_t]\quad \frac{\Gamma \vdash_s e_1 \to \mathbf{true} \quad \Gamma \vdash_s e_2 \to v}{\Gamma \vdash_s \mathbf{cond}(e_1\ e_2\ e_3) \to v} \qquad [\text{COND}_f]\quad \frac{\Gamma \vdash_s e_1 \to \mathbf{false} \quad \Gamma \vdash_s e_3 \to v}{\Gamma \vdash_s \mathbf{cond}(e_1\ e_2\ e_3) \to v}$$

$$[\text{FIX}]\quad \frac{\Gamma + [f : (\Gamma, \mathbf{fix}\ \lambda f.\lambda x.e)] \vdash_s \lambda x.e \to v}{\Gamma \vdash_s \mathbf{fix}\ \lambda f.\lambda x.e \to v}$$

$$[\text{APP}]\quad \frac{\Gamma \vdash_s e_1 \to (\Gamma', \lambda x.e_1') \quad \Gamma' + [x : (\Gamma, e_2)] \vdash_s e_1' \to v}{\Gamma \vdash_s e_1 e_2 \to v}$$

Figure 2: Natural Semantics of the language

$$[\text{VAL}]\quad \Gamma \vdash_i b \to <b, \wedge>$$

$$[\lambda]\quad \Gamma \vdash_i \lambda x.e \to <(\Gamma, \lambda z.e[z/x]), \wedge> \quad \textbf{where } z \textbf{ is a fresh variable}$$

$$[\text{VAR}]\quad \frac{\Gamma' \vdash_i e \to <v, p>}{\Gamma[x : (\Gamma', e)] \vdash_i x \to <v, <x> @ p>}$$

$$[\text{OP}]\quad \frac{\Gamma \vdash_i e_1 \to <v_1, p_1> \quad \Gamma \vdash_i e_2 \to <v_2, p_2>}{\Gamma \vdash_i e_1\ \mathbf{Op}\ e_2 \to <v_1\ \mathbf{Op}\ v_2, p_1 @ p_2>}$$

$$[\text{COND}_t]\quad \frac{\Gamma \vdash_i e_1 \to <\mathbf{true}, p> \quad \Gamma \vdash_i e_2 \to <v, q>}{\Gamma \vdash_i \mathbf{cond}(e_1\ e_2\ e_3) \to <v, p @ q>}$$

$$[\text{COND}_f]\quad \frac{\Gamma \vdash_i e_1 \to <\mathbf{false}, p> \quad \Gamma \vdash_i e_3 \to <v, q>}{\Gamma \vdash_i \mathbf{cond}(e_1\ e_2\ e_3) \to <v, p @ q>}$$

$$[\text{FIX}]\quad \frac{\Gamma + [f : (\Gamma, \mathbf{fix}\ \lambda f.\lambda x.e)] \vdash_i \lambda x.e \to <v, p>}{\Gamma \vdash_i \mathbf{fix}\ \lambda f.\lambda x.e \to <v, p>}$$

$$[\text{APP}]\quad \frac{\Gamma \vdash_i e_1 \to <(\Gamma', \lambda x.e_1'), p> \quad \Gamma' + [x : (\Gamma, e_2)] \vdash_i e_1' \to <v, q>}{\Gamma \vdash_i e_1 e_2 \to <v, p @ q>}$$

Figure 3: Instrumented path semantics

Note that **true** and **false** are boolean values whereas **True** and **False** are simple properties in the language. Symbols $\wedge$ and $\vee$ are overloaded but no confusion should arise from this abuse of notation.

The semantics of simple properties is defined by the function S which returns the set of paths satisfying a given property:

DEFINITION **3.1**

$$S : \pi \rightarrow \mathbf{P}\ (P)$$

$$\begin{array}{lcl} S(\delta_i) & = & \{p \mid \overline{\delta_i}(p) = \mathbf{true}\} \\ S(\mathbf{True}) & = & P \\ S(\mathbf{False}) & = & \emptyset \\ S(\pi_1 \wedge \pi_2) & = & S(\pi_1) \cap S(\pi_2) \end{array}$$

The inference system for properties is defined in Figure 6. It makes use of a partial ordering on properties defined in Figure 5.

Before presenting the inference system for properties, we introduce a convenient notation:

DEFINITION **3.2**

Let the disjunctive normal form of $\overline{\delta_i}(p_1 \mathbin{@} p_2)$

$$\bigvee_{j=1}^{n} \Big(\bigwedge_{k=1}^{k_j} \overline{\delta}_{a_k}(p_1) \wedge \bigwedge_{l=1}^{l_j} \overline{\delta}_{b_l}(p_2) \Big)$$

We note

$$A(\delta_i) = \{(\pi_1^j, \pi_2^j) \mid \pi_1^j = \bigwedge_{k=1}^{k_j} \delta_{a_k} \quad and \quad \pi_2^j = \bigwedge_{l=1}^{l_j} \delta_{b_l}\}$$

As the following lemma shows, $A(\delta_i)$ provides sufficient conditions on p_1 and p_2 for $p_1 \mathbin{@} p_2$ to satisfy δ_i.

LEMMA **3.3**

$$(\pi_1, \pi_2) \in A(\delta_i) \quad \text{and} \quad p_1 \in S(\pi_1) \quad \text{and} \quad p_2 \in S(\pi_2)$$

$$\Longrightarrow p_1 \mathbin{@} p_2 \in S(\delta_i)$$

The rules in Figure 6 can be understood by comparison with their counterpart in Figure 3. The partial ordering is used in some rules to extract θ_i from $\theta_1 \wedge \ldots \wedge \theta_n$. Functional expressions may satisfy simple properties on paths (like $Need^{\,x}$) accounting for the evaluation path for their reduction to weak head normal form (lambda abstraction) and arrow properties (like $Need^{\,x} \rightarrow Need^{\,y}$) accounting for their functional behaviour. This explains why $\vdash_g$

Language of properties
$\pi ::= \delta_i \mid \mathbf{True} \mid \mathbf{False} \mid \pi_1 \wedge \pi_2$ (simple)
$\theta ::= \pi \mid \pi \wedge \phi \mid \phi$ (general)
$\phi ::= \theta_1 \rightarrow \theta_2 \mid \phi_1 \wedge \phi_2$ (functional)

Environments
$Env_p ::= \emptyset \mid Env_p + [x : \theta]$

Basic properties
$\overline{\delta_i}(\wedge) = c_0$
$\overline{\delta_i}(<x_1>) = c_1$
...
$\overline{\delta_i}(<x_n>) = c_n$
$\overline{\delta_i}(<\alpha>) = c_{n+1}$
$\overline{\delta_i}(p_1 \ @ \ p_2) = exp$
$c \quad ::= \mathbf{true} \mid \mathbf{false}$

$exp ::= c \mid \overline{\delta_j}(p_1) \mid \overline{\delta_j}(p_2) \mid exp_1 \wedge \exp_2 \mid exp_1 \vee \exp_2$

Figure 4: Syntax of properties

$\theta \leq \theta$ $\quad \theta \leq \mathbf{True}$ $\quad \mathbf{False} \leq \theta$ $\quad \theta \leq \theta \wedge \theta$ $\quad \theta_1 \wedge \theta_2 \leq \theta_1$ $\quad \theta_1 \wedge \theta_2 \leq \theta_2$

$$\frac{\theta'_1 \leq \theta_1 \quad \theta_2 \leq \theta'_2}{\theta_1 \rightarrow \theta_2 \leq \theta'_1 \rightarrow \theta'_2} \qquad \frac{\theta_1 \leq \theta'_1 \quad \theta_2 \leq \theta'_2}{\theta_1 \wedge \theta_2 \leq \theta'_1 \wedge \theta'_2} \qquad (\theta_1 \rightarrow \theta_2) \wedge (\theta_1 \rightarrow \theta_3) \leq \theta_1 \rightarrow (\theta_2 \wedge \theta_3)$$

Figure 5: Partial ordering on properties

includes two rules for most of the syntactic constructs of the language (see APP_1 and APP_2 for instance): the first rule is used to prove simple properties and the second one to prove arrow properties.

