J. Functional Programming **10** (6): 509–559, November 2000. Printed in the United Kingdom

*Automatic useless-code elimination for HOT functional programs**

FERRUCCIO DAMIANI
Dipartimento di Informatica, Università di Torino,
Corso Svizzera 185, 10149 Torino, Italy
(*e-mail:* damiani@di.unito.it)

PAOLA GIANNINI
DISTA, Università del Piemonte Orientale,
Corso Borsalino 54, 15100 Alessandria, Italy
(*e-mail:* giannini@di.unito.it)

Abstract

In this paper we present two type inference systems for detecting useless-code in higher-order typed functional programs. Type inference can be performed in an efficient and complete way, by reducing it to the solution of a system of constraints. We also give a useless-code elimination algorithm which is based on a combined use of these type inference systems. The main application of the technique is the optimization of programs extracted from proofs in logical frameworks, but it could be used as well in the elimination of useless-code determined by program transformations.

Capsule Review

Dead code is a subterm M of a term $t = C[M]$ which does not matter to computation, that is, which may be replaced by any other term of the same type without altering the observational behaviour of the program: $C[M] =_{obs} C[N]$ for any N with the type of M. Dead-code may be replaced by dummy variable, or even completely removed, in order to optimize a program.

The interest in dead-code by the authors started as a by-product of a proof system which generated programs meeting a specification out of proofs that the specification is satisfiable. Typically, the programs generated by such a system have plenty of dead-code. Yet, the authors' study seems of wider applicability, because the phenomenon of dead code arises in any program generated by indirect or automatic means. Indeed, a related approach, based also on the use of partial equivalence relation and pioneered by Hunt and Sands, existed already in the world of functional programming. Partial equivalence relations are used both to explain the typing system, and to prove that any term stated "dead-code" really is. The core of the paper is an original algorithm, described in great detail, which infers dead code from a combined use of three type inference systems.

* Partially supported by the Esprit TYPES W.G. and Cofin '99 TOSCA Project.

1 Introduction

Useless-code is any subexpression of a program that does not contribute to the final result of the program. Even though is unlikely to occur in human produced code, useless-code could be introduced by program transformations, or could be present in programs in which general-purpose functions taken from standard libraries are used in particular contexts.

To improve the quality of the code produced a compiler may perform a *useless-code analysis* to detect and remove useless-code in the source program. Useless-code analysis for typed functional programming languages has been originally studied in the context of *logical frameworks*, like Coq, where functional programs are extracted from formal proofs. See Pfenning (1996) for a short survey on logical frameworks, and Paulin-Mohring (1989a, 1989b) for an introduction to program extraction. Programs extracted from proofs usually contain large parts that are useless for the computation of their result. Therefore some sort of simplification is mandatory. To this aim various useless-code elimination techniques have been developed in the last ten years, see for instance, Takayama (1991) and Berardi (1996).

Useless-code elimination focused on the elimination of useless function's formal parameters is often called *useless-variable elimination* (Shivers, 1991). The original technique proposed in Berardi (1996) is essentially a useless-variable elimination, while extensions of this technique (Boerio, 1995; Berardi and Boerio, 1995; Berardi and Boerio, 1997) consider more general forms of useless-code.

In this paper we present two non-standard type assignment systems and simplification mappings for removing useless-code in simply typed higher-order functional programs. We also present a complete and incremental useless-code elimination algorithm based on these systems.

In the non-standard type inference approach to program analysis, properties are represented by particular types (called *non-standard types*), and program analyses are presented via a system of logical inference rules. These systems are usually built by two sets of rules. One set of rules, the 'subtyping' rules, determines an entailment relation between the properties under investigation, and another set of rules, the 'type assignment' rules, defines the way in which properties are assigned to the terms of the language. Non-standard type systems tailored to a specific analysis, such as strictness, totality, binding-time, etc. have been introduced in the literature (Kuo and Mishra, 1989; Jensen, 1992; Benton, 1992; Wright, 1992; Hankin and Le Métayer, 1995; Solberg, 1995; Barendsen and Smetsers, 1995; Dussart *et al.*, 1995; Mossin, 1997; Damiani, 1998).

The basic idea of the useless-code elimination techniques presented in this paper is inspired by the techniques of Berardi (1996) and Berardi and Boerio (1995). We decorate a typed functional program with non-standard types (called *evaluation types*). Such types are used to prove that the program can be evaluated without evaluating some of its subexpressions, which are therefore useless-code. For the programs of type nat (the type of natural numbers) there are two evaluation types:

- δ^{nat}, which is the property of the terms of type nat that can be evaluated, and so they can only be replaced by terms with the same value, and

- ω^{nat}, which is the property of the terms of type nat that are not evaluated, and so they can be replaced by any term of type nat.

Evaluation types for programs of higher types are defined from these basic evaluation types using the standard type construction. For instance the evaluation type $\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}$ is the property of all the functions of type $\mathsf{nat} \to \mathsf{nat}$ that, when applied to some argument, can be evaluated without using their argument (like $\lambda x^{\mathsf{nat}}.Q$ where x does not occur in Q). In other words, $\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}$ characterizes all the functions of type $\mathsf{nat} \to \mathsf{nat}$ that do not use their argument: such functions are constant. The evaluation type $\delta^{\mathsf{nat}} \to \delta^{\mathsf{nat}}$, instead, does not give any information about useless-code, since it says that the application of a function to an argument can be evaluated if the argument can be evaluated. To detect the useless-code in a given term M we will assume for it an evaluation type containing only δ's, i.e. we assume that M can be evaluated, and we will try to assign evaluation types containing only ω's to as many subterms of M as possible, in order to detect as many useless subterms as possible. Let us look at three simple examples of useless-code detection and elimination.

Example 1.1

- Take

$$M = (\lambda x^{\mathsf{nat}}.3)P$$

for some term P of type nat. Since x is never used in the body of the λ-abstraction $F = \lambda x^{\mathsf{nat}}.3$, we have that F has the property $\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}$. This implies that P is not used in the computation of the value of M and could be replaced by any term of the same type. In fact, in a call-by-name language, M behaves like the term

$$M_1 = (\lambda x^{\mathsf{nat}}.3)\mathsf{d}$$

where d is a place-holder for the useless-code removed.
Note that, in this case, it is possible to perform a further simplification and remove both the useless-variable x and the place-holder d, producing as final result the body of the λ-abstraction (the constant 3).

- Take

$$N = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle Q_1, Q_2\rangle)P$$

for some terms Q_1, Q_2, and P of type nat such that Q_1 does not contain any occurrence of x. Since the projection prj_1 returns the first component of the pair, the term Q_2 is useless-code. Moreover, since Q_1 does not contain any occurrence of x, the λ-abstraction $G = \lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle Q_1, Q_2\rangle$ has the property $\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}$. This implies that also P is useless-code. So, in a call-by-name language, N behaves like the term

$$N_1 = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle Q_1, \mathsf{d}\rangle)\mathsf{d}.$$

Also in this case it is possible to simplify further N_1, by removing the useless-variable x, the projection operator, and the pair constructor, obtaining the final term Q_1.

- In the previous examples the simplified terms were obtainable from the original

ones just by reduction[1], but this is not always the case. Let

$$R = (\lambda f^{\mathsf{nat}\to\mathsf{nat}}.fP)(\lambda x^{\mathsf{nat}}.3)$$

where P is a term of type nat. The function $H = \lambda f^{\mathsf{nat}\to\mathsf{nat}}.fP$ has the property $(\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}) \to \delta^{\mathsf{nat}}$, and $F = \lambda x^{\mathsf{nat}}.3$ has the property $\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}$. This implies that the subexpression P of R is useless-code. So, in a call-by-name language, R behaves like the term

$$R_1 = (\lambda f^{\mathsf{nat}\to\mathsf{nat}}.f\,\mathsf{d})(\lambda x^{\mathsf{nat}}.3).$$

In this case, a further simplification gives the simpler term $R_2 = (\lambda f^{\mathsf{nat}}.f)3$, which cannot be obtained from R by reduction.

The soundness of the evaluation type assignment systems and of the induced program transformations is proved via a partial equivalence relation semantics of the evaluation types, showing that the simplified programs are observationally equivalent to the original ones.

In section 2 of this paper we introduce the programming language we are dealing with and its operational semantics. Section 3 shows how program properties can be represented by partial equivalence relations on a term model of the programming language.

In section 4 we describe the language of evaluation types and its semantics, we also give a complete axiomatization of the logical implication between evaluation types. Section 5 introduces an evaluation type assignment system and a program simplification based on the information provided by the evaluation types assigned to a program and to its subexpressions. We prove that a program and its simplified version are observationally equivalent. Section 6 presents an evaluation type assignment system, which characterizes a proper subset of the useless-code characterized by the system in section 5: it restricts to useless-variable detection. The program simplification induced by this type system is 'deeper' than that of section 5, since it removes useless-variables without introducing dummy place-holders.

In section 7 we present two algorithms for inferring a representation of all the evaluation typings of a term for the systems of sections 5 and 6.

In section 8 we show that the best way of using the useless-code elimination techniques of Sections 5 and 6 is to compose them in a sequential way: first detecting and removing useless-code using the technique of section 5, and then applying the one of section 6 to the result.

In section 9 we discuss the independence of the techniques presented from the particular evaluation strategy of the language considered in the paper (call-by-name[2]). Related work is considered in section 10.

Appendix A contains the completeness proof for the e-type entailment relation between e-types (introduced in section 4), while in Appendix B we show how the algorithms described in section 7 can be integrated in an incremental useless-code detection and elimination procedure.

[1] By 'reduction' we mean β-reduction plus the reduction rules for projections.
[2] The evaluation strategy used by languages like Miranda, Haskell and Clean.

$$\text{(Var)}\ \frac{x^\rho \in \mathscr{V}}{x^\rho \in \Lambda_\rho} \qquad \text{(Con)}\ \frac{\mathsf{k}^\rho \in \mathscr{K}}{\mathsf{k}^\rho \in \Lambda_\rho}$$

$$(\to \text{I})\ \frac{M \in \Lambda_\sigma}{\lambda x^\rho.M \in \Lambda_{\rho\to\sigma}} \qquad (\to \text{E})\ \frac{M \in \Lambda_{\rho\to\sigma} \quad N \in \Lambda_\rho}{MN \in \Lambda_\sigma}$$

$$(\times \text{I})\ \frac{M_1 \in \Lambda_{\rho_1} \quad M_2 \in \Lambda_{\rho_2}}{\langle M_1, M_2\rangle \in \Lambda_{\rho_1\times\rho_2}} \qquad (\times \text{E}_i)\ \frac{M \in \Lambda_{\rho_1\times\rho_2}}{\mathsf{prj}_i M \in \Lambda_{\rho_i}}\ i \in \{1,2\}$$

$$\text{(Fix)}\ \frac{M \in \Lambda_\rho}{\mathsf{fix}\, x^\rho.M \in \Lambda_\rho} \qquad \text{(Ifz)}\ \frac{N \in \Lambda_{\mathsf{nat}} \quad M_1 \in \Lambda_\rho \quad M_2 \in \Lambda_\rho}{\mathsf{ifz}\, N\, \mathsf{then}\, M_1\, \mathsf{else}\, M_2 \in \Lambda_\rho}$$

Fig. 1. PCFP terms.

A preliminary version of the material presented in this paper appeared in Coppo *et al.* (1996) and Damiani (1998, Chap. 7).

2 The language PCFP

In this section we introduce a simple functional programming language and its operational semantics. We use the acronym PCFP (which stands for 'Programming Computable Functions with Pairs') for this language since it is the dialect of the language PCF (Plotkin, 1977) obtained by using natural numbers instead of integers and adding a type constructor for pairs.

2.1 PCFP syntax and evaluation rules

The set of PCFP types $\mathbf{T}$ is defined assuming as *ground* type the set of natural numbers, nat.

Definition 2.1 (*PCFP types*)
PCFP types, ranged over by ρ, σ, and τ (with superscripts and subscripts if needed), are defined by the following grammar: $\rho ::= \mathsf{nat} \mid \rho \to \rho \mid \rho \times \rho$.

PCFP terms are defined from a set of typed *term constants*

$$\mathscr{K} = \{\mathsf{0}^{\mathsf{nat}},\ \mathsf{1}^{\mathsf{nat}},\ \ldots,\ \mathsf{succ}^{\mathsf{nat}\to\mathsf{nat}},\ \mathsf{pred}^{\mathsf{nat}\to\mathsf{nat}},\ +^{\mathsf{nat}\times\mathsf{nat}\to\mathsf{nat}},\ -^{\mathsf{nat}\times\mathsf{nat}\to\mathsf{nat}}\}$$

(ranged over by k), and a set $\mathscr{V}$ of typed *term variables* (ranged over by $x^\rho, y^\sigma, \ldots$). The type of a constant k is denoted by $\mathbf{T}(\mathsf{k})$.

Definition 2.2 (*PCFP terms*)
The set of *PCFP terms* $\Lambda_{\mathbf{T}}$, ranged over by M, N, ..., is defined by:

$$\Lambda_{\mathbf{T}} = \bigcup_{\rho\in\mathbf{T}} \Lambda_\rho,$$

where the set Λ_ρ is defined by the rules in figure 1.

According to Definition 2.2, in a PCFP term M the types of variables and constants are explicitly mentioned. In the following we often omit to write types

$$\text{(CAN)}\ \frac{K \in \mathbf{V_T}}{K \Downarrow K} \qquad \text{(APP)}\ \frac{M \Downarrow \lambda x.P \quad P[x := N] \Downarrow K}{MN \Downarrow K}$$

$$(\mathscr{K}\text{APP}_1)\ \frac{M \Downarrow \mathsf{k} \quad N \Downarrow \mathsf{k}_1}{MN \Downarrow \mathsf{k}_2}\ (\mathsf{k}_1, \mathsf{k}_2) \in \mathbf{mean}(\mathsf{k})$$

$$(\mathscr{K}\text{APP}_2)\ \frac{M \Downarrow \mathsf{k} \quad N \Downarrow \langle N_1, N_2\rangle \quad N_1 \Downarrow \mathsf{k}_1 \quad N_2 \Downarrow \mathsf{k}_2}{MN \Downarrow \mathsf{k}_3}\ (\langle \mathsf{k}_1, \mathsf{k}_2\rangle, \mathsf{k}_3) \in \mathbf{mean}(\mathsf{k})$$

$$(\text{PRJ}_i)\ \frac{P \Downarrow \langle M_1, M_2\rangle \quad M_i \Downarrow K}{\mathsf{prj}_i P \Downarrow K}\ i \in \{1,2\} \qquad \text{(FIX)}\ \frac{M[x := \mathsf{fix}\, x.M] \Downarrow K}{\mathsf{fix}\, x.M \Downarrow K}$$

$$(\text{IFZ}_1)\ \frac{N \Downarrow \mathsf{0} \quad M_1 \Downarrow K}{\mathsf{ifz}\, N\, \mathsf{then}\, M_1\, \mathsf{else}\, M_2 \Downarrow K} \qquad (\text{IFZ}_2)\ \frac{N \Downarrow \mathsf{n} \quad M_2 \Downarrow K}{\mathsf{ifz}\, N\, \mathsf{then}\, M_1\, \mathsf{else}\, M_2 \Downarrow K}\ \mathsf{n} \neq \mathsf{0}$$

Fig. 2. 'Natural semantics' evaluation rules.

which are clear from the context. The finite set of the *free variables* of a term M, denoted by $\text{FV}(M)$, is defined in the standard way. In the following we identify terms modulo renaming of the bound variables.

As usual a substitution is a finite function mapping term variables to terms, denoted by $[x_1 := N_1, \ldots, x_n := N_n]$ ($[\vec{x} := \vec{N}]$ for short), which respects the types, i.e. each $x_i^{\rho_i}$ is substituted by a term N_i of the same type. Substitution acts on free variables, the renaming of the bound variables is implicitly assumed.

Let $\Lambda_{\mathbf{T}}^{\mathbf{c}} = \{M \mid M \in \Lambda_{\mathbf{T}} \text{ and } \text{FV}(M) = \emptyset\}$ be the set of the *closed* PCFP terms. The process of evaluating a program is specified in a standard way by giving a *structural operational semantics* (Plotkin, 1981; Kahn, 1988) in the form of an inductively defined *evaluation relation*, $M \Downarrow K$, where M is a closed term and K is a closed term in *weak head normal form* (*whnf*), i.e. an element of the set of *values*

$$\mathbf{V_T} = \mathscr{K} \cup \{\lambda x^{\rho}.N \mid \lambda x^{\rho}.N \in \Lambda_{\mathbf{T}}^{\mathbf{c}}\} \cup \{\langle M_1, M_2\rangle \mid \langle M_1, M_2\rangle \in \Lambda_{\mathbf{T}}^{\mathbf{c}}\}.$$

Any PCFP constant has either type nat or $\mathsf{nat} \to \mathsf{nat}$ or $\mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$. The meaning of a functional constant k is given by a set $\mathbf{mean}(\mathsf{k})$ of pairs, such that if $(K, \mathsf{k}_1) \in \mathbf{mean}(\mathsf{k})$ then $\mathsf{k}K$ evaluates to k_1. For example $(\mathsf{5}, \mathsf{6}) \in \mathbf{mean}(\mathsf{succ})$ and $(\langle \mathsf{1}, \mathsf{3}\rangle, \mathsf{4}) \in \mathbf{mean}(+)$.

Definition 2.3 (*Value of a term*)
Let $M \in \Lambda_{\mathbf{T}}^{\mathbf{c}}$. We write $M \Downarrow K$, and say that M *evaluates to* K, if this statement is derivable by using the rules in figure 2.

Let $M \Downarrow$, to be read 'M is convergent', mean that, for some K, $M \Downarrow K$, and let $M \Uparrow$, to be read 'M is divergent', mean that, for no K, $M \Downarrow K$.

2.2 Ground contextual equivalence and bisimilarity

Two expressions M and N of a programming language are equivalent if any occurrence of M in a complete program can be replaced by N, and vice versa, without changing the results of program execution. To formalize this for a particular

programming language it is necessary to specify which are the observable results of program execution.

Following Pitts (1997), we introduce the *ground contextual equivalence* on PCFP terms, which is the congruence on terms induced by the contextual preorder that compares the termination behaviour of programs just at the ground type nat. This amounts to assume that *complete PCFP programs* are closed terms of type nat, and that the only observable behaviour of a complete program P is its divergence or converge to some natural number.

Let $(\mathscr{C}[\ ]^{\rho})^{\sigma}$ denote a typed context of type σ with a hole of type ρ in it.