In order to relate the inference system $\vdash_g$ with the operational semantics of the language $\vdash_s$, we need to introduce a consistency relation between environments.

DEFINITION **3.4 (Consistency)**

$$\begin{aligned}
&Cons(<v,p>,\theta) = Cons_\pi(<v,p>,\theta) \text{ and } Cons_\phi(<v,p>,\theta)\\
&Cons_\pi(<v,p>,\theta) = (\theta \leq \pi \Rightarrow p \in S(\pi))\\
&Cons_\phi(<(\Gamma,\lambda x.e),p>,\theta) =\\
&\qquad (\theta \leq \theta_1 \rightarrow \theta_2 \text{ and } Corr_e((\Gamma_1,e_1),\theta_1) \Rightarrow\\
&\qquad\qquad Corr_e((\Gamma + [x : (\Gamma_1,e_1)],e),\theta_2))
\end{aligned}$$

DEFINITION **3.5 (Correlation)**

$$\begin{aligned}
&Corr_e((\Gamma,e),\theta) = (\Gamma \vdash_i e \rightarrow <v,p> \Longrightarrow Cons(<v,p>,\theta))\\
&Corr_\gamma(\Gamma_i,\Gamma_g) = \forall x.\ Corr_e(\Gamma_i(x),\Gamma_g(x))
\end{aligned}$$

Cons is expressed in terms of $Cons_\pi$ and $Cons_\phi$ which handle of respectively the simple properties part π and the functional properties part ϕ of a conjunction $\pi \wedge \phi$. *Cons* applies to pairs $< v, p >$ which are terminal values in the instrumented semantics. $Corr_e$ and $Corr_\gamma$ define correlation relations between closures and properties. The mutually recursive

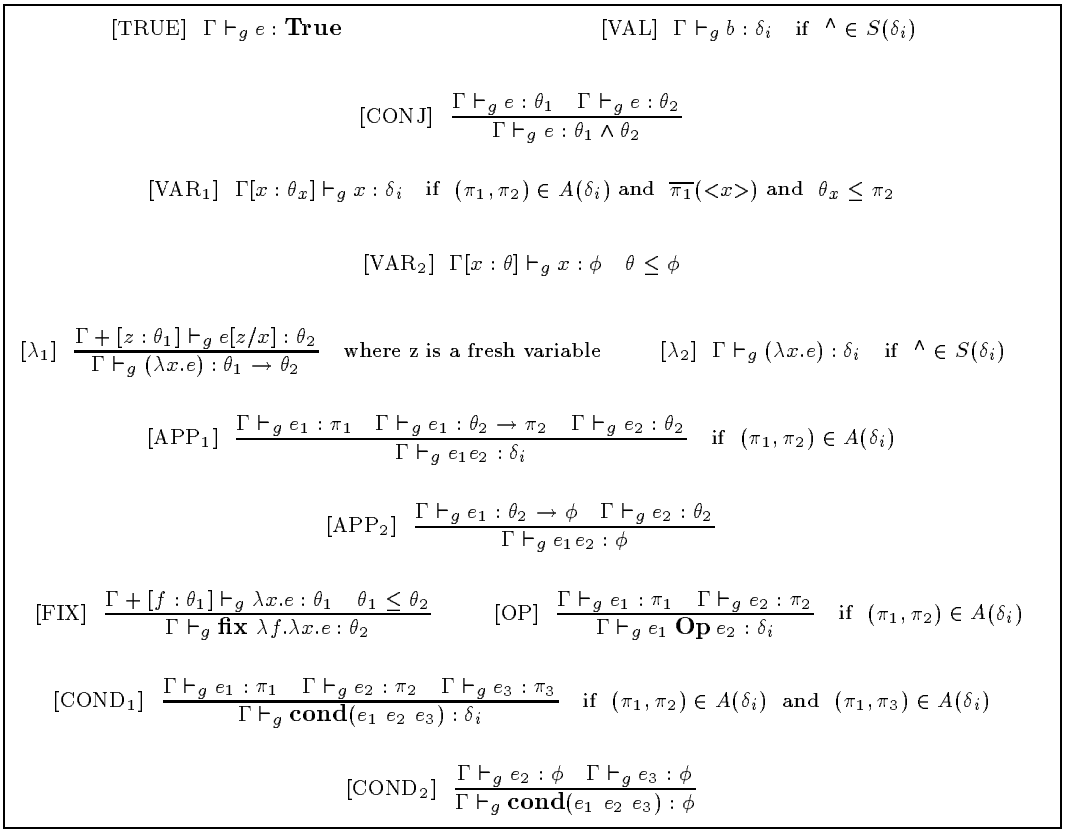

Figure 6: General inference system

definitions of $Cons$ and $Corr_e$ are well founded because we consider a typed language. We can now establish the correctness of the inference system as follows.

THEOREM **3.6**

$$\forall e.\ \forall \Gamma_i.\ \forall \Gamma_g. \quad Corr_\gamma(\Gamma_i, \Gamma_g) \text{ and } \Gamma_g \vdash_g e : \theta \ \Rightarrow Corr_e((\Gamma_i, e), \theta)$$

The proof of this theorem can be made by recurrence on the proof tree of $\Gamma_g \vdash_g e : \theta$. We just study the cases for [VAR$_1$] and [APP$_1$] here:

Case [VAR$_1$]: We have:

$$\Gamma_g[x : \theta_x] \vdash_g x : \delta_i \quad \text{if} \quad (\pi_1, \pi_2) \in A(\delta_i) \text{ and } \overline{\pi_1}(<x>) \text{ and } \theta_x \leq \pi_2$$

$$\frac{\Gamma' \vdash_i e \rightarrow <v, p>}{\Gamma_i[x : (\Gamma', e)] \vdash_i x \rightarrow <v, <x> @ p>}$$

We have

$$Corr_\gamma(\Gamma_i[x : (\Gamma', e)], \Gamma_g[x : \theta_x])$$

and

$$\Gamma' \vdash_i e \rightarrow <v, p>$$

which entails

$$Cons(<v, p>, \theta_x)$$

from the definition of $Corr_\gamma$. Thus $p \in S(\pi_2)$ from the definition of $Cons$.
But $<x> \in S(\pi_1)$, so, from lemma 3.3 $(<x> @ p) \in S(\delta_i)$

which entails

$$Cons(<v, <x> @ p>, \delta_i)$$

and

$$Corr_e((\Gamma_i[x : (\Gamma', e)], x), \delta_i)$$

Case [APP$_1$]: We have

$$\frac{\Gamma_g \vdash_g e_1 : \pi_1 \quad \Gamma_g \vdash_g e_1 : \theta_2 \rightarrow \pi_2 \quad \Gamma_g \vdash_g e_2 : \theta_2}{\Gamma_g \vdash_g e_1 e_2 : \delta_i}$$

$$\text{if } (\pi_1, \pi_2) \in A(\delta_i)$$

$$\frac{\Gamma_i \vdash_i e_1 \rightarrow <(\Gamma', \lambda x.e_1'), p_1> \quad \Gamma' + [x : (\Gamma_i, e_2)] \vdash_i e_1' \rightarrow <v, p_2>}{\Gamma_i \vdash_i e_1 e_2 \rightarrow <v, p_1 @ p_2>}$$

Three properties follow from the induction hypotheses:
(1) $Cons(<(\Gamma', \lambda x.e'_1), p_1>, \pi_1)$
(2) $Cons(<(\Gamma', \lambda x.e'_1), p_1>, \theta_2 \rightarrow \pi_2)$
(3) $Corr_e((\Gamma_i, e_2), \theta_2)$
(1) entails $p_1 \in \pi_1$ and (2) and (3) entail

$$Corr_e((\Gamma' + [x : (\Gamma_i, e_2)], e'_1), \pi_2).$$

So $p_2 \in \pi_2$ and, from lemma 3.3, we have

$$(p_1 \mathbin{@} p_2) \in S(\delta_i) \text{ and } Corr_e((\Gamma_i, e_1 e_2), \delta_i).$$

The definition of $Corr_e$ involves a condition $\Gamma \vdash_i e \rightarrow <v, p>$ which means that the correctness theorem does not provide any information for expressions which are not well-defined in the original semantics (non-terminating expressions for example). The following definition allows us to characterise properties which cannot be satisfied and the subsequent corollary (which follows from the correctness theorem) shows that our language of properties can also be used to prove that an expression is not well-defined.

DEFINITION **3.7**

$$\begin{aligned}
&Empty(\pi) = (S(\pi) = \emptyset)\\
&Empty(\pi \wedge \phi) = (Empty(\pi) \vee Empty(\phi))\\
&Empty(\phi_1 \wedge \phi_2) = (Empty(\phi_1) \vee Empty(\phi_2))\\
&Empty(\theta_1 \rightarrow \theta_2) = (Empty(\theta_2))
\end{aligned}$$

COROLLARY **3.8**

$$\forall e.\ \forall \Gamma_i.\ \forall \Gamma_g.\ (Corr_\gamma(\Gamma_i, \Gamma_g) \text{ and } \Gamma_g \vdash_g e : \theta \text{ and } Empty(\theta)$$

$$\Rightarrow \nexists\ v.\ \Gamma_i \vdash_s e \rightarrow v \qquad \textit{(e is not well-defined)}$$

4 Derivation of specific inference systems from specifications of properties

In this section we show how specialised inference systems can be derived from the general inference system is defined in Figure 6. First we present a mechanical procedure which returns a specialised inference system is_p from the general inference system is_g and a specific property $\overline{p}$. Then we provide the definition of several standard properties (using the syntax defined in Figure 4) and we show the derivation of their associated inference systems.

4.1 The specialisation procedure

First we define the syntax of inference rules as follows:

$$\begin{array}{lcl} IR & ::= & (R, C) \\ R & ::= & [A, S] \mid [R_1, S, S_1] \mid [R_2, S, S_1, S_2] \mid [R_3, S, S_1, S_2, S_3] \\ S & ::= & [Env, E, \theta] \mid \theta_1 \leq \theta_2 \\ C & ::= & \text{Condition} \mid \emptyset \end{array}$$

IR represents an inference rule: it is made of a rule and a condition (which may be empty). In the definition of R (Rules), A is the name of an axiom and R_1, R_2 and R_3 are names of rules with respectively one, two and three premises. S is the category of sequents and θ is a general property (see Figure 4). An inference system *is* is defined as a set of inference rules (see Figure 7).

The specialisation procedure *Spec* is defined in Figure 7. It takes as argument a property $\overline{p}$ (defined using the syntax of Figure 4). It returns a specialised inference system which is computed from the general inference system is_g using the production rules defined by the relation $\rightsquigarrow$. Note that each rule of the general system can produce several rules of the specialised system (or no rule at all) according to $\rightsquigarrow$.

4.2 Derivation of specific inference systems

We illustrate the specialisation procedure by showing the definition of several standard properties in our framework and deriving the corresponding instantiations of is_g using *Spec*. Figure 8 contains the definitions of the properties $Need^{\,x}$, $Abs^{\,x}$, $Pred^{\,x,y}$, $Un^{\,x}$ and $Naft^{\,x,y}$ which were introduced in section 1.2. These definitions should be obvious. Let us now focus on the derivation of the specialised inference systems. We only consider the systems for $Need^{\,x}$ and $Pred^{\,x,y}$ in this section. The interested reader can find the systems corresponding to $Abs^{\,x}$, $Un^{\,x}$ and $Naft^{\,x,y}$ in the appendix.

Some of the side conditions of the inference rules of is_g are empty. These rules appear in every specialised system (see the first rule in the definition of $\rightsquigarrow$). They are gathered in Figure 10. The specific rules for $Need^{\,x}$ and $Pred^{\,x,y}$ are presented in Figure 11 and Figure 12. Let us consider for instance the rules corresponding to VAL and VAR_1 and show how they result from the application of *Spec*.

Case VAL: We have:

$$(\ [\text{VAL}, [\Gamma,\ b,\ \delta_i]\ ],\ \hat{}\ \in S(\delta_i)\ ),\ \overline{p}\ \rightsquigarrow (\ [\text{VAL}, [\Gamma,\ b,\ p]\ ], \emptyset\ )\quad \text{if}\ \ \overline{p}(\hat{})$$

For $Need^{\,x}$, there is no instantiation of the rule VAL in Figure 11 because, from the definition of the property $Need^{\,x}$, we have $\overline{Need}^{\,x}(\hat{}) = \textbf{false}$. For $Pred^{\,x,y}$, we have $\overline{Pred}^{\,x,y}(\hat{}) = \textbf{true}$, and:

$$(\ [\text{VAL}, [\Gamma,\ b,\ \delta_i]\ ],\ \hat{}\ \in S(\delta_i)\ ),\ \overline{Pred}^{\,x,y} \rightsquigarrow (\ [\text{VAL}, [\Gamma,\ b,\ Pred^{\,x,y}]\ ], \emptyset\ )$$

Case VAR_1: We have:

$IS = \mathcal{P}(IR)$

is_g = the general inference system defined in Figure 6

Spec : $\overline{P} \longrightarrow IS$

Spec $\overline{p} = \{ir_p \mid ir_g \in is_g \text{ and } ir_g, \overline{p} \rightsquigarrow ir_p\}$