Definition 2.4 (*Ground contextual equivalence*)
Let M and N be terms of type ρ. Define $M \preceq_{\mathrm{obs}} N$ whenever, for all contexts $(\mathscr{C}[\ ]^{\rho})^{\mathsf{nat}}$, if $\mathscr{C}[M]$ and $\mathscr{C}[N]$ are closed terms, then $\mathscr{C}[M] \Downarrow$ implies $\mathscr{C}[N] \Downarrow$. The relation $\preceq_{\mathrm{obs}}$ is the *ground contextual preorder* and the equivalence induced by $\preceq_{\mathrm{obs}}$, denoted by $\simeq_{\mathrm{obs}}$, is the *ground observational equivalence*.

In Pitts (1997), a co-inductive characterization of ground contextual equivalence in terms of bisimulation is given. This characterization allow to prove more easily the equivalence results on terms needed to show that the optimizations presented in this paper are correct. We give here the definition of bisimulation for PCFP, and state the two main results presented in Pitts (1997), namely Theorems 2.6 and 2.8. Such results will be used in this paper.

Definition 2.5 (*PCFP bisimulations and bisimilarity*)
A *PCFP bisimulation* $\mathscr{B}$ is a type indexed family of relations on closed terms,

$$\mathscr{B}_{\rho} \subseteq \{(M,N) \mid M,N \in \Lambda_{\rho} \text{ and } \mathrm{FV}(M) \cup \mathrm{FV}(N) = \emptyset\}$$

satisfying the conditions in figure 3. *PCFP bisimilarity* is the largest bisimulation and will be denoted by $\simeq$.

A PCFP bisimulation $\mathscr{B}$ can be extended to a relation on (possibly) open terms $\mathscr{B}^{\mathbf{o}}$ (called the *open extension* of $\mathscr{B}$). For any PCFP terms P and Q such that $\mathrm{FV}(P) \cup \mathrm{FV}(Q) \subseteq \{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}$, define: $M \mathscr{B}^{\mathbf{o}}_{\rho} N$ if and only if, for every substitution $[x_1^{\sigma_1} := N_1, \ldots, x_n^{\sigma_n} := N_n]$ such that $\bigcup_{1 \leqslant i \leqslant n} \mathrm{FV}(N_i) = \emptyset$,

$$P[\vec{x} := \vec{N}] \ \mathscr{B}_{\rho} \ Q[\vec{x} := \vec{N}].$$

Let $\simeq^{\mathbf{o}}$ denote the *PCFP open bisimilarity*. The following results hold.

Theorem 2.6 (*Operational extensionality for PCFP*)
Ground contextual equivalence coincides with bisimilarity, that is:

$$M \simeq_{\mathrm{obs}} N \text{ if and only if } M \simeq^{\mathbf{o}} N.$$

Consider, for every type ρ, the set of everywhere divergent terms of type ρ. That is for type nat the always divergent terms, for an arrow type the terms that for every input produce an everywhere divergent term, and for pairs that terms whose projections are terms everywhere divergent. Notice that terms in these sets are all observationally equivalent. In particular, the term $\mathsf{fix}\, x^{\rho}.x$ is a 'canonical' representative of the set of the everywhere divergent terms of type ρ.

(bis 1a)	$(M\ \mathscr{B}_{\mathsf{nat}}\ N$	and $M \Downarrow \mathsf{k})$	implies $N \Downarrow \mathsf{k}$
(bis 1b)	$(M\ \mathscr{B}_{\mathsf{nat}}\ N$	and $N \Downarrow \mathsf{k})$	implies $M \Downarrow \mathsf{k}$
(bis 2)	$M\ \mathscr{B}_{\rho\to\sigma}\ N$	implies	for all $P \in \Lambda_\rho$, $MP\ \mathscr{B}_\sigma\ NP$
(bis 3)	$M\ \mathscr{B}_{\rho\times\sigma}\ N$	implies	$\mathsf{prj}_1(M)\ \mathscr{B}_\rho\ \mathsf{prj}_1(N)$ and $\mathsf{prj}_2(M)\ \mathscr{B}_\sigma\ \mathsf{prj}_2(N)$

Fig. 3. Bisimulation conditions for PCFP

The following definition introduces the set of finite approximations of a recursive term $\mathsf{fix}\,x.M$. As usual approximations are defined starting from an everywhere divergent term, and iterating the substitution of the recursion variable with the term defined at the previous stage.

Definition 2.7
For all natural numbers m define $\mathsf{fix}^{(m)}\,x^\rho.M$ as follows:

$$\begin{aligned}\mathsf{fix}^{(0)}\,x^\rho.M &= \mathsf{fix}\,x^\rho.x\\ \mathsf{fix}^{(n+1)}\,x^\rho.M &= M[x^\rho := \mathsf{fix}^{(n)}\,x^\rho.M].\end{aligned}$$

Theorem 2.8 ('*Compactness*' *of evaluation*)
For every PCFP context of type nat, $(\mathscr{C}[\ ]^\rho)^{\mathsf{nat}}$, if $(\mathscr{C}[\mathsf{fix}\,x.M]^\rho)^{\mathsf{nat}} \Downarrow \mathsf{k}$ then exists $m \geqslant 0$ such that $(\mathscr{C}[\mathsf{fix}^{(m)}\,x.M]^\rho)^{\mathsf{nat}} \Downarrow \mathsf{k}$.

2.3 A closed term model of PCFP

Let $\mathbf{I}(\rho)$ denote the set of the equivalence classes of the relation $\simeq_{\mathrm{obs}}$ on the closed terms of type ρ in $\Lambda^{\mathbf{c}}_{\mathbf{T}}$, and let $[M]$ denote the equivalence class of the closed term M. The preorder $\preceq_{\mathrm{obs}}$ induces a partial order on $\mathbf{I}(\rho)$: define $[M] \preceq_{\mathrm{obs}} [N]$ if and only if $M \preceq_{\mathrm{obs}} N$.

The *closed term model* $\mathscr{M}$ of PCFP is defined by interpreting each type ρ as the poset $\mathbf{I}(\rho)$ ordered by $\preceq_{\mathrm{obs}}$. For every type ρ, $[\mathsf{fix}\,x^\rho.x]$ is the least element of $\mathbf{I}(\rho)$. An *environment* is a mapping $e : \mathscr{V} \to \bigcup_{\rho\in\mathbf{T}} \mathbf{I}(\rho)$ which respects types, i.e. a mapping such that, for all x^ρ, $e(x^\rho) \in \mathbf{I}(\rho)$. The interpretation of a term M in an environment e is defined in a standard way by:

$$[\![M]\!]_e = [M[x_1 := N_1, \ldots, x_n := N_n]],$$

where $\{x_1, \ldots, x_n\} = \mathrm{FV}(M)$ and $[N_l] = e(x_l)$ $(1 \leqslant l \leqslant n)$. Note that, using Theorem 2.6, we can prove that $\mathscr{M}$ is an extensional model.

3 Partial equivalence relations and program properties

In this section we show how some properties of PCFP programs P of type ρ can be represented as partial equivalence relations over the interpretation, $\mathbf{I}(\rho)$, of the type ρ in the closed term model $\mathscr{M}$.

A *Partial Equivalence Relation* (*PER* for short) over a set A is a symmetric and transitive binary relation over A. Let $\mathscr{R}$ be a PER over a set A, we call *domain of* $\mathscr{R}$

the subset of A defined as:

$$\mathrm{Dom}(\mathscr{R}) = \{E \mid (E, E') \in \mathscr{R} \text{ or } (E', E) \in \mathscr{R} \text{ for some } E'\}.$$

Note that $\mathscr{R}$ is reflexive on its domain. The use of PERs is well known in literature on program analysis, and dates at least back to Hunt and Sands (1991).

In the following let 'PER over a type ρ' mean 'PER over the set $\mathbf{I}(\rho)$'. A PER $\mathscr{R}$ over ρ is an equivalence relation between the subset $\mathrm{Dom}(\mathscr{R})$ of the meanings $\mathbf{I}(\rho)$ of terms in the model $\mathscr{M}$: the elements of $\mathrm{Dom}(\mathscr{R})$ that are in the same equivalence class are identified by the property expressed by $\mathscr{R}$. The following definition explains what is meant by: a term P of type ρ satisfies the property (PER) $\mathscr{R}$ over ρ.

Definition 3.1 (P *satisfies* $\mathscr{R}$)
Let $\mathscr{R}, \mathscr{R}_1, \ldots, \mathscr{R}_n$ $(n \geqslant 0)$ be PER over types $\rho, \rho_1, \ldots, \rho_n$, respectively. We say that a term P of type ρ with free variables $x_1^{\rho_1}, \ldots, x_n^{\rho_n}$ *satisfies* the property $\mathscr{R}$ under the assumptions $\mathscr{R}_i$ for $x_i^{\rho_i}$ $(1 \leqslant i \leqslant n)$ if, for all the environments e and e' such that $(e(x_i^{\rho_i}), e'(x_i^{\rho_i})) \in \mathscr{R}_i$, we have that $(\llbracket P \rrbracket_e, \llbracket P \rrbracket_{e'}) \in \mathscr{R}$.

A PER $\mathscr{R}$ over ρ can be read as a relation between closed terms such that $M, N \in \Lambda^{\mathsf{c}}_{\mathbf{T}}$ are indistinguishable w.r.t. $\mathscr{R}$ if and only if $([M], [N]) \in \mathscr{R}$. This view is useful to understand which is the property represented by a given PER.

- For every type $\rho \in \mathbf{T}$, consider the diagonal PER over $\mathbf{I}(\rho)$,

  $$\Delta^\rho = \{([M], [M]) \mid [M] \in \mathbf{I}(\rho)\}.$$

  The relation Δ^ρ is the PCFP bisimilarity (it identifies two closed terms of type ρ if and only if they are observationally equivalent), so it represents the property for which the value of a term *does matter*. In other words, the relation Δ^ρ has a different equivalent class for each of the meanings of the terms of type ρ.
- The trivial PER over $\mathbf{I}(\rho)$,

  $$\Omega^\rho = \{([M], [N]) \mid [M], [N] \in \mathbf{I}(\rho)\},$$

  identifies all the closed terms of type ρ, so it represents the 'true' property for which the value of a term *does not matter*. In other words, the relation Ω^ρ consists of one equivalence class containing all the meanings of the terms of type ρ.
- Given a PER $\mathscr{R}_1$ over ρ_1 and a PER $\mathscr{R}_2$ over ρ_2, let $\mathscr{R}_1 \twoheadrightarrow \mathscr{R}_2$ be the PER over $\rho_1 \to \rho_2$ defined as follows:

  $$\mathscr{R}_1 \twoheadrightarrow \mathscr{R}_2 \quad = \quad \{([F], [G]) \mid \forall([M], [N]) \in \mathscr{R}_1.([FM], [GN]) \in \mathscr{R}_2\}.$$

  For instance, $\Omega^{\mathsf{nat}} \to \Delta^{\mathsf{nat}}$ can be seen as the property of being a constant function of type $\mathsf{nat} \to \mathsf{nat}$, since two closed terms P and Q are identified by $\Omega^{\mathsf{nat}} \to \Delta^{\mathsf{nat}}$ if and only if: for all $M, N \in \Lambda^{\mathsf{c}}_{\mathsf{nat}}$, PM and QN are observationally equivalent. So the value returned by P and Q must be the same, and must be independent from their input.
  According to the $\mathscr{M}$ semantics we have that, for every $\rho_1, \rho_2 \in \mathbf{T}$,

— $\Delta^{\rho_1} \twoheadrightarrow \Delta^{\rho_2} = \Delta^{\rho_1 \to \rho_2}$, and
— $\Omega^{\rho_1} \twoheadrightarrow \Omega^{\rho_2} = \Omega^{\rho_1 \to \rho_2} = \mathscr{R}_1 \twoheadrightarrow \Omega^{\rho_2}$ (for all the PERs $\mathscr{R}_1$ over ρ_1).

- Given a PER $\mathscr{R}_1$ over ρ_1 and a PER $\mathscr{R}_2$ over ρ_2, let $\mathscr{R}_1 \divideontimes \mathscr{R}_2$ be the PER over $\rho_1 \times \rho_2$ defined as follows:

$$\mathscr{R}_1 \divideontimes \mathscr{R}_2 \quad = \quad \{([M],[N]) \mid \forall i \in \{1,2\}.([\mathsf{prj}_i M],[\mathsf{prj}_i N]) \in \mathscr{R}_i\}.$$

Observe that, for any pair of closed terms (P,Q), the fact that $([P],[Q]) \in \mathscr{R}_1 \divideontimes \mathscr{R}_2$ does not imply neither that $P \Downarrow \langle P_1, P_2\rangle$, for some P_1 and P_2 nor $Q \Downarrow \langle Q_1, Q_2\rangle$, for some Q_1 and Q_2. In particular this may not be the case when $\mathscr{R}_1 = \Omega^{\rho_1}$ and $\mathscr{R}_2 = \Omega^{\rho_2}$.
It is easy to show that
— $\Delta^{\rho_1} \divideontimes \Delta^{\rho_2} = \Delta^{\rho_1 \times \rho_2}$, and
— $\Omega^{\rho_1} \divideontimes \Omega^{\rho_2} = \Omega^{\rho_1 \times \rho_2}$.

Example 3.2
The term $G = (\lambda z^{\mathsf{nat}}.\lambda y^{\mathsf{nat}\times\mathsf{nat}}.+\langle \mathsf{prj}_1 y, x_1\rangle)\, x_2$ satisfies the property

$$(\Delta^{\mathsf{nat}} \divideontimes \Omega^{\mathsf{nat}}) \twoheadrightarrow \Delta^{\mathsf{nat}},$$

under the assumptions Δ^{nat} for the free variable x_1 and Ω^{nat} for the free variable x_2. In fact,

- the free variable x_2 is not used, and
- the second component of any pair of numbers to which G is applied is not used.

The set-theoretic inclusion between PERs over a type ρ represents a logical implication between properties, i.e. if $\mathscr{R}_1 \subseteq \mathscr{R}_2$ and a pair of closed terms (P,Q) is identified by $\mathscr{R}_1$, then (P,Q) is also identified by $\mathscr{R}_2$. For instance, for every PER $\mathscr{R}$ over ρ, $\mathscr{R} \subseteq \Omega^{\rho}$, according to the fact that Ω^{ρ} represents the 'true' property.

4 Evaluation types

In this section we introduce a set of non-standard types over **T**, the set of the *evaluation types* (*e-types* for short), which is at the heart of the program analysis and transformation technique presented in this paper. We first introduce the e-types syntax and semantics and then a complete axiomatization of the logical implication between e-types.

4.1 E-types syntax and semantics

Let ϕ range over e-types and ϕ^{ρ} range over e-types with *underlying type* ρ. In the following we will often omit the superscript ρ when it is either not relevant or clear from the context. For an e-type $\phi \in \mathbf{L}(\rho)$, let $\epsilon(\phi)$ denote the underlying type, ρ, of ϕ.

Definition 4.1 (*Evaluation types*)
The set of the e-types, **L**, is defined by: $\mathbf{L} = \cup_{\rho \in \mathbf{T}} \mathbf{L}(\rho)$, where the sets $\mathbf{L}(\rho)$ are defined by the rules in figure 4. The symbols δ and ω are called *basic properties.*

$$(\delta)\ \delta^{\mathsf{nat}} \in \mathbf{L}(\mathsf{nat}) \qquad (\omega)\ \omega^{\rho} \in \mathbf{L}(\rho) \qquad (\rightarrow)\ \frac{\phi_1 \in \mathbf{L}(\rho_1) \quad \phi_2 \in \mathbf{L}(\rho_2)}{\phi_1 \rightarrow \phi_2 \in \mathbf{L}(\rho_1 \rightarrow \rho_2)}\ \phi_2 \neq \omega^{\rho_2}$$

$$(\times)\ \frac{\phi_1 \in \mathbf{L}(\rho_1) \quad \phi_2 \in \mathbf{L}(\rho_2)}{\phi_1 \times \phi_2 \in \mathbf{L}(\rho_1 \times \rho_2)}\ \exists i \in \{1,2\}.\phi_i \neq \omega^{\rho_i}$$

Fig. 4. Evaluation types.

The meaning of e-types is given by interpreting each e-type, ϕ^{ρ}, as a PER, $[\![\phi^{\rho}]\!]$ over the interpretation of the type ρ in the model $\mathscr{M}$ (see Section 3).

Definition 4.2 (*Semantics of e-types*)
The interpretation $[\![\phi]\!]$ of an e-type ϕ is defined by:

$$\begin{aligned}
[\![\delta^{\mathsf{nat}}]\!] &= \Delta^{\mathsf{nat}} \\
[\![\omega^{\rho}]\!] &= \mathbf{\Omega}^{\rho} \\
[\![\phi_1 \rightarrow \phi_2]\!] &= [\![\phi_1]\!] \twoheadrightarrow [\![\phi_2]\!] \\
[\![\phi_1 \times \phi_2]\!] &= [\![\phi_1]\!] \mathbin{\ast\!\!\!\ast} [\![\phi_2]\!].
\end{aligned}$$

The previous definition justifies the choice of not having e-types of the form

- $\phi^{\rho_1} \rightarrow \omega^{\rho_2}$ and $\omega^{\rho_1} \times \omega^{\rho_2}$ (where ϕ is any e-type, and ρ_1 and ρ_2 are any types), since the first would denote the equivalence relation $\mathbf{\Omega}^{\rho_1 \rightarrow \rho_2}$, and the second would denote $\mathbf{\Omega}^{\rho_1 \times \rho_2}$, and
- δ^{ρ} where $\rho \neq \mathsf{nat}$, since $\Delta^{\rho_1} \twoheadrightarrow \Delta^{\rho_2} = \Delta^{\rho_1 \rightarrow \rho_2}$ and $\Delta^{\rho_1} \mathbin{\ast\!\!\!\ast} \Delta^{\rho_2} = \Delta^{\rho_1 \times \rho_2}$ for all ρ_1 and ρ_2.

Indeed, the e-type syntax has been defined in such a way that (syntactically) different e-types denote different PERs.

A useful property of the e-types semantics is the following.

Proposition 4.3
If, for all $n \geqslant 0$, $([(\mathscr{C}_1[\mathsf{fix}^{(n)}\, x.M]^{\rho})^{\sigma}], [(\mathscr{C}_2[\mathsf{fix}^{(n)}\, x.M]^{\rho})^{\sigma}]) \in [\![\phi^{\sigma}]\!]$, then
$([(\mathscr{C}_1[\mathsf{fix}\, x.M]^{\rho})^{\sigma}], [(\mathscr{C}_2[\mathsf{fix}\, x.M]^{\rho})^{\sigma}]) \in [\![\phi^{\sigma}]\!]$

Proof
By structural induction on ϕ^{σ}.