$(R, \emptyset),\ \overline{p} \rightsquigarrow (R, \emptyset)$

$(\ [\text{VAL}, [\Gamma,\ b,\ \delta_i]\ ],\ \hat{}\ \in S(\delta_i)\ ),\ \overline{p} \rightsquigarrow (\ [\text{VAL}, [\Gamma,\ b,\ p]\ ], \emptyset\ )$ if $\overline{p}(\hat{})$

$(\ [\text{VAR}_1, [\Gamma[x : \theta_x],\ x,\ \delta_i]\ ],\ (\pi_1, \pi_2) \in A(\delta_i)$ and $\overline{\pi_1}(<x>)$ and $\theta_x \leq \pi_2\ ),\ \overline{p} \rightsquigarrow$
$(\ [\text{VAR}_1^j, [\Gamma[x : \theta_x],\ x,\ p]\ ],\ \theta_x \leq \pi_2^j\ )$ if $(\pi_1^j, \pi_2^j) \in A(p)$ and $\overline{\pi_1}^j(<x>)$

$(\ [\lambda_2,\quad [\Gamma,\ (\lambda x.e),\ \delta_i]\ ],\ \hat{}\ \in S(\delta_i)\ ),\ \overline{p} \rightsquigarrow (\ [\lambda_2,\quad [\Gamma,\ (\lambda x.e),\ p]\ ], \emptyset\ )$ if $\overline{p}(\hat{})$

$(\ [\text{APP}_1, [\Gamma,\ e_1e_2,\ \delta_i]\ , [\Gamma,\ e_1,\ \pi_1]\ , [\Gamma,\ e_1,\ \theta_2 \rightarrow \pi_2]\ , [\Gamma,\ e_2,\ \theta_2]\ ],\ (\pi_1, \pi_2) \in A(\delta_i)\ ),\ \overline{p}$
$\rightsquigarrow$
$(\ [\text{APP}_1^j, [\Gamma,\ e_1e_2,\ p]\ , [\Gamma,\ e_1,\ \pi_1^j]\ ,\ [\Gamma,\ e_1,\ \theta_2 \rightarrow \pi_2^j]\ , [\Gamma,\ e_2,\ \theta_2]\ ],\ \emptyset\ )$ if $(\pi_1^j, \pi_2^j) \in A(p)$

$(\ [\text{OP},\ [\Gamma,\ e_1\ \mathbf{Op}\ e_2,\ \delta_i]\ , [\Gamma,\ e_1,\ \pi_1]\ , [\Gamma,\ e_2,\ \pi_2]\ ],\ (\pi_1, \pi_2) \in A(\delta_i)\ ),\ \overline{p} \rightsquigarrow$
$(\ [\text{OP}^j, [\Gamma,\ e_1\ \mathbf{Op}\ e_2,\ p]\ , [\Gamma,\ e_1,\ \pi_1^j]\ , [\Gamma,\ e_2,\ \pi_2^j]\ ],\ \emptyset\ )$ if $(\pi_1^j, \pi_2^j) \in A(p)$

$(\ [\text{COND}_1, [\Gamma,\ \mathbf{cond}(e_1\ e_2\ e_3),\ \delta_i]\ , [\Gamma,\ e_1,\ \pi_1]\ , [\Gamma,\ e_2,\ \pi_2]\ , [\Gamma,\ e_3,\ \pi_3]\ ],$
$(\pi_1, \pi_2) \in A(\delta_i)$ and $(\pi_1, \pi_3) \in A(\delta_i)\ ),\ \overline{p}$
$\rightsquigarrow$
$(\ [\text{COND}_1^j, [\Gamma,\ \mathbf{cond}(e_1\ e_2\ e_3),\ p]\ , [\Gamma,\ e_1,\ \pi_1^j]\ , [\Gamma,\ e_2,\ \pi_2^j]\ , [\Gamma,\ e_3,\ \pi_3^j]\ ],\ \emptyset\ )$
if $(\pi_1^j, \pi_2^j) \in A(p)$ and $(\pi_1^j, \pi_3^j) \in A(p)$

Figure 7: Specialisation procedure

$$(\ [\mathrm{VAR}_1, [\Gamma[x : \theta_x], \ x, \ \delta_i] \ ], \ (\pi_1, \pi_2) \in A(\delta_i) \text{ and } \overline{\pi_1}(<x>) \text{ and } \theta_x \leq \pi_2 \ ), \ \overline{p} \ \leadsto$$
$$(\ [\mathrm{VAR}_1^j, [\Gamma[x : \theta_x], \ x, \ p] \ ], \ \ \theta_x \leq \pi_2^j \ ) \ \text{ if } (\pi_1^j, \pi_2^j) \in A(p) \text{ and } \overline{\pi_1}^j(<x>)$$

$A(\mathit{Need}^{\,x})$ contains two elements $(\mathit{Need}^{\,x}, \mathbf{True})$ and $(\mathbf{True}, \mathit{Need}^{\,x})$, so we have two cases:

1. $(\mathit{Need}^{\,x}, \mathbf{True}) \in A(\mathit{Need}^{\,x})$ and $\overline{\mathit{Need}}^{\,x}(<x>) \ = \ \mathbf{true}$, and we have the production:

   $$(\ [\mathrm{VAR}_1, [\Gamma[x : \theta_x], \ x, \ \delta_i] \ ], \ (\pi_1, \pi_2) \in A(\delta_i) \text{ and } \overline{\pi_1}(<x>) \text{ and } \theta_x \leq \pi_2 \ ), \ \overline{\mathit{Need}}^{\,x} \ \leadsto$$
   $$(\ [\mathrm{VAR}_1^1, [\Gamma[x : \theta_x], \ x, \ \mathit{Need}^{\,x}] \ ], \ \emptyset)$$

   The empty condition is generated because $\theta_x \leq \mathbf{True}$ is an axiom of the partial ordering on properties in Figure 5. Furthermore there is no condition on θ_x, which justifies the simplification in Figure 11.

2. $(\mathbf{True}, \mathit{Need}^{\,x}) \in A(\mathit{Need}^{\,x})$ and $\overline{\mathbf{True}}^{\,x}(<\alpha>) \ = \ \mathbf{true}$, and we have the production:

   $$(\ [\mathrm{VAR}_1, [\Gamma[\alpha : \theta_\alpha], \ \alpha, \ \delta_i] \ ], \ (\pi_1, \pi_2) \in A(\delta_i) \text{ and } \overline{\pi_1}(<\alpha>) \text{ and } \theta_\alpha \leq \pi_2 \ ), \ \overline{\mathit{Need}}^{\,x} \ \leadsto$$
   $$(\ [\mathrm{VAR}_1^2, [\Gamma[\alpha : \theta_\alpha], \ \alpha, \ \mathit{Need}^{\,x}] \ ], \ \theta_\alpha \leq \mathit{Need}^{\,x} \ )$$

   The treatment of this case for $\mathit{Pred}^{\,x,y}$ is similar.

To conclude this section, Figure 13 introduces two small examples and shows the kind of properties that can be proven in the specific systems.

The notation $\overline{x}$ stands for $\bigwedge\{\delta_i \in \mathrm{Prop} \mid \overline{\delta_i}(< x >)\}$. It is the conjunction of all the properties that the singleton path $< x >$ satisfies. *Prop* is the finite set of properties under consideration. If we consider for instance the set of variables $\{x, y, z\}$ and the corresponding *Need* and *Pred* properties, we have:

$$\overline{x} = \mathit{Need}^{\,x} \wedge \mathit{Pred}^{\,x,y} \wedge \mathit{Pred}^{\,x,z} \wedge \mathit{Pred}^{\,y,z} \wedge \mathit{Pred}^{\,z,y}$$

This notation is convenient and it represents a more informative property than the very weak **True**.

We should stress the fact that the variable names used in the definition of functions are entirely irrelevant because variables are systematically renamed by α conversion in the semantics of the language. We have shown the type $f : \ \mathit{Need}^{\,x} \rightarrow \mathbf{True} \rightarrow \mathit{Need}^{\,x}$ for the sake of clarity but we could have chosen as well $f : \ \mathit{Need}^{\,y} \rightarrow \mathbf{True} \rightarrow \mathit{Need}^{\,y}$. The function f' is introduced to show that the first occurrence of x in the definition of f represents the first access to x in any path. To do so, we need to distinguish the first occurrence which is done by adding one argument to the function as shown in Figure 13 (the same technique is used in [2]). Similar properties concerning the last access to a variable or the unicity of accesses can be derived using the systems described in the appendix.

$$
\begin{array}{lcl}
\overline{Need}^{\,x}(\hat{}) & = & \mathbf{false} \\
\overline{Need}^{\,x}(<x>) & = & \mathbf{true} \\
\overline{Need}^{\,x}(<\alpha>) & = & \mathbf{false} \\
\overline{Need}^{\,x}(p_1 \,@\, p_2) & = & \overline{Need}^{\,x}(p_1) \vee \overline{Need}^{\,x}(p_2) \\
 & & \\
\overline{Abs}^{\,x}(\hat{}) & = & \mathbf{true} \\
\overline{Abs}^{\,x}(<x>) & = & \mathbf{false} \\
\overline{Abs}^{\,x}(<\alpha>) & = & \mathbf{true} \\
\overline{Abs}^{\,x}(p_1 \,@\, p_2) & = & \overline{Abs}^{\,x}(p_1) \wedge \overline{Abs}^{\,x}(p_2) \\
 & & \\
\overline{Pred}^{\,x,y}(\hat{}) & = & \mathbf{true} \\
\overline{Pred}^{\,x,y}(<x>) & = & \mathbf{true} \\
\overline{Pred}^{\,x,y}(<y>) & = & \mathbf{false} \\
\overline{Pred}^{\,x,y}(<\alpha>) & = & \mathbf{true} \\
\overline{Pred}^{\,x,y}(p_1 \,@\, p_2) & = & \overline{Pred}^{\,x,y}(p_1) \wedge (\overline{Need}^{\,x}(p_1) \vee \overline{Pred}^{\,x,y}(p_2)) \\
 & & \\
\overline{Un}^{\,x}(\hat{}) & = & \mathbf{true} \\
\overline{Un}^{\,x}(<\alpha>) & = & \mathbf{true} \\
\overline{Un}^{\,x}(p_1 \,@\, p_2) & = & (\overline{Un}^{\,x}(p_1) \wedge \overline{Abs}^{\,x}(p_2)) \vee (\overline{Abs}^{\,x}(p_1) \wedge \overline{Un}^{\,x}(p_2)) \\
 & & \\
\overline{Naft}^{\,x,y}(\hat{}) & = & \mathbf{true} \\
\overline{Naft}^{\,x,y}(<\alpha>) & = & \mathbf{true} \\
\overline{Naft}^{\,x,y}(p_1 \,@\, p_2) & = & (\overline{Abs}^{\,x}(p_1) \wedge \overline{Naft}^{\,x,y}(p_2)) \vee (\overline{Abs}^{\,y}(p_2) \wedge \overline{Naft}^{\,x,y}(p_1))
\end{array}
$$

Figure 8: Definition of specific properties

$$A(Need^{\,x}) = \{(Need^{\,x}, \mathbf{True}), (\mathbf{True}, Need^{\,x})\}$$

$$A(Abs^{\,x}) = \{(Abs^{\,x}, Abs^{\,x})\}$$

$$A(Pred^{\,x,y}) = \{(Pred^{\,x,y} \wedge Need^{\,x}, \mathbf{True}), (Pred^{\,x,y}, Pred^{\,x,y})\}$$

$$A(Un^{\,x}) = \{(Un^{\,x}, Abs^{\,x}), (Abs^{\,x}, Un^{\,x})\}$$

$$A(Naft^{\,x,y}) = \{(Abs^{\,x}, Naft^{\,x,y}), (Naft^{\,x,y}, Abs^{\,y})\}$$

Figure 9: Values of A for the properties considered

$$[\text{TRUE}]\ \ \Gamma \vdash_p e : \mathbf{True} \qquad [\text{CONJ}]\ \ \frac{\Gamma \vdash_p e : \theta_1 \quad \Gamma \vdash_p e : \theta_2}{\Gamma \vdash_p e : \theta_1 \wedge \theta_2}$$

$$[\text{VAR}_2]\ \ \Gamma[x : \theta] \vdash_p x : \phi \quad \theta \leq \phi \qquad [\lambda_1]\ \ \frac{\Gamma + [z : \theta_1] \vdash_p e[z/x] : \theta_2}{\Gamma \vdash_p (\lambda x.e) : \theta_1 \rightarrow \theta_2}\ \ \text{where z is a fresh variable}$$

$$[\text{APP}_2]\ \ \frac{\Gamma \vdash_p e_1 : \theta_2 \rightarrow \phi \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : \phi} \qquad [\text{FIX}]\ \ \frac{\Gamma + [f : \theta_1] \vdash_p \lambda x.e : \theta_1 \quad \theta_1 \leq \theta_2}{\Gamma \vdash_p \mathbf{fix}\ \lambda f.\lambda x.e : \theta_2}$$

$$[\text{COND}_2]\ \ \frac{\Gamma \vdash_p e_2 : \phi \quad \Gamma \vdash_p e_3 : \phi}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : \phi}$$

Figure 10: General rules

$$[\text{VAR}_1^1]\ \ \Gamma \vdash_p x : \mathit{Need}^{\,x}$$

$$[\text{VAR}_1^2]\ \ \Gamma[\alpha : \theta_\alpha] \vdash_p \alpha : \mathit{Need}^{\,x} \quad \text{if}\ \ \theta_\alpha \leq \mathit{Need}^{\,x} \qquad \alpha \neq x$$

$$[\text{APP}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : \theta_2 \rightarrow \mathit{Need}^{\,x} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : \mathit{Need}^{\,x}} \qquad [\text{APP}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Need}^{\,x}}{\Gamma \vdash_p e_1 e_2 : \mathit{Need}^{\,x}}$$

$$[\text{OP}_1]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Need}^{\,x}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : \mathit{Need}^{\,x}} \qquad [\text{OP}_2]\ \ \frac{\Gamma \vdash_p e_2 : \mathit{Need}^{\,x}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : \mathit{Need}^{\,x}}$$

$$[\text{COND}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Need}^{\,x}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : \mathit{Need}^{\,x}} \qquad [\text{COND}_1^2]\ \ \frac{\Gamma \vdash_p e_2 : \mathit{Need}^{\,x} \quad \Gamma \vdash_p e_3 : \mathit{Need}^{\,x}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : \mathit{Need}^{\,x}}$$