$\underline{\phi^{\sigma} = \omega^{\sigma}}$**.** Immediate.
$\underline{\phi^{\sigma} = \delta^{\mathsf{nat}}}$**.** The result is equivalent to Theorem 2.8.
$\underline{\phi^{\sigma} = \phi_1^{\sigma_1} \rightarrow \phi_2^{\sigma_2}}$**.** By absurd. Assume that $([(\mathscr{C}_1[\mathsf{fix}\, x.M]^{\rho})^{\sigma}], [(\mathscr{C}_2[\mathsf{fix}\, x.M]^{\rho})^{\sigma}]) \notin [\![\phi^{\sigma}]\!]$. Then, for some $([P], [Q]) \in [\![\phi_1^{\sigma_1}]\!]$, we have

$$([(\mathscr{C}_1[\mathsf{fix}\, x.M]^{\rho})^{\sigma} P], [(\mathscr{C}_2[\mathsf{fix}\, x.M]^{\rho})^{\sigma} Q]) \notin [\![\phi_2^{\sigma_2}]\!].$$

By induction hypothesis there exists $m \geqslant 0$ such that $([(\mathscr{C}_1[\mathsf{fix}^{(m)}\, x.M]^{\rho})^{\mathsf{nat}} P]$, $[(\mathscr{C}_2[\mathsf{fix}^{(m)}\, x.M]^{\rho})^{\mathsf{nat}} Q]) \notin [\![\phi_2^{\sigma_2}]\!]$. Therefore

$$([(\mathscr{C}_1[\mathsf{fix}^{(m)}\, x.M]^{\rho})^{\mathsf{nat}}], [(\mathscr{C}_2[\mathsf{fix}^{(m)}\, x.M]^{\rho})^{\mathsf{nat}}]) \notin [\![\phi^{\sigma}]\!].$$

$$\text{(Ref)}\ \phi \leqslant \phi \qquad (\omega)\ \phi^{\rho} \leqslant \omega^{\rho} \qquad (\to)\ \frac{\psi_1 \leqslant \phi_1 \quad \phi_2 \leqslant \psi_2}{\phi_1 \to \phi_2 \leqslant \psi_1 \to \psi_2}\ \psi_2 \neq \omega^{\epsilon(\psi_2)}$$

$$(\times)\ \frac{\phi_1 \leqslant \psi_1 \quad \phi_2 \leqslant \psi_2}{\phi_1 \times \phi_2 \leqslant \psi_1 \times \psi_2}\ \exists i \in \{1,2\}.\psi_i \neq \omega^{\epsilon(\psi_i)}$$

Fig. 5. Entailment rules for e-types (system $\leqslant$).

$\underline{\phi^{\sigma} = \phi_1^{\sigma_1} \times \phi_2^{\sigma_2}}$. Similar.

□

4.2 A complete entailment relation for e-types

In this section we introduce an entailment relation between e-types, $\leqslant$. This relation models the set-theoretic inclusion between the interpretation of the e-types, representing the logical implication between properties.

Definition 4.4 (*Entailment relation* $\leqslant$)
Let $\phi, \psi \in \mathbf{L}$. We write $\phi \leqslant \psi$ to mean that $\phi \leqslant \psi$ is derivable by the rules in Fig. 5. By $\cong$ we denote the equivalence relation induced by $\leqslant$.

It is immediate to show that $\leqslant$ is reflexive and transitive. Moreover $\phi \cong \psi$ implies $\phi = \psi$.

The $\leqslant$ entailment relation between e-types is sound and complete w.r.t. the PER interpretation of Definition 4.2.

Theorem 4.5 (*Soundness of* $\leqslant$)
$\phi \leqslant \psi$ implies $[\![\phi]\!] \subseteq [\![\psi]\!]$.

Proof
By induction on the structure of the derivations. □

The proof of completeness relies on the definition, for each e-type ϕ, of a set of pairs of terms, $\mathbf{Chr}^{\phi}$, which characterize the pairs of terms in $[\![\phi]\!]$. In particular, the characteristic sets $\mathbf{Chr}^{\phi}$ have the properties expressed by the following lemma.

Lemma 4.6 (*Characteristic set* $\mathbf{Chr}^{\phi}$)
1. $\{([P],[Q]) \mid (P,Q) \in \mathbf{Chr}^{\phi}\} \subseteq [\![\phi]\!]$.
2. $\{([P],[Q]) \mid (P,Q) \in \mathbf{Chr}^{\phi}\} \subseteq [\![\psi]\!]$ implies $\phi \leqslant \psi$.

The construction of the set $\mathbf{Chr}^{\phi}$, which is rather technical, and the proof of Lemma 4.6 are presented in the Appendix A.

Theorem 4.7 (*Completeness of* $\leqslant$)
$[\![\phi]\!] \subseteq [\![\psi]\!]$ implies $\phi \leqslant \psi$.

Proof
By absurd. Assume $\phi \not\leqslant \psi$. Then by Lemma 4.6.2 $\{([P],[Q]) \mid (P,Q) \in \mathbf{Chr}^{\phi}\} \not\subseteq [\![\psi]\!]$ and by Lemma 4.6.1 $\{([P],[Q]) \mid (P,Q) \in \mathbf{Chr}^{\phi}\} \subseteq [\![\phi]\!]$. Therefore $[\![\phi]\!] \not\subseteq [\![\psi]\!]$. □

4.3 ω-types and δ-e-types

In this section we introduce two sets of e-types: the ω-e-types and the δ-e-types.

Definition 4.8 (ω-*e-types and* δ-*e-types*)

1. We call ω-e-types the e-types in the set $\mathbf{L}_\omega = \{\omega^\rho \mid \rho \in \mathbf{T}\}$.
2. The set of the δ-e-types ($\mathbf{L}_\delta$) is the subset of the e-types which do not contain subexpressions of the form ω^ρ (for some ρ). For every type $\rho \in \mathbf{T}$, let $\delta(\rho)$ be the unique δ-e-type of type ρ.

The following proposition shows that ω-e-types provide a syntactical characterization for the set of the e-types ϕ^ρ such that $[\![\phi^\rho]\!] = \Omega^\rho$ and δ-e-types for the set of the e-types ϕ^ρ such that $[\![\phi^\rho]\!] = \Delta^\rho$.

Proposition 4.9

1. $\phi^\rho \in \mathbf{L}_\omega$ if and only if $[\![\phi^\rho]\!] = \Omega^\rho$, and
2. $\phi^\rho \in \mathbf{L}_\delta$ if and only if $[\![\phi^\rho]\!] = \Delta^\rho$.

Proof

1. ($\Longrightarrow$). Immediate from Definition 4.2.
   ($\Longleftarrow$). We prove that $\phi^\rho \neq \omega^\rho$ implies $[\![\phi^\rho]\!] = \Omega^\rho$. Let $\phi^\rho \neq \omega^\rho$. By structural induction on $\phi \in \mathbf{L}$.

   $\phi = \delta^{\mathsf{nat}}$. Directly from Definition 4.2.

   $\phi^\rho = {\phi_1}^{\rho_1} \to {\phi_2}^{\rho_2}$. By definition of e-type, $\phi_2 \notin \mathbf{L}_\omega$, and, by induction, $[\![\phi_2]\!] \neq \Omega^{\rho_2}$. So there are $[P], [Q] \in \mathbf{I}(\rho_2)$ such that $([P], [Q]) \notin [\![\phi_2]\!]$. This implies that $([\lambda z^{\rho_1}.P], [\lambda z^{\rho_1}.Q]) \notin [\![\phi_1 \to \phi_2]\!]$, and so $[\![\phi_1 \to \phi_2]\!] \neq [\![\omega^{\rho_1 \to \rho_2}]\!]$.

   $\phi^\rho = {\phi_1}^{\rho_1} \times {\phi_2}^{\rho_2}$. By definition of e-type either $\phi_1 \notin \mathbf{L}_\omega$ or $\phi_2 \notin \mathbf{L}_\omega$. So, by induction, either $[\![\phi_1]\!] \notin \mathbf{L}_\omega$ or $[\![\phi_2]\!] \notin \mathbf{L}_\omega$. This implies $[\![\phi_1 \times \phi_2]\!] \neq [\![\omega^{\rho_1 \times \rho_2}]\!]$.

2. ($\Longrightarrow$). We prove that, for all $\rho \in \mathbf{T}$, $[\![\delta(\rho)]\!] = \Delta^\rho$. By induction on $\rho \in \mathbf{T}$.

   $\rho = \mathsf{nat}$. Immediate.

   $\rho = \rho_1 \to \rho_2$. By hypothesis both $[\![\delta(\rho_1)]\!] = \Delta^{\rho_1}$ and $[\![\delta(\rho_2)]\!] = \Delta^{\rho_2}$. By definition of $[\![\delta(\rho_1 \to \rho_2)]\!]$ we have that

   (*) $([P], [Q]) \in [\![\delta(\rho_1 \to \rho_2)]\!]$ if and only if, for all M and N such that $([M], [N]) \in [\![\delta(\rho_1)]\!]$, $([P\,M], [Q\,N]) \in [\![\delta(\rho_2)]\!]$.

   We first show that

   (a) $([P], [Q]) \in [\![\delta(\rho_1 \to \rho_2)]\!]$ implies $P \simeq Q$, and

   (b) $P \simeq Q$ implies $([P], [Q]) \in [\![\delta(\rho_1 \to \rho_2)]\!]$.

   The implication (a) is proved by absurd. If $P \not\simeq Q$, then, by definition of bisimilarity there would be an M such that $P\,M \not\simeq Q\,M$. So (by induction hypothesis) $([P\,M], [Q\,M]) \notin [\![\delta(\rho_2)]\!]$, and therefore (from (*)) $([P], [Q]) \notin [\![\delta(\rho_1 \to \rho_2)]\!]$. For (b), let $P \simeq Q$. This implies that, for all R and S such that $R \simeq S$, $P\,R \simeq P\,S \simeq Q\,S \simeq Q\,R$. So (by induction hypothesis and (*)) $([P], [Q]) \in [\![\delta(\rho_1 \to \rho_2)]\!]$.

   From the equivalence between $\simeq$ and contextual equivalence (Theorem 2.6) we get the result.

$\rho = \rho_1 \times \rho_2$. Similar.

($\Longleftarrow$). Assume that $[\![\phi^\rho]\!] = \Delta^\rho$. From ($\Longrightarrow$) $\Delta^\rho = [\![\delta(\rho)]\!]$. So Theorem 4.7 implies $\phi^\rho \cong \delta(\rho)$, and Definition 4.4 implies $\phi^\rho = \delta(\rho)$.

□

5 Detecting and removing useless-code I

In this section we first introduce an e-type assignment system for detecting useless-code in PCFP programs (section 5.1) . Then we give a mapping for removing from a program the useless-code proved by using the e-type assignment system (section 5.2) and consider the problem of finding and removing *all* the useless-code that can be proved by the e-type assignment system (section 5.3).

The basic idea for using e-types to detect the useless-code in a given PCFP term M of type ρ is the following: if we can prove that (under suitable assumptions on its free variables) an application $NM_1 \cdots M_n$ has a δ-e-type while some of its arguments M_i's have ω-e-types, then (according to the e-type semantics) such M_i's are useless-code and can be removed without changing the semantics of the program. This remark can be easily generalized to any subexpression of an expression having a δ-e-type.

5.1 An e-type assignment system for detecting useless-code

If x^ρ is a term variable of type ρ an assumption for x^ρ is an expression of the shape $x^\rho : \phi^\rho$. A basis is a set Σ of e-types assumptions for term variables. E-types are assigned to PCFP terms by a set of inference rules for judgements of the form $\Sigma \vdash_1 M^\phi$ where Σ is a basis containing an assumption for each free variable of M^ϕ, and M^ϕ is a *decorated* term, that is, a term with (some of) the e-types assigned to its subterms written in it. Such a decorated term can then be processed by a transformation procedure (like the one described in section 5.2) that simplifies it.

Notation 5.1
For a basis Σ let $\epsilon(\Sigma)$ denote the set of term variables in Σ, and for a decorated term M^ϕ define $\epsilon(M^\phi)$ to be the term obtained from M^ϕ by erasing all the e-type decorations. Let $\Sigma, x : \psi$ denote the basis $\Sigma \cup \{x : \psi\}$ where it is assumed that x does not appear in Σ.

Definition 5.2 (*E-type assignment system* $\vdash_1$)
We write $\Sigma \vdash_1 M^\phi$ to mean that $\Sigma \vdash M^\phi$ can be derived by the rules in figure 6.

Note that, being $\vdash_1$ an inference system, a term can have different decorations. However, for every judgment $\Sigma \vdash_1 M^\psi$ there is exactly one derivation. Moreover, the only way ω-types can be derived is by using the rule (ω).

Remark 5.3
The system in figure 6 could be expressed in a slightly different way (equivalent to

$$(\omega)\ \frac{\epsilon(\Sigma) \supseteq \mathrm{FV}(M) \quad M \in \Lambda_\rho}{\Sigma \vdash M^{\omega^\rho}}$$

$$(\mathrm{Var})\ \frac{\phi \leqslant \psi}{\Sigma, x : \phi \vdash x^\psi}\ \psi \notin \mathbf{L}_\omega \qquad (\mathrm{Con})\ \Sigma \vdash \mathsf{k}^{\delta(\mathbf{T}(\mathsf{k}))}$$

$$(\to \mathrm{I})\ \frac{\Sigma, x : \phi \vdash M^\psi}{\Sigma \vdash (\lambda x.M)^{\phi \to \psi}}\ \psi \notin \mathbf{L}_\omega \qquad (\to \mathrm{E})\ \frac{\Sigma \vdash M^{\phi \to \psi} \quad \Sigma \vdash N^\phi}{\Sigma \vdash (MN^\phi)^\psi}$$

$$(\times \mathrm{I})\ \frac{\Sigma \vdash {M_1}^{\phi_1} \quad \Sigma \vdash {M_2}^{\phi_2}}{\Sigma \vdash \langle M_1, M_2 \rangle^{\phi_1 \times \phi_2}}\ \exists i \in \{1,2\}.\phi_i \notin \mathbf{L}_\omega$$

$$(\times \mathrm{E}_i)\ \frac{\Sigma \vdash M^{\phi_1 \times \phi_2}}{\Sigma \vdash (\mathsf{prj}_i M^{\phi_{3-i}})^{\phi_i}}\ \phi_i \notin \mathbf{L}_\omega$$

$$(\mathrm{Fix})\ \frac{\Sigma, x : \phi \vdash M^\phi \quad \phi \leqslant \psi}{\Sigma \vdash (\mathsf{fix}\, x.M^\phi)^\psi}\ \psi \notin \mathbf{L}_\omega$$

$$(\mathrm{If})\ \frac{\Sigma \vdash N^{\delta^{\mathsf{nat}}} \quad \Sigma \vdash {M_1}^\phi \quad \Sigma \vdash {M_2}^\phi}{\Sigma \vdash (\mathsf{ifz}\, N\, \mathsf{then}\, M_1\, \mathsf{else}\, M_2)^\phi}\ \phi \notin \mathbf{L}_\omega$$

Fig. 6. Rules for e-type assignment (system $\vdash_1$).

this one) by removing the use of entailment in rules (Var) and (Fix), and by adding an explicit entailment rule:

$$(\leqslant)\ \frac{\Sigma \vdash M^\phi \quad \phi \leqslant \psi}{\Sigma \vdash M^\psi}\ \psi \notin \mathbf{L}_\omega.$$

We chose the actual presentation since it is more suitable to the definition of the e-type inference algorithm (see section 7).

To state the soundness of the e-type assignment system we introduce the following definitions.

Definition 5.4

1. Two environments e_1, e_2 are Σ-*related* if and only if, for all $x : \psi \in \Sigma$, $(e_1(x), e_2(x)) \in [\![\psi]\!]$.
2. Let $\Sigma \vdash_1 M^\phi$ and $\Sigma \vdash_1 N^\phi$. We write $\epsilon(M^\phi) \sim_\phi^\Sigma \epsilon(N^\phi)$ to mean that for all e_1, e_2, if e_1 and e_2 are Σ-related, then $([\![\epsilon(M^\phi)]\!]_{e_1}, [\![\epsilon(N^\phi)]\!]_{e_2}) \in [\![\phi]\!]$.

Observe that, for any term P of type ρ with free variables $x_1^{\rho_1}$, …, $x_n^{\rho_n}$, if $P \sim_\chi^{\{x_1^{\rho_1} : \psi_1, \ldots, x_n^{\rho_n} : \psi_n\}} P$, then P satisfies (in the sense of Definition 3.1) the property $\mathscr{R} = [\![\chi]\!]$ under the assumptions $\mathscr{R}_i = [\![\psi_i]\!]$ for $x_i^{\rho_i}$ $(1 \leqslant i \leqslant n)$. Soundness of the system $\vdash_1$ can now be stated as follows.

Theorem 5.5 (*Soundness of* $\vdash_1$)
Let $\Sigma \vdash_1 P^\chi$. Then $\epsilon(P^\chi) \sim_\chi^\Sigma \epsilon(P^\chi)$.

Proof
By induction on the structure of derivations. We only present three cases.