Figure 11: $\mathit{Need}^{\,x}$ specific rules

$$[\text{VAL}]\ \ \Gamma \vdash_p b : Pred^{\,x,y} \qquad\qquad [\lambda_2]\ \ \Gamma \vdash_p \lambda z.e : Pred^{\,x,y}$$

$$[\text{VAR}_1^1]\ \ \Gamma \vdash_p x : Pred^{\,x,y}$$

$$[\text{VAR}_1^2]\ \ \Gamma[\alpha : \theta_\alpha] \vdash_p \alpha : Pred^{\,x,y} \quad \text{if} \quad \theta_\alpha \leq Pred^{\,x,y} \quad \alpha \neq x \quad \alpha \neq y$$

$$[\text{APP}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : Pred^{\,x,y} \quad \Gamma \vdash_p e_1 : Need^{\,x}}{\Gamma \vdash_p e_1 e_2 : Pred^{\,x,y}}$$

$$[\text{APP}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : Pred^{\,x,y} \quad \Gamma \vdash_p e_1 : \theta_2 \rightarrow \text{Pred}^{\,x,y} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : Pred^{\,x,y}}$$

$$[\text{OP}_1]\ \ \frac{\Gamma \vdash_p e_1 : Pred^{\,x,y} \quad \Gamma \vdash_p e_1 : Need^{\,x}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : Pred^{\,x,y}} \qquad [\text{OP}_2]\ \ \frac{\Gamma \vdash_p e_1 : Pred^{\,x,y} \quad \Gamma \vdash_p e_2 : Pred^{\,x,y}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : Pred^{\,x,y}}$$

$$[\text{COND}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : Pred^{\,x,y} \quad \Gamma \vdash_p e_1 : \text{Need}^{\,x}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : Pred^{\,x,y}}$$

$$[\text{COND}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : Pred^{\,x,y} \quad \Gamma \vdash_p e_2 : Pred^{\,x,y} \quad \Gamma \vdash_p e_3 : Pred^{\,x,y}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : Pred^{\,x,y}}$$

Figure 12: $Pred^{\,x,y}$ specific rules

$$f\ x\ y = \mathbf{cond}((x = 0)\ y\ (f(x-1)(x*y)))$$
$$g\ x\ y\ z = \mathbf{cond}((z = 0)\ (x+y)\ (g\ y\ x\ (z-1)))$$
$$f'\ x_1\ x_2\ y = \mathbf{cond}((x_1 = 0)\ y\ (f(x_2-1)(x_2*y)))$$

$$f : Need^{\,x} \rightarrow \mathbf{True} \rightarrow Need^{\,x}$$
$$f : \mathbf{True} \rightarrow Need^{\,y} \rightarrow Need^{\,y}$$
$$f : \overline{x} \rightarrow \overline{y} \rightarrow Pred^{\,x,y}$$

$$g : Need^{\,x} \rightarrow \mathbf{True} \rightarrow \mathbf{True} \rightarrow Need^{\,x}$$
$$g : \mathbf{True} \rightarrow Need^{\,x} \rightarrow \mathbf{True} \rightarrow Need^{\,x}$$
$$g : \mathbf{True} \rightarrow \mathbf{True} \rightarrow Need^{\,x} \rightarrow Need^{\,x}$$
$$g : \overline{x} \rightarrow \overline{y} \rightarrow \overline{z} \rightarrow Pred^{\,z,x}$$

$$f' : \overline{x_1} \rightarrow \overline{x_2} \rightarrow \overline{y} \rightarrow Pred^{\,x_1,x_2}$$

Figure 13: Examples

5 Inference algorithms

In order to derive inference algorithms from the inference systems presented in the previous sections, we use the notion of *lazy type inference*. Lazy type inference was first introduced in [10] and it has proved to be an appropriate basis for the design of very efficient analysers [11]. A lazy inference algorithm works in a bottom-up (or backwards) mode, starting with an initial query of the form $[\Gamma, e, \theta]$ (when trying to prove $\Gamma \vdash_g e : \theta$). First let us note that the initial query generates a finite domain of properties (which is, roughly speaking, the product of the user-defined properties by the set of variables occurring in the initial query), so the system is decidable. If we examine the rules in Figure 6, we can notice that the generic inference system is almost deterministic. Most of the rules can be translated directly into a rule for the algorithm by reading them in a bottom-up (and left to right) fashion. The only exceptions are the rules $[APP_1]$, $[APP_2]$ and $[FIX]$. Let us examine them in turn.

The rules for application are not deterministic because the property θ_2 occurs in the premise but not in the conclusion. So it has to be "guessed" by the algorithm. As was shown in [10], traditional abstract interpretation solves this problem by computing systematically the most precise property of an expression (in our setting, the conjunction of all the properties satisfied by the expression). This may lead to very inefficient analysers when the abstract domain becomes large (which is the case with higher-order languages). In contrast, the lazy algorithm treats an application $e_1 e_2$ very much like a lazy evaluator: the type θ_2 of e_2 is not computed until it is really needed in the proof. The rules of the lazy algorithm which correspond to $[APP_1]$ and $[APP_2]$ are:

$$[\mathrm{APP}_{a1}] \quad \frac{\Gamma \vdash_a e_1 : \pi_1 \quad \Gamma \vdash_a e_1 : [\Gamma, e_2] \rightarrow \pi_2}{\Gamma \vdash_a e_1 e_2 : \delta_i} \quad \text{if } (\pi_1, \pi_2) \in A(\delta_i)$$

$$[\mathrm{APP}_{a2}] \quad \frac{\Gamma \vdash_a e_1 : [\Gamma, e_2] \rightarrow \phi}{\Gamma \vdash_a e_1 e_2 : \phi}$$

The pair $[\Gamma, e_2]$ is a closure representing the conjunction of all the properties of e_2 in the environment Γ. The set of properties under consideration being finite, this notation is well-defined. As a consequence, the syntax of properties is enriched with the form $[\Gamma, e] \rightarrow \theta$ and the partial ordering on properties is extended to take this new form into account. For instance, we have:

$$\frac{\Gamma \vdash_a e : \theta}{[\Gamma, e] \leq \theta}$$

The rule corresponding to $[FIX]$ in the inference algorithm involves an iteration starting from the initial query $[\Gamma, (\mathbf{fix}\ \lambda f.\lambda x.e), \theta_0]$. The algorithm first tries to prove $[\Gamma + [f : \theta_0], (\lambda x.e), \theta_0]$. If this query fails a necessary condition θ_1 on f is returned and the iteration proceeds with the query $[\Gamma + [f : (\theta_0 \wedge \theta_1)], (\lambda x.e), (\theta_0 \wedge \theta_1)]$. The chain of properties associated with f is strictly decreasing and the iteration must converge. We do not dwell on the lazy

inference algorithm in this paper because it is a direct application of previous results. The interested reader can find a more detailed presentation of lazy types in [10] and experimental results in [11].