- Let the derivation end with rule (Var). So $P = x$. Let e and e' be such that $(e(x), e'(x)) \in [\![\phi]\!]$. By Theorem 4.5 (soundness of $\leqslant$), $[\![\phi]\!] \subseteq [\![\chi]\!]$, so $(e(x), e'(x)) \in [\![\chi]\!]$ which proves the result.
- Let the derivation end with rule ($\to$ I). So $P = \lambda x.M$ and $\chi = \phi \to \psi$. We have to prove that for every e and e' which are Σ-related $([\![\lambda x.M]\!]_e, [\![\lambda x.M]\!]_{e'}) \in [\![\phi \to \psi]\!]$, i.e. $([(\lambda x.M)[\vec{y} := \vec{Q}]], [(\lambda x.M)[\vec{y} := \vec{Q}']]) \in [\![\phi \to \psi]\!]$, where $\vec{y} = y_1 \cdots y_k$ ($k \geqslant 0$) are the variables in Σ, $e(y_i) = [Q_i]$, $e'(y_i) = [Q_i']$, ϕ_i is the e-type of y_i in Σ, and $(e(y_i), e'(y_i)) \in [\![\phi_i]\!]$. By definition this is equivalent to show that, for all $([N], [N']) \in [\![\phi]\!]$, $([(\lambda x.M)[\vec{y} := \vec{Q}]N], [(\lambda x.M)[\vec{y} := \vec{Q}']N']) \in [\![\psi]\!]$, that is, since $\mathrm{FV}(N) = \mathrm{FV}(N') = \emptyset$, $([M[x := N, \vec{y} := \vec{Q}]], [M[x := N', \vec{y} := \vec{Q}']]) \in [\![\psi]\!]$, that is $([\![M]\!]_{e[x:=[N]]}, [\![M]\!]_{e'[x:=[N']]}) \in [\![\psi]\!]$. This follows directly by induction.
- Let the derivation end by with rule (Fix). So $P = \mathsf{fix}\, x.M$. We have to prove that $([\![\mathsf{fix}\, x.M]\!]_e, [\![\mathsf{fix}\, x.M]\!]_{e'}) \in [\![\phi]\!]$, i.e. $([(\mathsf{fix}\, x.M)[\vec{y} := \vec{Q}]], [(\mathsf{fix}\, x.M)[\vec{y} := \vec{Q}']]) \in [\![\phi]\!]$, where $\vec{y} = y_1 \cdots y_k$ ($k \geqslant 0$) are the variables in Σ, $e(y_i) = [Q_i]$, $e'(y_i) = [Q_i']$, ϕ_i is the e-type of y_i in Σ, and $(e(y_i), e'(y_i)) \in [\![\phi_i]\!]$. Since $\mathsf{fix}^{(0)}\, x.P = \mathsf{fix}\, z.z$ for any P, we have $([\mathsf{fix}^{(0)}\, x.M], [\mathsf{fix}^{(0)}\, x.M]) \in [\![\phi]\!]$. Assuming that $([\![\mathsf{fix}^{(n)}\, x.M]\!]_e, [\![\mathsf{fix}^{(n)}\, x.M]\!]_{e'}) \in [\![\phi]\!]$ and applying the induction hypothesis to the derivation $\Sigma, x : \phi \vdash_1 M : \phi$ we get $([\![M]\!]_{e[x:=[\![\mathsf{fix}^{(n)}\, x.M]\!]_e]}, [\![M]\!]_{e'[x:=[\![\mathsf{fix}^{(n)}\, x.M]\!]_{e'}]}) \in [\![\phi]\!]$, i.e. $([M[x := (\mathsf{fix}^{(n)}\, x.M)[\vec{y} := \vec{Q}], \vec{y} := \vec{Q}]], [M[x := (\mathsf{fix}^{(n)}\, x.M)[\vec{y} := \vec{Q}], \vec{y} := \vec{Q}]]) \in [\![\phi]\!]$, i.e. $([(\mathsf{fix}^{(n+1)}\, x.M)[\vec{y} := \vec{Q}]], [(\mathsf{fix}^{(n+1)}\, x.M)[\vec{y} := \vec{Q}]]) \in [\![\phi]\!]$, i.e. $([\![\mathsf{fix}^{(n+1)}\, x.M]\!]_e, [\![\mathsf{fix}^{(n+1)}\, x.M]\!]_{e'}) \in [\![\phi]\!]$. Then, by induction on n, we have that for all n, $([\![\mathsf{fix}^{(n)}\, x.M]\!]_e, [\![\mathsf{fix}^{(n)}\, x.M]\!]_{e'}) \in [\![\phi]\!]$. So we can conclude by Proposition 4.3 that $([\![\mathsf{fix}\, x.M]\!]_e, [\![\mathsf{fix}\, x.M]\!]_{e'}) \in [\![\phi]\!]$. Since $\phi \leqslant \psi$ then, by Theorem 4.5 (soundness of $\leqslant$), $[\![\phi]\!] \subseteq [\![\psi]\!]$ and therefore $([\![\mathsf{fix}\, x.M]\!]_e, [\![\mathsf{fix}\, x.M]\!]_{e'}) \in [\![\psi]\!]$.

□

The previous theorem implies that, given a term M, if we have a derivation $\Sigma \vdash_1 M'^{\chi}$ such that the subtem P of M is associated with the subderivation $\Sigma' \vdash_1 P'^{\phi}$, then if we replace P with a term $\sim_{\phi}^{\Sigma'}$-related to P we get a term $\sim_{\chi}^{\Sigma}$-related to M.

Example 5.6
Take the terms M, N, and R of Example 1.1:

- $M = (\lambda x^{\mathsf{nat}}.3)P$,
- $N = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle Q_1, Q_2\rangle)P$, where Q_1 does not contain any occurrence of x, and
- $R = (\lambda f^{\mathsf{nat}\to\mathsf{nat}}.fP)(\lambda x^{\mathsf{nat}}.3)$.

Assume that P, Q_1 and Q_2 are closed. It it easy to check that the following $\vdash_1$-typings hold:

- $\emptyset \vdash_1 ((\lambda x.3)P^{\omega^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$,
- $\emptyset \vdash_1 ((\lambda x.\mathsf{prj}_1\langle Q_1, Q_2\rangle^{\omega^{\mathsf{nat}}})P^{\omega^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$, and
- $\emptyset \vdash_1 ((\lambda f.fP^{\omega^{\mathsf{nat}}})(\lambda x.3)^{\omega^{\mathsf{nat}}\to\delta^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$.

This means that we can replace the subterms of M, N and R that have e-type ω^{nat} by with any other term of type nat without changing the meaning of M, N and P (which have e-type δ^{nat}).

5.2 A useless-code elimination

We now introduce a useless-code elimination mapping $\mathbf{O}_1$ that takes a $\vdash_1$-decorated term M^ϕ and returns a simplified version of it. The simplified version of the term has 'dummy variables' in place of subterms that are not needed for the evaluation of the term.

5.2.1 Dummy variables

For each type ρ, we consider a countable set $\{ \mathsf{d}^\rho, \mathsf{d}_1^\rho, \mathsf{d}_2^\rho, \dots \}$ of *dummy variables* of type ρ. We remark that dummy variables are not present in the original programs: they are introduced by the useless-code elimination mapping $\mathbf{O}_1$ as place-holders for the useless-code removed. In the following we assume that all the occurrences of dummy variables in a program are free and distinct[3].

Notation 5.7
For every term M, let $\mathrm{DV}(M)$ be the set of the dummy variables in M.

5.2.2 The simplification mapping

Let $\Lambda_{\mathbf{T}}^{\vdash_1}$ be the set of $\vdash_1$-decorated PCFP terms, i.e.

$$\Lambda_{\mathbf{T}}^{\vdash_1} = \{M^\phi \mid \Sigma \vdash_1 M^\phi \text{ for some e-type } \phi \text{ and basis } \Sigma\}.$$

Definition 5.8 (*Simplification mapping* $\mathbf{O}_1$)

1. The function
$$\mathbf{O}_1 : \Lambda_{\mathbf{T}}^{\vdash_1} \to \Lambda_{\mathbf{T}}^{\vdash_1}$$
is defined by the clauses in figure 7, where the occurrence of 'd' (second row) denotes a fresh dummy variable of the proper type.
2. If Σ is a basis then
$$\mathbf{O}_1(\Sigma) = \{x : \chi \mid x : \chi \in \Sigma \text{ and } \chi \notin \mathbf{L}_\omega\}.$$

Proposition 5.9 (*Correctness of* $\mathbf{O}_1$)
Let $\Sigma \vdash_1 M^\phi$, $\Sigma' = \mathbf{O}_1(\Sigma) \cup \{\mathsf{d}^\sigma : \omega^\sigma \mid \mathsf{d}^\sigma \in \mathrm{DV}(\epsilon(\mathbf{O}_1(M^\phi))\}$, and $\Sigma'' = \Sigma \cup \{\mathsf{d}^\sigma : \omega^\sigma \mid \mathsf{d}^\sigma \in \mathrm{DV}(\epsilon(\mathbf{O}_1(M^\phi))\}$. Then

1. $\Sigma' \vdash_1 \mathbf{O}_1(M^\phi)$, and
2. $\epsilon(M^\phi) \sim_\phi^{\Sigma''} \epsilon(\mathbf{O}_1(M^\phi))$.

[3] Dummy variables are simply a tool for proving the soundness of the simplification mapping $\mathbf{O}_1$: we will show that the value of a simplified term does not depend from the values assigned to its dummy variables. Indeed, since in a simplified program every dummy variable is useless-code, in a real implementation we can replace them with some polymorphic 'dummy constant' d.

$$
\begin{aligned}
\mathbf{O}_1(M^{\psi}) &= (\mathbf{o}_1(M,\psi))^{\psi}, \quad \text{where} \\
\mathbf{o}_1(M,\psi) &= \mathsf{d}, \quad \text{if } \psi \in \mathbf{L}_{\omega}\ . \quad \text{Otherwise:} \\
\mathbf{o}_1(\mathsf{k},\psi) &= \mathsf{k} \\
\mathbf{o}_1(x,\psi) &= x \\
\mathbf{o}_1(\langle M_1, M_2\rangle, \psi_1 \times \psi_2) &= \langle \mathbf{o}_1(M_1,\psi_1), \mathbf{o}_1(M_2,\psi_2)\rangle \\
\mathbf{o}_1(\mathsf{prj}_i M^{\psi_{3-i}}, \psi_i) &= \mathsf{prj}_i(\mathbf{o}_1(M, \psi_1 \times \psi_2))^{\psi_{3-i}} \quad \text{where } i \in \{1,2\} \\
\mathbf{o}_1(MN^{\phi}, \psi) &= \mathbf{o}_1(M, \phi \to \psi)(\mathbf{o}_1(N,\phi))^{\phi} \\
\mathbf{o}_1(\lambda x.M, \psi_1 \to \psi_2) &= \lambda x.\mathbf{o}_1(M,\psi_2) \\
\mathbf{o}_1(\mathsf{fix}\, x.M^{\phi}, \psi) &= \mathsf{fix}\, x.(\mathbf{o}_1(M,\phi))^{\psi} \\
\mathbf{o}_1(\mathsf{ifz}\, N\, \mathsf{then}\, M_1\, \mathsf{else}\, M_2, \psi) &= \mathsf{ifz}\, \mathbf{o}_1(N, \delta^{\mathsf{nat}})\, \mathsf{then}\, \mathbf{o}_1(M_1,\psi)\, \mathsf{else}\, \mathbf{o}_1(M_2,\psi)
\end{aligned}
$$

Fig. 7. Mapping $\mathbf{O}_1$ on terms.

Proof
Both (1) and (2) are by induction on the structure of the derivations, observing that $\Sigma \vdash_1 M^{\phi}$ implies both $\Sigma'' \vdash_1 M^{\phi}$ and $\Sigma'' \vdash_1 \mathbf{O}_1(M^{\phi})$. □

To use the simplification mapping $\mathbf{O}_1$ to simplify terms while preserving their meaning we identify a subset of $\vdash_1$-typings for which the $\sim_{\phi}^{\Sigma}$ relations implies the $\simeq_{\mathrm{obs}}$ relation.

Definition 5.10 (*Faithful $\vdash_1$-typing*)
$\Sigma \vdash_1 M^{\phi}$ is a *faithful* $\vdash_1$-*typing of* M if $\phi \in \mathbf{L}_{\delta}$, and for all $x : \psi \in \Sigma$, $\psi \in \mathbf{L}_{\omega} \cup \mathbf{L}_{\delta}$.

Example 5.11
The $\vdash_1$-typings of the terms M, N, and R in Example 5.6 are faithful. By applying the simplification mapping $\mathbf{O}_1$ we get the following $\vdash_1$-typings:

- $\{\mathsf{d} : \omega^{\mathsf{nat}}\} \vdash_1 ((\lambda x.3)\mathsf{d}^{\omega^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$,
- $\{\mathsf{d}_1 : \omega^{\mathsf{nat}},\ \mathsf{d}_2 : \omega^{\mathsf{nat}}\} \vdash_1 ((\lambda x.\mathsf{prj}_1\langle Q_1, \mathsf{d}_1\rangle^{\omega^{\mathsf{nat}}})\mathsf{d}_2^{\omega^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$, and
- $\{\mathsf{d} : \omega^{\mathsf{nat}}\} \vdash_1 ((\lambda f.f\mathsf{d}^{\omega^{\mathsf{nat}}})(\lambda x.3)^{\omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$.

Then, by erasing all the e-type decorations, we get the simplified terms:

- $M_1 = (\lambda x^{\mathsf{nat}}.3)\mathsf{d}^{\mathsf{nat}}$,
- $N_1 = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle Q_1, \mathsf{d}_1^{\mathsf{nat}}\rangle)\mathsf{d}_2^{\mathsf{nat}}$, and
- $R_1 = (\lambda f^{\mathsf{nat}\to\mathsf{nat}}.f\mathsf{d}^{\mathsf{nat}})(\lambda x^{\mathsf{nat}}.3)$.

The proof that the simplification performed by the mapping $\mathbf{O}_1$ on faithful decorated terms preserve $\simeq_{\text{obs}}$ relies on the following lemma.

Lemma 5.12
Let $\Sigma \vdash_1 M^\phi$ and $\Sigma \vdash_1 N^\phi$ be faithful $\vdash_1$-typings. Then $\epsilon(M^\phi) \sim_\phi^\Sigma \epsilon(N^\phi)$ implies $\epsilon(M^\phi) \simeq_{\text{obs}} \epsilon(N^\phi)$.

Proof
Let Σ' be the restriction of Σ to the variables $\text{FV}(M) \cup \text{FV}(N)$. Since $\Sigma \vdash_1 M^\phi$ is faithful, then for all $x : \psi \in \Sigma'$, $\psi \in \mathbf{L}_\omega \cup \mathbf{L}_\delta$. So for all e, e is Σ' related with e. Let now $\epsilon(M^\phi) \sim_\phi^{\Sigma'} \epsilon(N^\phi)$, then (by Definition 5.4.2) $([\![\epsilon(M^\phi)]\!]_e, [\![\epsilon(N^\phi)]\!]_e) \in [\![\phi]\!]$. Therefore, since $\phi \in \mathbf{L}_\delta$, from Proposition 4.9 we get for all e, $[\![\epsilon(M^\phi)]\!]_e = [\![\epsilon(N^\phi)]\!]_e$. Using the definition of $\simeq^{\mathbf{o}}$ and Theorem 2.6 we conclude that $\epsilon(M^\phi) \simeq_{\text{obs}} \epsilon(N^\phi)$. □

We can now show how to use the system $\vdash_1$ and the simplification mapping $\mathbf{O}_1$ to simplify terms producing observationally equivalent terms. In particular we show that all the dummy variables introduced by applying the mapping $\mathbf{O}_1$ on a faithful $\vdash_1$-typing are useless-code.

Theorem 5.13 ($\mathbf{O}_1$ *on faithful* $\vdash_1$*-typings preserves* $\simeq_{\text{obs}}$)
Let $\Sigma \vdash_1 M^\phi$ be a faithful $\vdash_1$-typing. Then $\epsilon(M^\phi) \simeq_{\text{obs}} \epsilon(\mathbf{O}_1(M^\phi))$.

Proof
Let $\Sigma' = \Sigma \cup \{\mathsf{d}^\sigma : \omega^\sigma \mid \mathsf{d}^\sigma \in \text{DV}(\epsilon(M^\phi))\}$. Since $\Sigma \vdash_1 M^\phi$ is a faithful $\vdash_1$-typing then also $\Sigma' \vdash_1 M^\phi$ and $\Sigma' \vdash_1 \mathbf{O}_1(M^\phi)$ are faithful $\vdash_1$-typings. So the result follows immediately from Proposition 5.9.2 and Lemma 5.12. □

Example 5.14
Let

$$M = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, gx\rangle, (\lambda y^{\mathsf{nat}}.1)P\rangle)(\lambda z^{\mathsf{nat}}.3)Q \in \Lambda_{\mathsf{nat}}$$

where $P, Q \in \Lambda_{\mathsf{nat}}$ are such that $\text{FV}(P) = \{u^{\mathsf{nat}}\}$ and $\text{FV}(Q) = \{v^{\mathsf{nat}}\}$. Then $\text{FV}(M) = \{f^{(\mathsf{nat}\to\mathsf{nat})\to\mathsf{nat}}, u^{\mathsf{nat}}, v^{\mathsf{nat}}\}$.

Take the faithful $\vdash_1$-typing $\Sigma' \vdash_1 M'$, where (writing δ and ω instead of δ^{nat} and ω^{nat})

$$\begin{array}{lcl}\Sigma' & = & \{f : (\delta\to\delta)\to\delta, u : \omega, v : \omega\}, \text{ and}\\ M' & = & ((\lambda g.\lambda x.+\langle +\langle fg^{\omega\to\delta}, gx^\omega\rangle^{\delta\times\delta}, (\lambda y.1)P^\omega\rangle^{\delta\times\delta})(\lambda z.3)^{\omega\to\delta}Q^\omega)^\delta.\end{array}$$

Applying the $\mathbf{O}_1$ simplification mapping we get $\mathbf{O}_1(\Sigma') \cup \{\mathsf{d}_1 : \omega, \mathsf{d}_2 : \omega, \mathsf{d}_3 : \omega\} \vdash_1 \mathbf{O}_1(M')$, where

$$\begin{array}{lcl}\mathbf{O}_1(\Sigma') & = & \{f : (\delta\to\delta)\to\delta\}, \text{ and}\\ \mathbf{O}_1(M') & = & ((\lambda g.\lambda x.+\langle +\langle fg^{\omega\to\delta}, g\mathsf{d}_1^\omega\rangle^{\delta\times\delta}, (\lambda y.1)\mathsf{d}_1^\omega\rangle^{\delta\times\delta})(\lambda z.3)^{\omega\to\delta}\mathsf{d}_1^\omega)^\delta.\end{array}$$

Let N_1 be the term obtained from $\mathbf{O}_1(M')$ by erasing the e-type decorations, i.e. $N_1 = \epsilon(\mathbf{O}_1(M'))$. We have

$$N_1 = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, g\mathsf{d}_1^{\mathsf{nat}}\rangle, (\lambda y^{\mathsf{nat}}.1)\mathsf{d}_2^{\mathsf{nat}}\rangle)(\lambda z^{\mathsf{nat}}.3)\mathsf{d}_3^{\mathsf{nat}}$$

with $\text{FV}(N_1) = \{f^{(\mathsf{nat}\to\mathsf{nat})\to\mathsf{nat}}, \mathsf{d}_1^{\mathsf{nat}}, \mathsf{d}_2^{\mathsf{nat}}, \mathsf{d}_3^{\mathsf{nat}}\}$.

$$\phi^\rho \sqsubseteq \omega^\rho$$

$$\delta^{\mathsf{nat}} \sqsubseteq \delta^{\mathsf{nat}}$$

$$\frac{\phi_1 \sqsubseteq \psi_1 \quad \phi_2 \sqsubseteq \psi_2}{\phi_1 \to \phi_2 \sqsubseteq \psi_1 \to \psi_2} \ \psi_2 \notin \mathbf{L}_\omega$$

$$\frac{\phi_1 \sqsubseteq \psi_1 \quad \phi_2 \sqsubseteq \psi_2}{\phi_1 \times \phi_2 \sqsubseteq \psi_1 \times \psi_2} \ \exists i \in \{1,2\}.\psi_i \notin \mathbf{L}_\omega$$

Fig. 8. Relation $\sqsubseteq$ for e-types.