6 Related work

A standard denotational semantics and a non-standard path semantics for a lazy higher-order functional language are presented in [2]. A path analysis is then designed as a computable abstraction of the path semantics. Two applications of path analysis are studied: strictness analysis and destructive aggregate updating (the latter requiring an extended notion of path). In contrast with our definition of paths, only one occurrence of a variable can occur in a path. While this restriction is crucial in [2] to avoid infinite paths, it is unnecessary in our setting because paths are not manipulated *per se* in the analyses: they are abstracted through properties. Keeping all the occurrences allows us to state properties about the uniqueness of a reference to a variable. [17] describes a backwards analysis to infer evaluation order information for data structures in a typed lazy functional language. Evaluation order types are used to represent the order in which the parts of a data structure are evaluated. Again the notion of path slightly differs from the one considered here. The techniques described in [18] apply to both first and higher order languages with strict semantics and non-composite data values. A context free grammar is derived from a program and the instrumented operational semantics. This grammar safely describes the operational properties of the program, and questions about single-threading can be answered by a simple analysis of the grammar.

The most important departure of our work in comparison with previous contributions on path analysis is the mechanical derivation of path-based analyses from specifications of properties rather than the design of a proper path analysis. Considering analyses of particular properties on paths allows us to design efficient analysers without giving up accuracy. This is out of reach of current analysis techniques for the general path analysis problem.

A shortcoming of operational semantics in comparison with denotational semantics is that finite and infinite computations are not treated in a uniform way. As shown in section 3, we can still express non termination through the use of an empty property on paths. For instance a function defined as **fix** $\lambda f.\lambda x.\ f\ x$ satisfies the property **True** $\rightarrow$ **False**. Empty properties convey a *negative information*. A general formalism based on (positive and negative) inductive definitions is defined in [6] and it is illustrated with $G^\infty SOS$, a generalised structural operational semantics which makes it possible to reason in a uniform way on finite and infinite behaviours (the latter being defined by negative rules). It is shown in [7] that the standard strictness analysis of functional programs [12] can be derived as an abstraction of a $G^\infty SOS$. In contrast with this latter contribution, we do not attempt to recast the standard strictness analysis in the operational framework here. We rather focus on the more operational neededness property. Strictness can be recovered by integrating a divergence analysis: a function is strict if it needs its argument or it diverges for any argument (*i.e.* it satisfies property $\overline{x} \rightarrow \overline{Need}^{\,x}$ or **True** $\rightarrow$ **False**).

7 Further work

The work described in this paper suggests a general methodology for defining a static analysis based on natural semantics. The two main refinement steps are the following:

1. The definition of an abstraction function which extracts from sequents the pieces of information relevant to the considered analysis (namely paths in this paper). This abstraction leads to an instrumented semantics and a generic inference system.

2. The definition of a specific property by recurrence on abstract proof trees. Specific inference systems are derived from the property and the generic system obtained in the first step.

We have made specific choices in this paper (natural semantics of a non-strict functional language, abstraction in terms of paths) but we believe that the approach is more widely applicable. We just take two examples to illustrate other choices for the abstraction function:

1. Single-threading analysis can be described using an abstraction function which keeps track of both accesses A^x to variables of a specific type ρ (where ρ is the type to be globalised) and creations C^x of new values of type ρ. The former situation occurs in the rule [VAR] and the latter in the rule [APP]. The property ST^x stating that any execution is single-threaded in a variable x of type ρ can be defined by recurrence on abstract trees as we did in section 4 (checking that no abstract tree can include a pattern $C^x \ldots C^y \ldots A^x$).

2. Closure analysis can be specified using an abstraction function recording the pairs $(f, \lambda x.e)$ and defining properties to extract the relevant information from abstract trees.

We should emphasise also that the framework can accommodate different operational semantics. We are currently investigating the generality of the approach with the aim of designing a practical system to help in the design of static analyses based on operational semantics.

Another issue for further work is the design of a framework for proving the correctness of optimisations relying on static analyses described as abstractions of operational semantics. This would be the counterpart of the efforts described in [4, 8]. [19] presents such a proof for selective thunkification when transforming call-by-name into call-by-value. The proof of correspondence between a "big steps" operational semantics (natural semantics) and a "small steps" operational semantics, or an abstract machine, can be done as shown in [14].

References

[1] A. Bloss and P. Hudak, *Variations on strictness analysis*, in *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming Languages and Computer Architecture*, ACM Sigplan, pp. 132-142.

[2] A. Bloss, *Path analysis and the optimisation of nonstrict functional languages*, ACM TOPLAS, Vol. 16, No 3, May 1994, pp. 328-369.

[3] G. L. Burn, C. L. Hankin and S. Abramsky, *Strictness analysis for higher-order functions*, Science of Computer Programming, Nov. 1986, Vol. 7, pp. 249-278.

[4] G. Burn, D. Le Métayer, *Proving the correctness of compiler optimisations based on a global analysis*, Journal of Functional Programming, to appear.

[5] P. Cousot and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in *Proceedings* 4^{th} *POPL*, 1977, pp. 238-252.

[6] P. Cousot and R. Cousot, *Inductive definitions, semantics and abstract interpretation*, in *Proceedings* 19^{th} *POPL*, 1992, pp. 83-94.

[7] P. Cousot and R. Cousot, *Galois connection based abstract interpretations for strictness analysis*, in *Formal methods in programming and their applications*, D. Bjorner and M. Broy (eds.), Springer Verlag, 1993, pp. 98-127.

[8] O. Danvy and J. Hatcliff, *CPS transformation after strictness analysis*, Technical Report, Kansas State University, to appear in ACM LOPLAS.

[9] P. Fradet, *Syntactic detection of single-threading using continuations*, in *Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture*, LNCS 523, Springer Verlag, 1991, pp. 241-258.

[10] C. L. Hankin and D. Le Métayer, *Deriving algorithms from type inference systems: Application to strictness analysis*, in *Proceedings of POPL'94*, ACM Press, 1994, pp. 202-212.

[11] C. L. Hankin and D. Le Métayer, *A type-based framework for program analysis*, in *Proceedings of* 1^{st} *Static Analysis Symposium*, LNCS 864, Springer Verlag, 1994, pp. 380-394.

[12] A. Mycroft, *Abstract Interpretation and Optimising Transformations for Applicative Programs*, PhD thesis, University of Edinburgh, December 1981.

[13] F. Nielson, *A denotational framework for data flow analysis*, Acta Informatica 18, 1982, pp. 265-287.

[14] H. R. Nielson and F. Nielson, *Semantics with applications, a formal introduction*, Wiley, 1992.

[15] M. Rosendhal, *Higher-order chaotic iteration sequences*, in *Proceedings of the 5th Int. Symp. Programming Language Implementation and Logic Programming*, LNCS 714, Springer-Verlag, 1993.

[16] D. Schmidt, *Detecting global variables in denotational specifications*, ACM TOPLAS, Vol. 7, 1985, pp. 299-310.

[17] P. Sestoft, *Analysis and efficient implementation of functional programs*, PhD Thesis, Diku, Copenhagen, Denmark, October 1991.