5.3 Optimal faithful $\vdash_1$-typings

To detect all the useless-code of a term term M that can be proved useless in system $\vdash_1$, it is useful to see whether there is a faithful $\vdash_1$-typing of M that shows all the useless-code that is showed by some faithful $\vdash_1$-typing of M. If such an 'optimal' faithful $\vdash_1$-typing exists then the problem of detecting all the useless-code that can be proved by system $\vdash_1$ would be reduced to the problem of computing the 'optimal' faithful $\vdash_1$-typing.

Given two faithful $\vdash_1$-typings of M, $\Sigma \vdash_1 M^\phi$ and $\Sigma' \vdash_1 M'^\phi$, the one which allows the mapping $\mathbf{O}_1$ to remove more useless-code is the one that assigns ω-e-types to more subterms of M. This fact induces a partial order relation, $\sqsubseteq$, on the set of the $\vdash_1$-typings of M, defined as follows.

Definition 5.15 (*The $\sqsubseteq$ partial order*)

1. For basic properties the order $\sqsubseteq$ is defined by: $\delta \sqsubseteq \delta$, $\delta \sqsubseteq \omega$ and $\omega \sqsubseteq \omega$.
2. For e-types the partial order $\sqsubseteq$ is defined by the rules in figure 8[4].
3. For basis the partial order $\sqsubseteq$ is defined by: $\Sigma \sqsubseteq \Sigma'$ if for every $x^\rho : \psi \in \Sigma'$ there exists $x^\rho : \phi \in \Sigma$ such that $\phi \sqsubseteq \psi$.
4. For $\vdash_1$-decorated terms the partial order $\sqsubseteq$ is defined by the rules in figure 9.
5. For $\vdash_1$-typings the partial order $\sqsubseteq$ is defined by: $\Sigma \vdash_1 M^\phi \sqsubseteq \Sigma' \vdash_1 M'^\psi$ if $\Sigma \sqsubseteq \Sigma'$ and $M^\phi \sqsubseteq M'^\psi$.

It is straightforward to check that the relation $\sqsubseteq$ is a partial order.

Definition 5.16 (*Optimal faithful $\vdash_1$-typing*)
For every PCFP term M the *optimal faithful $\vdash_1$-typing of M* is the maximum element of the set

$$\{\Sigma \vdash_1 M' \mid \Sigma \vdash_1 M' \text{ is a faithful } \vdash_1\text{-typing of } M\} \ .$$

In section 7 we will prove (Theorem 7.14) that for every PCFP term M there exists the optimal faithful $\vdash_1$-typing of M. In particular we will give an algorithm that returns such a typing.

[4] The partial order $\sqsubseteq$ for e-types, which is covariant in both arguments of the constructor $\to$, should not be confused with the e-type entailment relation $\leqslant$, which is contravariant in the left argument of $\to$.

$$M^{\phi} \sqsubseteq \epsilon(M)^{\omega^{\epsilon(\phi)}}$$

$$\frac{\phi \sqsubseteq \psi}{x^{\phi} \sqsubseteq x^{\psi}} \ \psi \notin \mathbf{L}_{\omega}$$

$$\mathsf{k}^{\delta(\mathbf{T}(\mathsf{k}))} \sqsubseteq \mathsf{k}^{\delta(\mathbf{T}(\mathsf{k}))}$$

$$\frac{\phi_1 \sqsubseteq \psi_1 \quad M^{\phi_2} \sqsubseteq N^{\psi_2}}{(\lambda x.M)^{\phi_1 \to \phi_2} \sqsubseteq (\lambda x.N)^{\psi_1 \to \psi_2}} \ \psi_2 \notin \mathbf{L}_{\omega}$$

$$\frac{F^{\phi_1 \to \phi_2} \sqsubseteq G^{\psi_1 \to \psi_2} \quad M^{\phi_1} \sqsubseteq N^{\psi_1}}{(FM^{\phi_1})^{\phi_2} \sqsubseteq (GN^{\psi_1})^{\psi_2}}$$

$$\frac{M_1^{\phi_1} \sqsubseteq N_1^{\psi_1} \quad M_2^{\phi_2} \sqsubseteq N_2^{\psi_2}}{\langle M_1, M_2 \rangle^{\phi_1 \times \phi_2} \sqsubseteq \langle N_1, N_2 \rangle^{\psi_1 \times \psi_2}} \ \exists i \in \{1,2\}.\psi_i \notin \mathbf{L}_{\omega}$$

$$\frac{M^{\phi_1 \times \phi_2} \sqsubseteq N^{\psi_1 \times \psi_2}}{(\mathsf{prj}_i M^{\phi_{3-i}})^{\phi_i} \sqsubseteq (\mathsf{prj}_i N^{\psi_{3-i}})^{\psi_i}} \ i \in \{1,2\} \text{ and } \psi_i \notin \mathbf{L}_{\omega}$$

$$\frac{M^{\phi_1} \sqsubseteq N^{\psi_1} \quad \phi_2 \sqsubseteq \psi_2}{(\mathsf{fix}\, x.M^{\phi_1})^{\phi_2} \sqsubseteq (\mathsf{fix}\, x.N^{\psi_1})^{\psi_2}} \ \psi_1, \psi_2 \notin \mathbf{L}_{\omega}$$

$$\frac{B^{\delta^{\mathsf{nat}}} \sqsubseteq C^{\delta^{\mathsf{nat}}} \quad M_1^{\phi} \sqsubseteq N_1^{\psi} \quad M_2^{\phi} \sqsubseteq N_2^{\psi}}{(\mathsf{ifz}\, B \,\mathsf{then}\, M_1 \,\mathsf{else} M_2)^{\phi} \sqsubseteq (\mathsf{ifz}\, C \,\mathsf{then}\, N_1 \,\mathsf{else} N_2)^{\psi}} \ \psi \notin \mathbf{L}_{\omega}$$

Fig. 9. Relation $\sqsubseteq$ for decorated terms.

6 Detecting and removing useless-code II

We now introduce a useless-code type assignment, $\vdash_2$, which shows less useless-code than $\vdash_1$. In section 6.2, we will see that the system $\vdash_2$ induces a more effective simplification mapping, which transforms not only the code of a program but also the type of its subexpressions. In particular, the simplification does not introduce dummy variables. In Section 6.3 we show that system $\vdash_2$ can be used to remove some of the dummy variables introduced by $\mathbf{O}_1$.

6.1 A weaker e-type assignment system for detecting useless-code

Definition 6.1 (*E-type assignment system* $\vdash_2$)
We write $\Sigma \vdash_2 M^{\phi}$ to mean that $\Sigma \vdash M^{\phi}$ can be derived by the rules in figure 6 in which the rules (Var) and (Fix) are replaced by the following:

$$\text{(Var) } \Sigma, x : \phi \vdash x^{\phi} \quad \phi \notin \mathbf{L}_{\omega} \qquad\qquad \text{(Fix) } \frac{\Sigma, x : \phi \vdash M^{\phi}}{\Sigma \vdash (\mathsf{fix}\, x.M^{\phi})^{\phi}} \ \phi \notin \mathbf{L}_{\omega} \ .$$

Also in this case, for every judgment $\Sigma \vdash_2 M^{\psi}$ there is exactly one derivation.

Example 6.2
Take the terms M, N and R of Example 5.6:

$$
\begin{array}{rcl}
\mathbf{O}_2(\delta^{\mathsf{nat}}) & = & \delta^{\mathsf{nat}} \\
\mathbf{O}_2(\phi \to \psi) & = & \begin{cases} \mathbf{O}_2(\phi) \to \mathbf{O}_2(\psi) & \text{if } \phi \notin \mathbf{L}_\omega \\ \mathbf{O}_2(\psi) & \text{if } \phi \in \mathbf{L}_\omega \end{cases} \\
\mathbf{O}_2(\phi \times \psi) & = & \begin{cases} \mathbf{O}_2(\phi) \times \mathbf{O}_2(\psi) & \text{if } \phi, \psi \notin \mathbf{L}_\omega \\ \mathbf{O}_2(\phi) & \text{if } \psi \in \mathbf{L}_\omega \\ \mathbf{O}_2(\psi) & \text{if } \phi \in \mathbf{L}_\omega \end{cases}
\end{array}
$$

Fig. 10. Mapping $\mathbf{O}_2$ on types.

- $M = (\lambda x^{\mathsf{nat}}.3)P$,
- $N = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle Q_1, Q_2\rangle)P$, where Q_1 does not contain any occurrence of x, and
- $R = (\lambda f^{\mathsf{nat}\to\mathsf{nat}}.fP)(\lambda x^{\mathsf{nat}}.3)$.

Assume that P, Q_1 and Q_2 are closed. It it easy to check that the two $\vdash_1$-typings of Example 5.6 are also $\vdash_2$-typings, i.e.

- $\emptyset \vdash_2 ((\lambda x.3)P^{\omega^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$,
- $\emptyset \vdash_2 ((\lambda x.\mathsf{prj}_1\langle Q_1, Q_2\rangle^{\omega^{\mathsf{nat}}})P^{\omega^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$, and
- $\emptyset \vdash_2 ((\lambda f.fP^{\omega^{\mathsf{nat}}})(\lambda x.3)^{\omega^{\mathsf{nat}}\to\delta^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$.

The following theorem states the soundness of the system $\vdash_2$.

Theorem 6.3 (*Soundness of* $\vdash_2$)
Let $\Sigma \vdash_2 M^\phi$. Then $\epsilon(M^\phi) \sim_\phi^\Sigma \epsilon(M^\phi)$.

Proof
By Theorem 5.5, since $\Sigma \vdash_2 M^\phi$ implies $\Sigma \vdash_1 M^\phi$. □

6.2 A more effective useless-code elimination

In this section, following Berardi (1996), we introduce a simplification mapping $\mathbf{O}_2$ that, given a $\vdash_2$-decorated term M^ϕ with $\phi \notin \mathbf{L}_\omega$, returns an optimized version of it. The simplification performed by $\mathbf{O}_2$, which is essentially a useless-variable elimination, can be much stronger than the simplification performed by $\mathbf{O}_1$. For instance, $\mathbf{O}_2$ transforms an application like $((\lambda x.M)N^{\omega^\rho})^{\delta(\sigma)}$, in which the function $\lambda x.M$ does not use its argument, in the body of the applied function $M^{\delta(\sigma)}$.

Since $\mathbf{O}_2$ introduces a deep change in the structure of the term we need to define it also on types and not only on terms.

6.2.1 The mapping on types

Definition 6.4 (*Simplification mapping* $\mathbf{O}_2$ *on types*)
The function $\mathbf{O}_2 : (\mathbf{L} - \mathbf{L}_\omega) \to (\mathbf{L} - \mathbf{L}_\omega)$ is defined by the clauses in Fig. 10.

$$\mathbf{I}_{\delta^{\mathsf{nat}}} = \lambda x^{\mathsf{nat}}.x$$

$$\mathbf{J}_{\delta^{\mathsf{nat}}} = \lambda x^{\mathsf{nat}}.x$$

$$\mathbf{I}_{\phi\to\psi} = \begin{cases} \lambda f^{\epsilon(\phi\to\psi)}.\lambda z^{\epsilon(\mathbf{O}_2(\phi))}.\mathbf{I}_\psi(f(\mathbf{J}_\phi z)) & \text{if } \phi \notin \mathbf{L}_\omega \\ \lambda f^{\epsilon(\phi\to\psi)}.\mathbf{I}_\psi(f\mathsf{D}^{\epsilon(\phi)}) & \text{if } \phi \in \mathbf{L}_\omega \end{cases}$$

$$\mathbf{J}_{\phi\to\psi} = \begin{cases} \lambda f^{\epsilon(\mathbf{O}_2(\phi)\to\mathbf{O}_2(\psi))}.\lambda z^{\epsilon(\phi)}.\mathbf{J}_\psi(f(\mathbf{I}_\phi z)) & \text{if } \phi \notin \mathbf{L}_\omega \\ \lambda y^{\epsilon(\mathbf{O}_2(\psi))}.\lambda z^{\epsilon(\phi)}.\mathbf{J}_\psi y & \text{if } \phi \in \mathbf{L}_\omega \end{cases}$$

$$\mathbf{I}_{\phi_1\times\phi_2} = \begin{cases} \lambda z^{\epsilon(\phi_1\times\phi_2)}.\langle \mathbf{I}_{\phi_1}(\mathsf{prj}_1 z), \mathbf{I}_{\phi_2}(\mathsf{prj}_2 z)\rangle & \text{if } \phi_1,\phi_2 \notin \mathbf{L}_\omega \\ \lambda z^{\epsilon(\phi_1\times\phi_2)}.\mathbf{I}_{\phi_1}(\mathsf{prj}_1 z) & \text{if } \phi_2 \in \mathbf{L}_\omega \\ \lambda z^{\epsilon(\phi_1\times\phi_2)}.\mathbf{I}_{\phi_2}(\mathsf{prj}_2 z) & \text{if } \phi_1 \in \mathbf{L}_\omega \end{cases}$$

$$\mathbf{J}_{\phi_1\times\phi_2} = \begin{cases} \lambda z^{\epsilon(\mathbf{O}_2(\phi_1)\times\mathbf{O}_2(\phi_2))}.\langle \mathbf{J}_{\phi_1}(\mathsf{prj}_1 z), \mathbf{J}_{\phi_2}(\mathsf{prj}_2 z)\rangle & \text{if } \phi_1,\phi_2 \notin \mathbf{L}_\omega \\ \lambda z^{\epsilon(\mathbf{O}_2(\phi_1))}.\langle \mathbf{J}_{\phi_1} z, \mathsf{D}^{\epsilon(\phi_2)}\rangle & \text{if } \phi_2 \in \mathbf{L}_\omega \\ \lambda z^{\epsilon(\mathbf{O}_2(\phi_2))}.\langle \mathsf{D}^{\epsilon(\phi_1)}, \mathbf{J}_{\phi_2} z\rangle & \text{if } \phi_1 \in \mathbf{L}_\omega \end{cases}$$

Fig. 11. Mappings $\mathbf{I}_\cdot$ and $\mathbf{J}_\cdot$.

Note that, in general, $\epsilon(\mathbf{O}_2(\phi)) \neq \epsilon(\phi)$. For any e-type $\phi \notin \mathbf{L}_\omega$ there is a (PCFP-definable) isomorphism between $\mathbf{I}(\epsilon(\phi))$ (the interpretation of $\epsilon(\phi)$ in $\mathscr{M}$) and $\mathbf{I}(\epsilon(\mathbf{O}_2(\phi)))$ which is defined by a pair of PCFP terms. The basic idea is to map a constant function which has e-type $\omega^\rho \to \delta(\sigma)$ in a term of e-type $\delta(\sigma)$, and vice-versa. This idea is lifted to e-types with more involved structure in a standard way.

Definition 6.5
For $\chi \in \mathbf{L} - \mathbf{L}_\omega$, the closed PCFP terms $\mathbf{I}_\chi \in \Lambda_{\epsilon(\chi)\to\epsilon(\mathbf{O}_2(\chi))}$ and $\mathbf{J}_\chi \in \Lambda_{\epsilon(\mathbf{O}_2(\chi))\to\epsilon(\chi)}$ are defined by induction on χ according to the clauses in figure 11 where every occurrence of D denotes a closed term of the proper type.

The basic properties of these mappings are expressed by the following lemma that can be proved by induction on e-types.

Lemma 6.6

1. For all $\chi \in \mathbf{L} - \mathbf{L}_\omega$,
   (a) $[\lambda x^{\epsilon(\mathbf{O}_2(\chi))}.\mathbf{I}_\chi(\mathbf{J}_\chi x)] = [\lambda x^{\epsilon(\mathbf{O}_2(\chi))}.x]$, and
   (b) $[\lambda x^{\epsilon(\chi)}.\mathbf{J}_\chi(\mathbf{I}_\chi x)] = [\lambda x^{\epsilon(\chi)}.x]$.
2. Let $\phi,\psi \notin \mathbf{L}_\omega$, and M, N be closed terms such that $([M],[M]) \in [\![\mathbf{O}_2(\phi\to\psi)]\!]$ and $([N],[N]) \in [\![\mathbf{O}_2(\phi)]\!]$. Then $[\mathbf{J}_\psi(MN)] = [(\mathbf{J}_{\phi\to\psi}(M))(\mathbf{J}_\phi N)]$.
3. Let $\phi_1,\phi_2 \notin \mathbf{L}_\omega$, and M be a closed term such that $([M],[M]) \in [\![\mathbf{O}_2(\phi_1\times\phi_2)]\!]$. Then, for $i \in \{1,2\}$, $[\mathbf{J}_{\phi_i}(\mathsf{prj}_i M)] = [\mathsf{prj}_i(\mathbf{J}_{\phi_1\times\phi_2}(M))]$.
4. Let $\phi \in \mathbf{L}_\delta$. Then $[\mathbf{J}_\phi] = [\mathbf{I}_\phi] = [\lambda x^{\epsilon(\phi)}.x]$.

$$\mathbf{O}_2(M^{\psi}) = \mathbf{o}_2(M, \psi) \quad \text{where}$$

$$\begin{aligned}
\mathbf{o}_2(\mathsf{k}, \psi) &= \mathsf{k} \\
\mathbf{o}_2(x, \psi) &= \mathbf{o}(x) \\
\mathbf{o}_2(\langle M_1, M_2\rangle, \psi_1 \times \psi_2) &= \begin{cases} \langle \mathbf{o}_2(M_1, \psi_1), \mathbf{o}_2(M_2, \psi_2)\rangle & \text{if } \psi_1, \psi_2 \notin \mathbf{L}_\omega \\ \mathbf{o}_2(M_1, \psi_1) & \text{if } \psi_2 \in \mathbf{L}_\omega \\ \mathbf{o}_2(M_2, \psi_2) & \text{if } \psi_1 \in \mathbf{L}_\omega \end{cases} \\
\mathbf{o}_2(\mathsf{prj}_i M^{\psi_{3-i}}, \psi_i) &= \begin{cases} \mathsf{prj}_i(\mathbf{o}_2(M, \psi_1 \times \psi_2))^{\mathbf{O}_2(\psi_{3-i})} & \text{if } \psi_1, \psi_2 \notin \mathbf{L}_\omega \\ \mathbf{o}_2(M, \psi_1 \times \psi_2) & \text{if } \psi_{(3-i)} \in \mathbf{L}_\omega \end{cases} \\
&\quad \text{where } i \in \{1, 2\} \\
\mathbf{o}_2(MN^{\phi}, \psi) &= \begin{cases} \mathbf{o}_2(M, \phi \to \psi)(\mathbf{o}_2(N, \phi))^{\mathbf{O}_2(\phi)} & \text{if } \phi \notin \mathbf{L}_\omega \\ \mathbf{o}_2(M, \phi \to \psi) & \text{if } \phi \in \mathbf{L}_\omega \end{cases} \\
\mathbf{o}_2(\lambda x.M, \psi_1 \to \psi_2)) &= \begin{cases} \lambda \mathbf{o}(x).\mathbf{o}_2(M, \psi_2) & \text{if } \psi_1 \notin \mathbf{L}_\omega \\ \mathbf{o}_2(M, \psi_2) & \text{if } \psi_1 \in \mathbf{L}_\omega \end{cases} \\
\mathbf{o}_2(\mathsf{fix}\, x.M, \phi)) &= \mathsf{fix}\, \mathbf{o}(x).(\mathbf{o}_2(M, \phi))^{\mathbf{O}_2(\phi)} \\
\mathbf{o}_2(\mathsf{ifz}\, N\, \mathsf{then}\, M_1\, \mathsf{else}\, M_2, \psi) &= \mathsf{ifz}\, \mathbf{o}_2(N, \delta^{\mathsf{nat}})\, \mathsf{then}\, \mathbf{o}_2(M_1, \psi)\, \mathsf{else}\, \mathbf{o}_2(M_2, \psi)
\end{aligned}$$

Fig. 12. Mapping $\mathbf{O}_2$ on terms.

6.2.2 *The mapping on terms*

We now define the simplification mapping $\mathbf{O}_2$ on terms. Note that the mapping $\mathbf{O}_2$ may modify the type of the term and of its subterms.

Let $\Lambda_{\mathbf{T}}^{\vdash_2}$ be the set of the $\vdash_2$ decorated terms, i.e.

$$\Lambda_{\mathbf{T}}^{\vdash_2} = \{M^{\phi} \mid \Sigma \vdash_2 M^{\phi} \text{ for some e-type } \phi \text{ and basis } \Sigma\}.$$

In the following definition we assume that all free and bound variables of a term M have different names, and that there is a mapping $\mathbf{o}$ which maps variables of type $\epsilon(\phi)$ in variables of type $\epsilon(\mathbf{O}_2(\phi))$. We assume also that the domain and the range of $\mathbf{o}$ are disjoint.

Definition 6.7 (*Simplification mapping* $\mathbf{O}_2$ *on terms*)

1. The function

$$\mathbf{O}_2 : \{M^{\phi} \mid M^{\phi} \in \Lambda_{\mathbf{T}}^{\vdash_2} \text{ and } \phi \notin \mathbf{L}_\omega\} \to \{M^{\phi} \mid M^{\phi} \in \Lambda_{\mathbf{T}}^{\vdash_2} \text{ and } \phi \notin \mathbf{L}_\omega\}$$

   is defined by the clauses in Fig. 12.

2. If Σ is a basis then

$$\mathbf{O}_2(\Sigma) = \{\mathbf{o}(x) : \mathbf{O}_2(\psi) \mid x : \psi \in \Sigma \text{ and } \psi \notin \mathbf{L}_\omega\}.$$

Notice that, for every $\Sigma \vdash_2 M^{\phi}$, in $\mathbf{O}_2(\Sigma)$ and $\mathbf{O}_2(M^{\phi})$ all the variables that have

an ω-e-type in Σ have been removed by the simplification. The mapping $\mathbf{O}_2$ is correct.

Proposition 6.8 (*Correctness of* $\mathbf{O}_2$)
Let $\Sigma \vdash_2 M^\phi$ (where $\phi \notin \mathbf{L}_\omega$). Then

1. $\mathbf{O}_2(\Sigma) \vdash_2 \mathbf{O}_2(M^\phi)$, and
2. for all environments e_1, e_2 such that $x{:}\psi \in \Sigma$ and $\psi \notin \mathbf{L}_\omega$ implies $(e_1(x), [\mathbf{J}_\psi N]) \in [\![\psi]\!]$ (where $[N] = e_2(\mathbf{o}(x))$), we have that: $([\![M]\!]_{e_1}, [\![\mathbf{J}_\phi \epsilon(\mathbf{O}_2(M^\phi))]\!]_{e_2}) \in [\![\phi]\!]$.

Proof
Both (1) and (2) are by induction on derivations. In particular the proof of (2) uses Lemma 6.6. □

Faithful $\vdash_2$-typings are defined in the same way of faithful $\vdash_1$-typings (see Definition 5.10).

Example 6.9
The $\vdash_2$-typings of the terms M, N, and R in Example 6.2 are faithful. By applying the simplification mapping $\mathbf{O}_2$ we get the following $\vdash_2$-typings:

- $\emptyset \vdash_2 \mathsf{3}^{\delta^{\mathsf{nat}}}$,
- $\emptyset \vdash_2 Q_1^{\delta^{\mathsf{nat}}}$, and
- $\emptyset \vdash_2 ((\lambda f.f)\mathsf{3}^{\delta^{\mathsf{nat}}})^{\delta^{\mathsf{nat}}}$.

Then, by erasing all the e-type decorations, we get the simplified terms:

- $M_1 = \mathsf{3}$,
- $N_1 = Q_1$, and
- $R_1 = (\lambda f^{\mathsf{nat}}.f)\mathsf{3}$.

Note that, if $\Sigma \vdash_2 M^\phi$ is a faithful $\vdash_2$-typing, then $\epsilon(\mathbf{O}_2(\phi)) = \epsilon(\phi)$, $\mathbf{O}_2(\Sigma) \subseteq \Sigma$, and $[\mathbf{J}_\phi \epsilon(\mathbf{O}_2(M^\phi))] = [\epsilon(\mathbf{O}_2(M^\phi))]$. We can now prove that if $\Sigma \vdash_2 M^\phi$ is a faithful $\vdash_2$-type assignment then $\epsilon(M^\phi)$ and $\epsilon(\mathbf{O}_2(M^\phi))$ are observationally equivalent.

Theorem 6.10 ($\mathbf{O}_2$ *on faithful* $\vdash_2$*-typings preserves* $\simeq_{\mathrm{obs}}$)
Let $\Sigma \vdash_2 M^\phi$ be a faithful $\vdash_2$-typing. Then $\epsilon(M^\phi) \simeq_{\mathrm{obs}} \epsilon(\mathbf{O}_2(M^\phi))$.

Proof
Since $\Sigma \vdash_2 M^\phi$ is a faithful $\vdash_2$-typing then also $\Sigma \vdash_2 \mathbf{O}_2(M^\phi)$ is a faithful $\vdash_2$-typing. Therefore for all environments e, e is Σ related to itself. From Proposition 6.8, and the fact that $[\mathbf{J}_\phi \epsilon(\mathbf{O}_2(M^\phi))] = [\epsilon(\mathbf{O}_2(M^\phi))]$ which follows from 6.6.4, we derive that $([\![M]\!]_e, [\![\epsilon(\mathbf{O}_2(M^\phi))]\!]_e) \in [\![\phi]\!]$. So from Proposition 4.9 and $\phi \in \mathbf{L}_\delta$, we get that for all environments e, $[\![M]\!]_e \simeq_{\mathrm{obs}} [\![\epsilon(\mathbf{O}_2(M^\phi))]\!]_e$. Using the definition of $\simeq^{\mathbf{o}}$ and Theorem 2.6 we conclude that $\epsilon(M^\phi) \simeq_{\mathrm{obs}} \epsilon(\mathbf{O}_2(M^\phi))$. □

Example 6.11
Let

$$M = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, gx\rangle, (\lambda y^{\mathsf{nat}}.\mathsf{1})P\rangle)(\lambda z^{\mathsf{nat}}.\mathsf{3})Q$$

be the term of Example 5.14. Take the faithful $\vdash_2$-typing $\Sigma'' \vdash_2 M''$, where (writing δ and ω instead of δ^{nat} and ω^{nat})

$$\begin{array}{lcl} \Sigma'' & = & \{f : (\delta \to \delta) \to \delta, u : \omega, v : \delta\}, \text{ and} \\ M'' & = & ((\lambda g.\lambda x. + \langle +\langle f g^{\delta\to\delta}, g x^{\delta}\rangle^{\delta\times\delta}, (\lambda y.1) P^{\omega}\rangle^{\delta\times\delta})(\lambda z.3)^{\delta\to\delta} Q^{\delta})^{\delta} \ . \end{array}$$

Applying the $\mathbf{O}_2$ simplification mapping we get $\mathbf{O}_2(\Sigma'') \vdash_2 \mathbf{O}_2(M'')$, where

$$\begin{array}{lcl} \mathbf{O}_2(\Sigma'') & = & \{f : (\delta \to \delta) \to \delta, v : \delta\}, \text{ and} \\ \mathbf{O}_2(M'') & = & ((\lambda g.\lambda x. + \langle +\langle f g^{\delta\to\delta}, g x^{\delta}\rangle^{\delta\times\delta}, 1\rangle^{\delta\times\delta})(\lambda z.3)^{\delta\to\delta} Q^{\delta})^{\delta} \ . \end{array}$$

Let N_2 be the term obtained from $\mathbf{O}_2(M'')$ by erasing the e-type decorations, i.e., $N_2 = \epsilon(\mathbf{O}_2(M''))$. We have

$$N_2 = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}. + \langle +\langle f g, g x\rangle, 1\rangle)(\lambda z^{\mathsf{nat}}.3) Q$$

with $\mathrm{FV}(N_2) = \{f^{(\mathsf{nat}\to\mathsf{nat})\to\mathsf{nat}}, v^{\mathsf{nat}}\}$.

Optimal faithful $\vdash_2$-typings are defined like optimal faithful $\vdash_1$-typings (see section 5.3). In section 7 we will give an algorithm that for every PCFP term M returns the optimal faithful $\vdash_2$-typing of M.

6.3 *Combined use of* $\mathbf{O}_1$ *and* $\mathbf{O}_2$

In this section we show that the combined use of the simplification mappings $\mathbf{O}_1$ and $\mathbf{O}_2$ improves the useless-code elimination performed by either one of them.

Example 6.12
Let

$$M = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}. + \langle +\langle f g, g x\rangle, (\lambda y^{\mathsf{nat}}.1) P\rangle)(\lambda z^{\mathsf{nat}}.3) Q$$

be the term of Example 5.14.

Let N_1 be the term obtained from M as shown in Example 5.14, i.e. by

- decorating M using the system $\vdash_1$,
- applying the simplification mapping $\mathbf{O}_1$, and
- erasing the e-type decorations.

We have that

$$N_1 = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}. + \langle +\langle f g, g \mathsf{d}_1^{\mathsf{nat}}\rangle, (\lambda y^{\mathsf{nat}}.1)\mathsf{d}_2^{\mathsf{nat}}\rangle)(\lambda z^{\mathsf{nat}}.3)\mathsf{d}_3^{\mathsf{nat}}$$

with $\mathrm{FV}(N_1) = \{f^{(\mathsf{nat}\to\mathsf{nat})\to\mathsf{nat}}, \mathsf{d}_1{}^{\mathsf{nat}}, \mathsf{d}_2{}^{\mathsf{nat}}, \mathsf{d}_3{}^{\mathsf{nat}}\}$.

Take the faithful $\vdash_2$-typing $\Sigma \vdash_2 N_1'$, where

$$\begin{array}{lcl} \Sigma & = & \{f : (\delta \to \delta) \to \delta, \mathsf{d}_1 : \omega, \mathsf{d}_2 : \omega, \mathsf{d}_3 : \omega\}, \text{ and} \\ N_1' & = & ((\lambda g.\lambda x. + \langle +\langle f g^{\delta\to\delta}, g \mathsf{d}_1^{\delta}\rangle^{\delta\times\delta}, (\lambda y.1)\mathsf{d}_2^{\omega}\rangle^{\delta\times\delta})(\lambda z^{\mathsf{nat}}.3)^{\delta\to\delta}\mathsf{d}_3^{\omega})^{\delta}. \end{array}$$

Applying the $\mathbf{O}_2$ simplification mapping we get $\mathbf{O}_2(\Sigma) \vdash_2 \mathbf{O}_2(N_1')$, where

$$\begin{array}{lcl} \mathbf{O}_2(\Sigma) & = & \{f : (\delta \to \delta) \to \delta, \mathsf{d}_1 : \omega\}, \text{ and} \\ \mathbf{O}_2(N_1') & = & ((\lambda g. + \langle +\langle f g^{\delta\to\delta}, g \mathsf{d}_1^{\delta}\rangle^{\delta\times\delta}, 1\rangle^{\delta\times\delta})(\lambda z^{\mathsf{nat}}.3)^{\delta\to\delta})^{\delta}. \end{array}$$

Let N be the term obtained from $\mathbf{O}_2(N_1')$ by erasing the e-type decorations, i.e. $N = \epsilon(N_1')$. We have that

$$N = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.+\langle +\langle fg, g\mathsf{d}_1^{\mathsf{nat}}\rangle, 1\rangle)(\lambda z^{\mathsf{nat}}.3)$$

with $\mathrm{FV}(N) = \{f^{(\mathsf{nat}\to\mathsf{nat})\to\mathsf{nat}}, \mathsf{d}_1{}^{\mathsf{nat}}\}$.

Comparing N with the term N_1 above and with the term N_2 of Example 6.11 we note that this combined use of the simplification mappings $\mathbf{O}_1$ and $\mathbf{O}_2$ improves the useless-code elimination performed by each one of them.

We remark that all the faithful typings considered in Example 6.12 are optimal faithful typings w.r.t. the corresponding e-type assignment system.

7 Computing optimal faithful typings

In this section we prove that for every PCFP term M there exists both the optimal faithful $\vdash_1$-typing and $\vdash_2$-typing. This result is proved by giving algorithms which compute such typings. We first introduce e-type schemes which are meant to represent sets of e-types, then we introduce the inference algorithm that given a term returns a decorated version of the term and an e-type scheme representing the set of e-types that can be assigned to the term.

7.1 Evaluation type schemes

In this section we first introduce the notion of e-pattern and e-constraint. Then we define e-type schemes as pairs of e-patterns and sets of e-constraints.

Definition 7.1 (*Evaluation type patterns*)
Let $\mathscr{A}$ be the set of *basic property variables* (*basic variables* for short), ranged by α, β, γ, ... The language $\mathbf{P}$ of *e-type patterns* (*e-patterns* for short), ranged over by θ, η, ..., is defined by the following grammar: $\theta ::= \alpha^{\mathsf{nat}} \mid \theta \to \theta \mid \theta \times \theta$, where $\alpha \in \mathscr{A}$.

A basic variable α can be *replaced* by another basic variable or *instantiated* to a basic property (δ or ω).

Definition 7.2 (*Replacements and instantiations*)

1. A *replacement* is a mapping $\mathbf{r} : \mathscr{A} \to \mathscr{A}$. A one-to-one replacement is called a *renaming*.
   Let $\alpha_1, \ldots, \alpha_n$ ($n \geqslant 0$) be basic variables such that $i \neq j$ implies $\alpha_i \neq \alpha_j$. As usual the expression $[\alpha_1 := \beta_1, \ldots, \alpha_n := \beta_n]$ denotes the replacement $\mathbf{r}$ which behaves as described on $\alpha_1, \ldots, \alpha_n$ and is the identity elsewhere.
2. An *instantiation* is a mapping $\mathbf{i} : \mathscr{A} \to \{\delta, \omega\}$.

Replacements and instantiations are extended to basic properties by defining $\mathbf{i}(a) = a$ and $\mathbf{r}(a) = a$, for $a \in \{\delta, \omega\}$.

The order relation $\sqsubseteq$ for basic properties (Definition 5.15.1) is such that: $\delta \sqsubseteq \delta$, $\delta \sqsubseteq \omega$ and $\omega \sqsubseteq \omega$. Let the symbol $\equiv$ denote the equivalence relation induced by $\sqsubseteq$. An e-constraint is an equality or an inequality (between basic variables and/or the basic property δ) or a guarded set of e-constraints.

Definition 7.3 (*E-constraints*)
An *e-constraint* is a formula of one of the following shapes:

- $\zeta_1 \equiv \zeta_2$
- $\zeta_1 \sqsubseteq \zeta_2$
- $(\delta \text{ in } \mathscr{G}) \Rightarrow \mathscr{E}$

where $\zeta_1, \zeta_2 \in \{\delta\} \cup \mathscr{A}$, $\mathscr{G}$ is a finite not empty subset of $\{\delta\} \cup \mathscr{A}$ and $\mathscr{E}$ is a finite set of e-constraints. We call the third kind of constraint a *conditional constraint*.

Definition 7.4 (*Solution of a set of constraints*)
An instantiation $\mathbf{i}$ *satisfies* a set of constraints $\mathscr{E}$ if

- $\zeta_1 \equiv \zeta_2 \in \mathscr{E}$ implies $\mathbf{i}(\zeta_1) \equiv \mathbf{i}(\zeta_2)$ (that is either $\mathbf{i}(\zeta_1) = \mathbf{i}(\zeta_2) = \delta$ or $\mathbf{i}(\zeta_1) = \mathbf{i}(\zeta_2) = \omega$), and
- $\zeta_1 \sqsubseteq \zeta_2 \in \mathscr{E}$ implies $\mathbf{i}(\zeta_1) \sqsubseteq \mathbf{i}(\zeta_2)$ (that is either $\mathbf{i}(\zeta_1) = \delta$ or $\mathbf{i}(\zeta_2) = \omega$), and
- $(\delta \text{ in } \mathscr{G}) \Rightarrow \mathscr{E}' \in \mathscr{E}$ implies that, if $\delta \in \mathbf{i}(\mathscr{G})$, then $\mathbf{i}$ satisfies $\mathscr{E}'$.

If $\mathbf{i}$ satisfies $\mathscr{E}$ we say that $\mathbf{i}$ is a *solution* of $\mathscr{E}$.

Let $\mathbf{sat}(\mathscr{E})$ denote the set of all the solutions of $\mathscr{E}$. According to Definition 7.4 above we have that $\mathbf{sat}(\mathscr{E}_1 \cup \mathscr{E}_2) = \mathbf{sat}(\mathscr{E}_1) \cap \mathbf{sat}(\mathscr{E}_2)$.

We can now give the definition of e-type scheme.

Definition 7.5 (*E-type schemes*)
An *e-type scheme* (*e-scheme* for short) is a pair $\langle \theta, \mathscr{E} \rangle$ where θ is an e-pattern and $\mathscr{E}$ is a finite set of e-constraints.