[18] P. Sestoft, *Replacing function parameters by global variables*, in *Proceedings of the 1989 ACM Conference on Functional Programming Languages and Computer Architecture*, ACM Sigplan, pp. 39-153.

[19] P. Steckler and M. Wand, *Selective thunkification*, in *Proceedings of* 1^{st} *Static Analysis Symposium*, LNCS 864, Springer Verlag, 1994, pp. 162-178.

Appendix

$$[\text{VAL}]\ \ \Gamma \vdash_p b : Abs^{\,x} \qquad [\lambda_2]\ \ \Gamma \vdash_p (\lambda z.e) : Abs^{\,x}$$

$$[\text{VAR}_1]\ \ \Gamma[\alpha : \theta_\alpha] \vdash_p \alpha : Abs^{\,x} \quad \text{if}\ \ \theta_\alpha \leq Abs^{\,x} \qquad [\text{OP}]\ \ \frac{\Gamma \vdash_p e_1 : Abs^{\,x} \quad \Gamma \vdash_p e_2 : Abs^{\,x}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : Abs^{\,x}}$$

$$[\text{APP}_1]\ \ \frac{\Gamma \vdash_p e_1 : Abs^{\,x} \quad \Gamma \vdash_p e_1 : \theta_2 \rightarrow Abs^{\,x} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : Abs^{\,x}}$$

$$[\text{COND}_1]\ \ \frac{\Gamma \vdash_p e_1 : Abs^{\,x} \quad \Gamma \vdash_p e_2 : Abs^{\,x} \quad \Gamma \vdash_p e_3 : Abs^{\,x}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : Abs^{\,x}}$$

Figure 14: $Abs^{\,x}$ specific rules

$$[\text{VAL}]\ \ \Gamma \vdash_p b : Un^{\,x} \qquad [\lambda_2]\ \ \Gamma \vdash_p (\lambda z.e) : Un^{\,x}$$

$$[\text{VAR}_1^1]\ \ \Gamma[x : \theta_x] \vdash_p x : Un^{\,x} \quad \text{if}\ \ \theta_x \leq Abs^{\,x} \qquad [\text{VAR}_1^2]\ \ \Gamma[\alpha : \theta_\alpha] \vdash_p \alpha : Un^{\,x} \quad \text{if}\ \ \theta_\alpha \leq Abs^{\,x}$$

$$[\text{VAR}_1^3]\ \ \Gamma[\alpha : \theta_\alpha] \vdash_p \alpha : Un^{\,x} \quad \text{if}\ \ \theta_\alpha \leq Un^{\,x}$$

$$[\text{OP}^1]\ \ \frac{\Gamma \vdash_p e_1 : Un^{\,x} \quad \Gamma \vdash_p e_2 : Abs^{\,x}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : Un^{\,x}} \qquad [\text{OP}^2]\ \ \frac{\Gamma \vdash_p e_1 : Abs^{\,x} \quad \Gamma \vdash_p e_2 : Un^{\,x}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : Un^{\,x}}$$

$$[\text{APP}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : Un^{\,x} \quad \Gamma \vdash_p e_1 : \theta_2 \rightarrow Abs^{\,x} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : Un^{\,x}}$$

$$[\text{APP}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : Abs^{\,x} \quad \Gamma \vdash_p e_1 : \theta_2 \rightarrow Un^{\,x} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : Un^{\,x}}$$

$$[\text{COND}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : Un^{\,x} \quad \Gamma \vdash_p e_2 : Abs^{\,x} \quad \Gamma \vdash_p e_3 : Abs^{\,x}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : Un^{\,x}}$$

$$[\text{COND}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : Abs^{\,x} \quad \Gamma \vdash_p e_2 : Un^{\,x} \quad \Gamma \vdash_p e_3 : Un^{\,x}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : Un^{\,x}}$$

Figure 15: $Un^{\,x}$ specific rules

$$[\text{VAL}]\ \ \Gamma \vdash_p b : \mathit{Naft}^{\,x,y} \qquad\qquad [\lambda_2]\ \ \Gamma \vdash_p (\lambda z.e) : \mathit{Naft}^{\,x,y}$$

$$[\text{VAR}_1^1]\ \ \Gamma[x:\theta_x] \vdash_p x : \mathit{Naft}^{\,x,y} \quad \textbf{if}\ \ \theta_x \leq \mathit{Abs}^{\,y}$$

$$[\text{VAR}_1^2]\ \ \Gamma[y:\theta_y] \vdash_p y : \mathit{Naft}^{\,x,y} \quad \textbf{if}\ \ \theta_y \leq \mathit{Naft}^{\,x,y}$$

$$[\text{VAR}_1^3]\ \ \Gamma[y:\theta_y] \vdash_p y : \mathit{Naft}^{\,x,y} \quad \textbf{if}\ \ \theta_y \leq \mathit{Abs}^{\,y}$$

$$[\text{VAR}_1^4]\ \ \Gamma[\alpha:\theta_\alpha] \vdash_p \alpha : \mathit{Naft}^{\,x,y} \quad \textbf{if}\ \ \theta_\alpha \leq \mathit{Naft}^{\,x,y}$$

$$[\text{VAR}_1^5]\ \ \Gamma[\alpha:\theta_\alpha] \vdash_p \alpha : \mathit{Naft}^{\,x,y} \quad \textbf{if}\ \ \theta_\alpha \leq \mathit{Abs}^{\,y}$$

$$[\text{OP}^1]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Abs}^{\,x} \quad \Gamma \vdash_p e_2 : \mathit{Naft}^{\,x,y}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : \mathit{Naft}^{\,x,y}} \qquad [\text{OP}^2]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Naft}^{\,x,y} \quad \Gamma \vdash_p e_2 : \mathit{Abs}^{\,y}}{\Gamma \vdash_p e_1\ \mathbf{Op}\ e_2 : \mathit{Naft}^{\,x,y}}$$

$$[\text{APP}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Abs}^{\,x} \quad \Gamma \vdash_p e_1 : \theta_2 \rightarrow \mathit{Naft}^{\,x,y} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : \mathit{Naft}^{\,x,y}}$$

$$[\text{APP}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Naft}^{\,x,y} \quad \Gamma \vdash_p e_1 : \theta_2 \rightarrow \mathit{Abs}^{\,y} \quad \Gamma \vdash_p e_2 : \theta_2}{\Gamma \vdash_p e_1 e_2 : \mathit{Naft}^{\,x,y}}$$

$$[\text{COND}_1^1]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Abs}^{\,x} \quad \Gamma \vdash_p e_2 : \mathit{Naft}^{\,x,y} \quad \Gamma \vdash_p e_3 : \mathit{Naft}^{\,x,y}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : \mathit{Naft}^{\,x,y}}$$

$$[\text{COND}_1^2]\ \ \frac{\Gamma \vdash_p e_1 : \mathit{Naft}^{\,x,y} \quad \Gamma \vdash_p e_2 : \mathit{Abs}^{\,y} \quad \Gamma \vdash_p e_3 : \mathit{Abs}^{\,y}}{\Gamma \vdash_p \mathbf{cond}(e_1\ e_2\ e_3) : \mathit{Naft}^{\,x,y}}$$

Figure 16: $\mathit{Naft}^{\,x,y}$ specific rules

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399