An e-scheme $\langle \theta, \mathscr{E} \rangle$ represents the set of e-types $\{\mathbf{i}(\theta) \mid \mathbf{i} \in \mathbf{sat}(\mathscr{E})\}$, where $\mathbf{i}(\theta)$ is the e-type obtained by applying the instantiation $\mathbf{i}$ to the e-pattern θ according to Definition 7.6 below. Replacements are extended to e-patterns by $\mathbf{r}(\alpha^{\mathsf{nat}}) = \mathbf{r}(\alpha)^{\mathsf{nat}}$, $\mathbf{r}(\theta \to \eta) = \mathbf{r}(\theta) \to \mathbf{r}(\eta)$, and $\mathbf{r}(\theta \times \eta) = \mathbf{r}(\theta) \times \mathbf{r}(\eta)$. Instead, the extension of instantiations to e-patterns is more complex, since we define it as a mapping from e-pattern to e-types.

Definition 7.6 (*Instantiation of an e-pattern*)
An instantiation $\mathbf{i}$ induces a mapping from e-patterns to e-types as specified by the following clauses.

- $\mathbf{i}(\alpha^{\mathsf{nat}}) = \mathbf{i}(\alpha)^{\mathsf{nat}}$
- $\mathbf{i}(\theta \to \eta) = \begin{cases} \omega^{\epsilon(\theta \to \eta)}, & \text{if } \mathbf{i}(\eta) \in \mathbf{L}_\omega \\ \mathbf{i}(\theta) \to \mathbf{i}(\eta), & \text{otherwise} \end{cases}$
- $\mathbf{i}(\theta_1 \times \theta_2) = \begin{cases} \omega^{\epsilon(\theta_1 \times \theta_2)}, & \text{if } \mathbf{i}(\theta_1), \mathbf{i}(\theta_2) \in \mathbf{L}_\omega \\ \mathbf{i}(\theta_1) \times \mathbf{i}(\theta_2), & \text{otherwise.} \end{cases}$

We extend the order $\sqsubseteq$ for basic properties (Definition 5.15.1) to a preorder for instantiations by considering the pointwise ordering.

Definition 7.7 (*Pointwise ordering on instantiations*)
Let $\mathbf{i}_1$, $\mathbf{i}_2$ be instantiations. We write $\mathbf{i}_1 \sqsubseteq \mathbf{i}_2$ if, for all $\alpha \in \mathscr{A}$, $\mathbf{i}_1(\alpha) \sqsubseteq \mathbf{i}_2(\alpha)$.

A useful property relating the $\sqsubseteq$ preorder for instantiations to the $\sqsubseteq$ preorder for e-types (Definition 5.15.2) is the following: for instantiations $\mathbf{i}_1$ and $\mathbf{i}_2$ and for an e-pattern θ, if $\mathbf{i}_1 \sqsubseteq \mathbf{i}_2$ then, from the covariance of $\sqsubseteq$ (see figure 8), $\mathbf{i}_1(\theta) \sqsubseteq \mathbf{i}_2(\theta)$.

$$(\sqsubseteq)\ \frac{\mathscr{E}[\delta/\alpha] \hookrightarrow \mathscr{I}}{\mathscr{E},\ \alpha \sqsubseteq \delta \hookrightarrow \mathscr{I} \cup \{\alpha\}} \qquad (\equiv)\ \frac{\mathscr{E}[\delta/\alpha] \hookrightarrow \mathscr{I}}{\mathscr{E},\ \mathrm{cns} \hookrightarrow \mathscr{I} \cup \{\alpha\}}\ \mathrm{cns} \in \{\alpha \equiv \delta, \delta \equiv \alpha\}$$

$$(\Rightarrow)\ \frac{\mathscr{E} \cup \mathscr{E}' \hookrightarrow \mathscr{I}}{\mathscr{E},\ (\delta\ \mathsf{in}\ \mathscr{G}) \Rightarrow \mathscr{E}' \hookrightarrow \mathscr{I}}\ \delta \in \mathscr{G} \qquad (\mathrm{STOP})\ \frac{\text{no other rule can be applied}}{\mathscr{E} \hookrightarrow \emptyset}$$

Fig. 13. 'Natural semantics' rules for e-constraints solution (system $\hookrightarrow$).

7.2 Computing the maximal solution of a set of constraints

In this section we prove that any finite set of constraints $\mathscr{E}$ has a maximal solution, and we give an algorithm for computing such a solution. In section 7.3, the e-type inference problem will be reduced to the solution of a finite set of e-constraints whose maximal solution corresponds to the optimal faithful typing. Before introducing the algorithm we look at a simple example.

Example 7.8
Consider the set of e-constraints:

$$\begin{aligned}\mathscr{E} \quad = \quad \{ \quad & \alpha_1 \sqsubseteq \delta, \\ & (\delta\ \mathsf{in}\ \{\alpha_1\}) \Rightarrow \{\alpha_1 \equiv \alpha_2, \alpha_3 \sqsubseteq \alpha_4\}, \\ & (\delta\ \mathsf{in}\ \{\alpha_3, \alpha_5\}) \Rightarrow \{\alpha_4 \sqsubseteq \delta, \alpha_6 \equiv \delta\}, \\ & (\delta\ \mathsf{in}\ \{\alpha_2\}) \Rightarrow \{\alpha_5 \sqsubseteq \alpha_6, \alpha_7 \equiv \delta\} \qquad \}.\end{aligned}$$

To find the maximal instantiation $\mathbf{i}$ in $\mathbf{sat}(\mathscr{E})$ observe that from the first constraint of $\mathscr{E}_1$ we get $\mathbf{i}(\alpha_1) = \delta$. Then from the second constraint we get $\mathbf{i}(\alpha_2) = \delta$, and finally from the last constraint of $\mathscr{E}$ we have $\mathbf{i}(\alpha_7) = \delta$. Let $\mathscr{I} = \{\alpha_1, \alpha_2, \alpha_7\}$, then $\mathbf{i}$ defined by: $\mathbf{i}(\alpha) = \delta$ if $\alpha \in \mathscr{I}$ and $\mathbf{i}(\alpha) = \omega$ otherwise, is the maximal instantiation in $\mathbf{sat}(\mathscr{E})$.

The algorithm for finding the maximal instantiation $\mathbf{i}$ that satisfies a finite set of e-constraints $\mathscr{E}$ is presented in natural semantics style using judgements $\mathscr{E} \hookrightarrow \mathscr{I}$, where $\mathscr{I}$ is the set of basic variables that *represents* $\mathbf{i}$, i.e. such that $\mathbf{i}(\alpha) = \delta$ if $\alpha \in \mathscr{I}$ and $\mathbf{i}(\alpha) = \omega$ otherwise. The idea is simply that of recognizing, following the equalities and the inequalities, all the basic variables that are forced to be instantiated to δ. All other basic variables are then replaced by ω in the maximal solution.

Definition 7.9 (*Algorithm for maximal solution of e-constraints*)
Let $\mathscr{E}$ be a finite set of e-constraints, and let cns be an e-constraint. The expression $\mathscr{E}, \mathrm{cns}$ denotes the set of constraints $\mathscr{E} \cup \{\mathrm{cns}\}$ where it is assumed that $\mathrm{cns} \notin \mathscr{E}$. We write $\mathscr{E} \hookrightarrow \mathscr{I}$ to mean that this judgement is derivable by the rules in figure 13.

Theorem 7.10 (*Soundness and completeness of algorithm* $\hookrightarrow$)
Let $\mathscr{E}$ be a finite set of e-constraints. Then $\mathscr{E} \hookrightarrow \mathscr{I}$ if and only if $\mathscr{I}$ represents the maximum of $\mathbf{sat}(\mathscr{E})$.

Proof
First, observe that $\mathscr{I}$ represents the maximal element of $\mathbf{sat}(\mathscr{E})$ if and only if for all $\mathscr{I}'$ representing an instantiation $\mathbf{i}' \in \mathbf{sat}(\mathscr{E})$, $\mathscr{I} \subseteq \mathscr{I}'$. Moreover if there is an

instantiation $\mathbf{i} \in \mathbf{sat}(\mathscr{E})$ then there is a maximal one. The result can now be proved by induction on the number of symbols in $\mathscr{E}$, observing that if $\mathscr{E}$ does not contain a constraint of the kind $\alpha \sqsubseteq \delta$, $\alpha \equiv \delta$, $\delta \equiv \alpha$, or $(\delta \text{ in } \mathscr{G}) \Rightarrow \mathscr{E}'$ with $\delta \in \mathscr{G}$, then the maximal solution of $\mathscr{E}$ is the one that assigns ω to all the basic variables. □

Remark 7.11 (*Complexity of algorithm* $\hookrightarrow$)
Define the *length of an e-constraint* cns, |cns|, and the *length of a finite set of e-constraints* $\mathscr{E}$, $|\mathscr{E}|$, to be the number of symbols in cns or $\mathscr{E}$. Given $\mathscr{E}$ we can find $\mathscr{I}$ such that $\mathscr{E} \hookrightarrow \mathscr{I}$ in linear time in $|\mathscr{E}|$.

7.3 An algorithm for inferring $\vdash_1$-typings

In this section we first introduce notions and simple functions involving e-types and e-patterns. Then we present the e-type inference algorithm $\mathscr{W}_1$ for $\vdash_1$.

1. Let θ, η be e-patterns such that $\epsilon(\theta) = \epsilon(\eta)$. We say that a replacement $\mathbf{r}$ is a unifier for θ and η if $\mathbf{r}(\theta) = \mathbf{r}(\eta)$. A *most general unifier* (*m.g.u.* for short) is a unifier $\mathbf{r}_0$ such that, for every unifier $\mathbf{r}$, there exist a replacement $\mathbf{r}'$ such that: $\mathbf{r} = \mathbf{r}' \circ \mathbf{r}_0$, where '$\circ$' denotes function composition.
   It is easy check that $\mathscr{U}(\theta, \eta)$, where $\mathscr{U}$ is the algorithm in figure 14, is a m.g.u. for θ and η.
2. Let ρ be a type. By $\mathbf{fresh}(\rho)$ we denote an e-pattern obtained from ρ by assigning a fresh basic variable to each occurrence of any ground type in ρ. For example: $\mathbf{fresh}(\mathsf{nat} \to \mathsf{nat}) = \alpha^{\mathsf{nat}} \to \beta^{\mathsf{nat}}$. For a set of term variables Γ, $\mathbf{fresh}(\Gamma) = \{x^{\rho} : \mathbf{fresh}(\rho) \mid x^{\rho} \in \Gamma\}$.
3. The function **vars** maps an e-pattern θ to the finite set of its basic variables. For instance, $\mathbf{vars}(\alpha^{\mathsf{nat}} \to \beta^{\mathsf{nat}}) = \{\alpha, \beta\}$.
4. The function **tail**, that maps e-patterns and δ-e-types to finite subsets of $\{\delta\} \cup \mathscr{A}$, is inductively defined by: $\mathbf{tail}(\zeta^{\mathsf{nat}}) = \{\zeta\}$ (for $\zeta \in \{\delta\} \cup \mathscr{A}$), $\mathbf{tail}(\theta \times \eta) = \mathbf{tail}(\theta) \cup \mathbf{tail}(\eta)$, and $\mathbf{tail}(\theta \to \eta) = \mathbf{tail}(\eta)$. We have that for all instantiations $\mathbf{i}$:
   (a) $\mathbf{i}(\mathbf{tail}(\theta)) = \{\omega\}$ if and only if $\mathbf{i}(\theta) = \omega^{\epsilon(\theta)}$.
5. Let θ, η be e-patterns or δ-e-types such that $\epsilon(\theta) = \epsilon(\eta)$. Then $\mathbf{cs}_{\leqslant}(\theta, \eta)$ and $\mathbf{acs}_{\leqslant}(\theta, \eta)$ denote[5] the sets of constraints inductively defined by the clauses in figures 15 and 16. We have that for all instantiations $\mathbf{i}$:
   (a) $\mathbf{i}(\eta) \notin \mathbf{L}_{\omega}$ implies: $\mathbf{i}(\theta) \leqslant \mathbf{i}(\eta)$ if and only if $\mathbf{i} \in \mathbf{sat}(\mathbf{cs}_{\leqslant}(\theta, \eta))$, and
   (b) $\mathbf{i}(\theta) \leqslant \mathbf{i}(\eta)$ if and only if $\mathbf{i} \in \mathbf{sat}(\mathbf{acs}_{\leqslant}(\theta, \eta))$.
6. For each constant k an e-scheme $\mathbf{ets}(\mathsf{k})$ is specified. The function $\mathbf{ets}(\cdot)$ is such that: for an integer k, $\mathbf{ets}(\mathsf{k}) = \langle \alpha^{\mathsf{nat}}, \emptyset \rangle$, for an unary operator k_1 of type $\mathsf{nat} \to \mathsf{nat}$, $\mathbf{ets}(\mathsf{k}_1) = \langle \alpha^{\mathsf{nat}} \to \beta^{\mathsf{nat}}, \{\alpha \sqsubseteq \beta\} \rangle$, and for a binary operator k_2 of type $\mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$, $\mathbf{ets}(\mathsf{k}_2) = \langle \alpha^{\mathsf{nat}} \times \beta^{\mathsf{nat}} \to \gamma^{\mathsf{nat}}, \{\alpha \sqsubseteq \gamma, \beta \sqsubseteq \gamma\} \rangle$, where α, β, γ are fresh variables. For all constants k let $\mathbf{ets}(\mathsf{k}) = \langle \theta, \mathscr{E} \rangle$. We have that:
   (a) $\mathbf{i} \in \mathbf{sat}(\mathscr{E})$ if and only if either $\mathbf{i}(\theta) \in \mathbf{L}_{\omega}$ or $\mathbf{i}(\theta) \in \mathbf{L}_{\delta}$.

[5] Here '**cs**' stands for 'constraint set' and '**acs**' stands for 'auxiliary constraint set'.

$$\mathscr{U}(\alpha^{\mathsf{nat}}, \beta^{\mathsf{nat}}) = [\beta := \alpha]$$

$$\mathscr{U}(\theta_1 \to \theta_2, \eta_1 \to \eta_2) = \text{let } \mathbf{r} = \mathscr{U}(\theta_2, \eta_2) \text{ in } \mathscr{U}(\mathbf{r}(\theta_1), \mathbf{r}(\eta_1)) \circ \mathbf{r} \text{ end}$$

$$\mathscr{U}(\theta_1 \times \theta_2, \eta_1 \times \eta_2) = \text{let } \mathbf{r} = \mathscr{U}(\theta_1, \eta_1) \text{ in } \mathscr{U}(\mathbf{r}(\theta_2), \mathbf{r}(\eta_2)) \circ \mathbf{r} \text{ end}$$

Fig. 14. E-pattern unification algorithm $\mathscr{U}$.

$$\mathbf{cs}_{\leqslant}(\zeta_1^{\mathsf{nat}}, \zeta_2^{\mathsf{nat}}) = \{\zeta_1 \sqsubseteq \zeta_2\}, \text{ where } \zeta_1, \zeta_2 \in \{\delta\} \cup \mathscr{A}$$

$$\mathbf{cs}_{\leqslant}(\theta_1 \to \cdots \to \theta_n \to \theta, \eta_1 \to \cdots \to \eta_n \to \eta) = \mathbf{cs}_{\leqslant}(\theta, \eta) \cup \bigcup_{1 \leqslant i \leqslant n} \mathbf{acs}_{\leqslant}(\eta_i, \theta_i),$$
where $n \geqslant 1$ and θ, η are not arrow e-patterns or arrow δ-e-types.

$$\mathbf{cs}_{\leqslant}(\theta_1 \times \theta_2, \eta_1 \times \eta_2) = \mathbf{acs}_{\leqslant}(\theta_1, \eta_1) \cup \mathbf{acs}_{\leqslant}(\theta_2, \eta_2)$$

Fig. 15. Function $\mathbf{cs}_{\leqslant}$.

We can now define the e-type inference algorithm $\mathscr{W}_1$. This algorithm is presented in figures 17 and 18. Let $M \in \Lambda_\rho$, if $\mathscr{W}_1(M) = \langle \Theta, M'^{\theta}, \mathscr{E} \rangle$ then Θ is a basis that associates to each term variable in FV(M) an e-pattern, M'^{θ} is a term decorated with e-patterns, and $\mathscr{E}$ is a finite set of e-constraints. We can read the conditional constraints generated by the algorithm $\mathscr{W}$ as saying: if this terms is not assigned an e-type with the rule (ω), then its decoration must satisfy the following constraints. Notice that for abstractions, pairs, and projections these conditional constraints are not generated since in these cases they are already present in the constraints generated by the subterms, and would be superfluous. This is also the case for constants, since looking at the e-scheme for constants, we can see that for all constants k, $\mathbf{ets}(\mathsf{k}) = \langle \theta, \mathscr{E} \rangle$ is such that $\mathbf{i} \in \mathbf{sat}(\mathscr{E})$ if and only if $\mathbf{i} \in \mathbf{sat}(\{(\delta \text{ in } \mathbf{tail}(\theta)) \Rightarrow \mathscr{E}\})$.

Replacements are extended to decorated terms by applying the replacement to every e-pattern in the term. The extension of instantiations to decorated terms is a mapping from e-pattern decorated terms to e-type decorated terms.

Definition 7.12 (*Instantiation of an e-pattern decorated term*)
An instantiation **i** induces a mapping from terms decorated with e-patterns to terms

$$\mathbf{acs}_{\leqslant}(\theta, \eta) = \begin{cases} \{(\delta \text{ in } \mathbf{tail}(\eta)) \Rightarrow \mathbf{cs}_{\leqslant}(\theta, \eta)\}, & \text{if } \theta, \eta \text{ are arrow e-patterns or} \\ & \quad \text{arrow } \delta\text{-e-types} \\ \mathbf{cs}_{\leqslant}(\theta, \eta), & \text{otherwise} \end{cases}$$

Fig. 16. Function $\mathbf{acs}_{\leqslant}$.

$$
\begin{array}{lll}
\mathscr{W}_1(P) = & \text{let} & \Theta = \mathbf{fresh}(\mathrm{FV}(P)) \\
 & & \text{and } \langle P', \mathscr{E}\rangle = \mathscr{W}(\Theta, P) \\
 & \text{in} & \langle \Theta, P', \mathscr{E}\rangle \text{ end}
\end{array}
$$

Fig. 17. Algorithm $\mathscr{W}_1$.

$$
\begin{array}{l}
\mathscr{W}(\Theta, P) = \text{case } P \text{ of} \\
\quad \mathsf{k}: \quad \text{let } \langle \theta, \mathscr{E}\rangle = \mathbf{ets}(\mathsf{k}) \\
\qquad\quad \text{in } \langle \mathsf{k}^{\theta}, \mathscr{E}\rangle \text{ end} \\
\quad x: \quad \text{let } \theta_1 = \Theta(x) \quad \text{and } \theta_2 = \mathbf{fresh}(\epsilon(\theta_1)) \\
\qquad\quad \text{in } \langle x^{\theta_2}, (\delta \in \mathbf{tail}(\theta_2)) \Rightarrow \mathbf{cs}_{\leqslant}(\theta_1, \theta_2)\rangle \text{ end} \\
\quad \lambda x^{\rho}.M: \quad \text{let } \theta = \mathbf{fresh}(\rho) \\
\qquad\quad \text{and } \langle M'^{\eta}, \mathscr{E}\rangle = \mathscr{W}(\Theta \cup \{x:\theta\}, M) \\
\qquad\quad \text{in } \langle (\lambda x.M')^{\theta\to\eta}, \mathscr{E}\rangle \text{ end} \\
\quad MN_1\cdots N_p, \text{ where } p \geqslant 1 \text{ and } M \text{ is not an application}: \\
\qquad\quad \text{let } \langle M'^{\eta_1\to\cdots\to\eta_p\to\eta}, \mathscr{E}_0\rangle = \mathscr{W}(\Theta, M) \\
\qquad\quad\quad \text{and, for all } 1 \leqslant i \leqslant p,\ \langle N_i'^{\theta_i}, \mathscr{E}_i\rangle = \mathscr{W}(\Theta, N_i) \\
\qquad\quad\quad \text{and } \mathbf{r} = \mathscr{U}(\eta_1, \theta_1)\circ(\cdots\circ\mathscr{U}(\eta_p, \theta_p)\cdots) \\
\qquad\quad \text{in } \langle \mathbf{r}((M'N_1'^{\theta_1}\cdots N_p'^{\theta_p})^{\eta}), \{(\delta \text{ in } \mathbf{tail}(\mathbf{r}(\eta))) \Rightarrow \mathbf{r}(\mathscr{E}_0 \cup \mathscr{E}_1 \cup \cdots \cup \mathscr{E}_p)\}\rangle \text{ end} \\
\quad \langle M_1, M_2\rangle: \quad \text{let } \langle M_1'^{\theta_1}, \mathscr{E}_1\rangle = \mathscr{W}(\Theta, M_1) \\
\qquad\quad\quad \text{and } \langle M_2'^{\theta_2}, \mathscr{E}_2\rangle = \mathscr{W}(\Theta, M_2) \\
\qquad\quad \text{in } \langle (\langle M_1', M_2'\rangle)^{\theta_1\times\theta_2}, \mathscr{E}_1 \cup \mathscr{E}_2\rangle \text{ end} \\
\quad \mathsf{prj}_i M: \quad \text{let } \langle M'^{\theta_1\times\theta_2}, \mathscr{E}\rangle = \mathscr{W}(\Theta, M) \\
\qquad\quad \text{in } \langle (\mathsf{prj}_i M'^{\theta_{3-i}})^{\theta_i}, \mathscr{E}\rangle \text{ end} \\
\quad \mathsf{fix}\ x^{\rho}.M: \\
\qquad\quad \text{let } \theta_1 = \mathbf{fresh}(\rho) \quad \text{and } \theta_2 = \mathbf{fresh}(\rho) \\
\qquad\quad\quad \text{and } \langle M'^{\eta}, \mathscr{E}\rangle = \mathscr{W}(\Theta \cup \{x:\theta_1\}, M) \\
\qquad\quad\quad \text{and } \mathbf{r} = \mathscr{U}(\theta_1, \eta) \\
\qquad\quad \text{in } \langle \mathbf{r}((\mathsf{fix}\, x.M'^{\eta})^{\theta_2}), \{(\delta \text{ in } \mathbf{tail}(\mathbf{r}(\theta_2))) \Rightarrow \mathbf{r}(\mathscr{E} \cup \mathbf{cs}_{\leqslant}(\theta_1, \theta_2))\}\rangle \text{ end} \\
\quad \mathsf{if}z\ N \text{ then } M_1 \text{ else } M_2: \\
\qquad\quad \text{let } \langle N'^{\alpha^{\mathsf{nat}}}, \mathscr{E}_0\rangle = \mathscr{W}(\Theta, N) \\
\qquad\quad\quad \text{and } \langle M_1'^{\theta_1}, \mathscr{E}_1\rangle = \mathscr{W}(\Theta, M_1) \\
\qquad\quad\quad \text{and } \langle M_2'^{\theta_2}, \mathscr{E}_2\rangle = \mathscr{W}(\Theta, M_2) \\
\qquad\quad\quad \text{and } \mathbf{r} = \mathscr{U}(\theta_1, \theta_2) \\
\qquad\quad \text{in } \langle \mathbf{r}((\mathsf{if}z\ N' \text{ then } M_1' \text{ else } M_2')^{\theta_1}), \\
\qquad\qquad \{(\delta \text{ in } \mathbf{tail}(\mathbf{r}(\theta_1))) \Rightarrow \mathbf{r}(\{\alpha \equiv \delta\} \cup \mathscr{E}_0 \cup \mathscr{E}_1 \cup \mathscr{E}_2)\}\rangle \text{ end}
\end{array}
$$

Fig. 18. Algorithm $\mathscr{W}$.

decorated with e-types as specified by the following clauses.

$$
\mathbf{i}(P^{\theta}) = \begin{cases} \epsilon(P)^{\mathbf{i}(\theta)}, & \text{if } \mathbf{i}(\theta) \in \mathbf{L}_{\omega} \\ \mathbf{i}(P)^{\mathbf{i}(\theta)}, & \text{otherwise}, \end{cases}
$$

where

$$\mathbf{i}(P) = \begin{cases} \mathsf{k}, & \text{if } P = \mathsf{k} \\ x, & \text{if } P = x \\ \lambda x.\mathbf{i}(M), & \text{if } P = \lambda x.M \\ \mathbf{i}(M)\mathbf{i}(N)^{\mathbf{i}(\theta_2)}, & \text{if } P = MN^{\theta_2} \\ \langle \mathbf{i}(M_1), \mathbf{i}(M_2) \rangle, & \text{if } P = \langle M_1, M_2 \rangle \\ \mathsf{prj}_i \mathbf{i}(M)^{\mathbf{i}(\theta_{3-i})}, & \text{if } P = \mathsf{prj}_i M^{\theta_{3-i}} \\ \mathsf{fix}\, x.\mathbf{i}(M), & \text{if } P = \mathsf{fix}\, x.M \\ \mathsf{ifz}\ \mathbf{i}(N)\ \mathsf{then}\ \mathbf{i}(M_1)\ \mathsf{else}\ \mathbf{i}(M_2), & \text{if } P = \mathsf{ifz}\ N\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2. \end{cases}$$

Correctness and completeness of the inference are expressed by the following proposition.

Proposition 7.13 (*Soundness and completeness of* $\mathscr{W}_1$ *w.r.t.* $\vdash_1$)
Let $P \in \Lambda_\rho$ and $\mathscr{W}_1(P) = \langle \Theta, P'^{\theta}, \mathscr{E} \rangle$. Then

1. for all instantiations $\mathbf{i} \in \mathbf{sat}(\mathscr{E})$, $\mathbf{i}(\Theta) \vdash_1 \mathbf{i}(P'^{\theta})$,
2. for all Σ and P''^{ϕ} such that $\epsilon(\Sigma) = \mathrm{FV}(\epsilon(P''^{\phi}))$ and $\epsilon(P''^{\phi}) = P$, if $\Sigma \vdash_1 P''^{\phi}$ then there exists $\mathbf{i} \in \mathbf{sat}(\mathscr{E})$ such that $\mathbf{i}(\Theta) = \Sigma$ and $\mathbf{i}(P'^{\theta}) = P''^{\phi}$.

Proof
From the definition of $\mathscr{W}_1$ (see figure 17) we have that $\langle P'^{\theta}, \mathscr{E} \rangle = \mathscr{W}(\Theta, P)$, where $\Theta = \mathbf{fresh}(\mathrm{FV}(P))$.

1 (soundness). By induction on the structure of terms.

2 (completeness). By induction on the structure of derivations. We consider only four cases:

- The derivation ends with rule (ω), so $\phi \in \mathbf{L}_\omega$. Observe that, for all terms P, the set of constraints returned by $\mathscr{W}(\Theta, P)$ is satisfied by the instantiation that maps every basic variable α to ω. The proof of this fact is by induction on the structure on terms.
- The derivation ends with rule (Var), so $\phi \notin \mathbf{L}_\omega$, $P''^{\phi} = x^{\phi}$, $\Sigma = \{x : \phi_1\}$ (for some ϕ_1 such that $\phi_1 \leqslant \phi$), $\Theta = \{x : \theta_1\}$, and $\mathscr{W}(\Theta, P) = \langle x^{\theta}, (\delta\, \mathsf{in}\, \mathbf{tail}(\theta)) \Rightarrow \mathbf{cs}_{\leqslant}(\theta_1, \theta) \rangle$.
  The result follows by property (a) of **tail** and by property (b) of $\mathbf{cs}_{\leqslant}$ (see points (4) and (5) at the beginning of this subsection).
- The derivation ends with rule ($\to$ E), so $\phi \notin \mathbf{L}_\omega$, $P''^{\phi} = (M'' N_1''^{\phi_1} \cdots N_p''^{\phi_p})^{\phi}$ (where M is not an application). We consider the case $p = 1$. We have that
  1. $\Sigma \vdash_1 M''^{\phi_1 \to \phi}$, $\Sigma \vdash_1 N_1''^{\phi_1}$, and $\Sigma \vdash_1 (M'' N_1''^{\phi_1})^{\phi}$,
  2. $\mathscr{W}(\Theta, M) = \langle M'^{\eta_1 \to \theta}, \mathscr{E}_0 \rangle$, $\mathscr{W}(\Theta, N_1) = \langle N_1'^{\theta_1}, \mathscr{E}_1 \rangle$, and $\mathscr{W}(\Theta, P) = \langle \mathbf{r}((M' N_1'^{\theta_1})^{\theta}), \{(\delta\ \mathsf{in}\ \mathbf{tail}(\mathbf{r}(\theta))) \Rightarrow \mathbf{r}(\mathscr{E}_0 \cup \mathscr{E}_1)\} \rangle$, where $\mathbf{r} = \mathscr{U}(\eta_1, \theta_1)$.

  By induction there is $\mathbf{i}_0 \in \mathbf{sat}(\mathscr{E}_0)$ such that $\mathbf{i}_0(\Theta) = \Sigma$ and $\mathbf{i}_0(M'^{\eta_1 \to \theta}) = M''^{\phi_1 \to \phi}$, and there is $\mathbf{i}_1 \in \mathbf{sat}(\mathscr{E}_1)$ such that $\mathbf{i}_1(\Theta) = \Sigma$ and $\mathbf{i}_1(N'^{\theta_1}) = N_1''^{\phi_1}$. The only basic variables that occur in both $\mathscr{E}_0$ and $\mathscr{E}_1$ are those in Θ so, since algorithm $\mathscr{U}$ returns a most general unifier, there must exist $\mathbf{i} \in$

$\mathbf{sat}(\{(\delta \text{ in } \mathbf{tail}(\mathbf{r}(\theta))) \Rightarrow \mathbf{r}(\mathscr{E}_0 \cup \mathscr{E}_1)\})$ such that $\mathbf{i}(\Theta) = \Sigma$ and $\mathbf{i}(\mathbf{r}((M' N_1'^{\theta_1})^{\theta})) = (M'' N_1''^{\phi_1})^{\phi}$.

- The derivation ends with rule (Fix), so $\phi \notin \mathbf{L}_\omega$ and $P = \mathsf{fix}\, x.M$. We have that
  1. $\Sigma, x : \phi_1 \vdash_1 M''^{\phi_1}$ and $\Sigma \vdash_1 (\mathsf{fix}\, x.M''^{\phi_1})^{\phi}$, where $\phi_1 \leqslant \phi$,
  2. $\mathscr{W}(\Theta \cup \{x : \theta_1\}, M) = \langle M'^{\eta}, \mathscr{E}\rangle$ and
     $\mathscr{W}(\Theta, P) = \langle \mathbf{r}((\mathsf{fix}\, x.M'^{\eta})^{\theta}), \{(\delta \text{ in } \mathbf{tail}(\mathbf{r}(\theta))) \Rightarrow \mathbf{r}(\mathscr{E} \cup \mathbf{cs}_{\leqslant}(\theta_1, \theta))\}\rangle$,
     where $\mathbf{r} = \mathscr{U}(\theta_1, \eta)$.

  By induction there is $\mathbf{i}_0 \in \mathbf{sat}(\mathscr{E})$ such that $\mathbf{i}_0(\Theta \cup \{x : \theta_1\}) = \Sigma \cup \{x : \phi_1\}$ and $\mathbf{i}_0(M'^{\eta}) = M''^{\phi_1}$. So the result follows by the fact that algorithm $\mathscr{U}$ returns a most general unifier.

□

We are interested in finding the optimal faithful $\vdash_1$-typing, since it shows all the useless-code that can be detected by using system $\vdash_1$. This can be done as shown by the following theorem.

Theorem 7.14 (*Optimal faithful $\vdash_1$-typing*)
Let $M \in \Lambda_\rho$, $\mathscr{W}_1(M) = \langle \Theta, M'^{\theta}, \mathscr{E}\rangle$. Define: $\mathbf{faithful}(\Theta, \theta) =$

$$\bigcup_{x:\eta \in \Theta} \{(\delta \text{ in } \mathbf{tail}(\eta)) \Rightarrow \{\gamma \equiv \delta | \gamma \in \mathbf{vars}(\eta)\}\} \quad \cup \quad \{\alpha \equiv \delta | \alpha \in \mathbf{vars}(\theta)\}.$$

If $\mathbf{i}$ is the maximum element of $\mathbf{sat}(\mathscr{E} \cup \mathbf{faithful}(\Theta, \theta))$ then $\mathbf{i}(\Theta) \vdash_1 \mathbf{i}(M'^{\theta})$ is the optimal faithful $\vdash_1$-typing.

Proof
First observe that, for every $x : \eta \in \Theta$, the constraint $(\delta \text{ in } \mathbf{tail}(\eta)) \Rightarrow \{\gamma \equiv \delta | \gamma \in \mathbf{vars}(\eta)\}$ forces η to be instantiated either to an ω-e-type or to a δ-e-type. Then the result follows by Proposition 7.13 since, for every $\mathbf{i}_1, \mathbf{i}_2 \in \mathbf{sat}(\mathscr{E} \cup \mathbf{faithful}(\Theta, \theta))$, $\mathbf{i}_1 \sqsubseteq \mathbf{i}_2$ implies $\mathbf{i}_1(\Theta) \vdash_1 \mathbf{i}_1(M'^{\theta}) \sqsubseteq \mathbf{i}_2(\Theta) \vdash_1 \mathbf{i}_2(M'^{\theta})$. □

Example 7.15
By applying algorithm $\mathscr{W}_1$ to the term

$$N = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle y^{\mathsf{nat}}, x\rangle) z^{\mathsf{nat}}$$

we get $\mathscr{W}_1(N) = \langle \Theta, N''^{\alpha_1'}, \mathscr{E}_1\rangle$, where (writing ζ instead of ζ^{nat}):

$$\begin{array}{ll} \Theta & = \{y : \alpha_1, z : \alpha_2\}, \\ N''^{\alpha_1'} & = ((\lambda x.\mathsf{prj}_1\langle y, x\rangle^{\beta_1'}) z^{\beta_1})^{\alpha_1'}, \quad \text{and} \\ \mathscr{E}_1 & = \{(\delta \text{ in } \{\alpha_1'\}) \Rightarrow \{\alpha_1 \sqsubseteq \alpha_1'\}, (\delta \text{ in } \{\beta_1'\}) \Rightarrow \{\beta_1 \sqsubseteq \beta_1'\}, (\delta \text{ in } \{\beta_1\}) \Rightarrow \{\alpha_2 \sqsubseteq \beta_1\}\}. \end{array}$$

We also have

$$\mathbf{faithful}(\Theta, \alpha_1') \quad = \quad \{ \ (\delta \text{ in } \{\alpha_1\}) \Rightarrow \{\alpha_1 \equiv \delta\}, \ (\delta \text{ in } \{\alpha_2\}) \Rightarrow \{\alpha_2 \equiv \delta\}, \ \alpha_1' \equiv \delta \ \}.$$

Applying the algorithm of figure 13, we get $\mathscr{E}_1 \cup \mathbf{faithful}(\Theta, \alpha_1') \hookrightarrow \mathscr{I}_1$, where $\mathscr{I}_1 = \{\alpha_1', \alpha_1\}$.

Let $\mathbf{i}_1$ be the instantiation represented by $\mathscr{I}_1$, then $\mathbf{i}_1(\Theta) \vdash_1 \mathbf{i}_1(N''^{\alpha'_1})$, where

$$\begin{array}{lcl} \mathbf{i}_1(\Theta) & = & \{y : \delta,\ z : \omega\} \quad \text{and} \\ \mathbf{i}_1(N''^{\alpha'_1}) & = & ((\lambda x.\mathsf{prj}_1\langle y, x\rangle^{\omega})z^{\omega})^{\delta}\ , \end{array}$$

is the faithful $\vdash_1$-typing showing all the useless-code that can be detected by using $\vdash_1$.

By applying the simplification mapping $\mathbf{O}_1$ we get:

$$\begin{array}{lcl} \mathbf{O}_1(\mathbf{i}_1(\Theta)) & = & \{y : \delta\}, \quad \text{and} \\ \mathbf{O}_1(\mathbf{i}_1(N''^{\alpha'_1})) & = & ((\lambda x.\mathsf{prj}_1\langle y, \mathsf{d}_1\rangle^{\omega})\mathsf{d}_2^{\omega})^{\delta}. \end{array}$$

Remark 7.16 (*Complexity of algorithm* $\mathscr{W}_1$)

Define the *length of a PCFP type* ρ, $|\rho|$, and *length of a PCFP term* P, $|P|$, to be the number of symbols in ρ or P. Define the *width of a PCFP type* ρ, $\mathbf{width}(\rho)$, and the *width of a PCFP term* P, $\mathbf{width}(P)$, as follows

$$\mathbf{width}(\rho) = \begin{cases} 1, & \text{if } \rho = \mathsf{nat} \\ \mathbf{width}(\rho_2), & \text{if } \rho = \rho_1 \to \rho_2 \\ \mathbf{width}(\rho_1) + \mathbf{width}(\rho_1), & \text{if } \rho = \rho_1 \times \rho_2 \end{cases}$$

$$\mathbf{width}(P) = \max\{\mathbf{width}(\sigma) \mid Q \text{ is a subterm of } M \text{ of type } \sigma\}.$$

Define the *length of a pattern decorated term* P'^{θ}, $|P'^{\theta}|$, to be the number of symbols in P'^{θ}.

For every PCFP term P, let $\mathscr{W}_1(P) = \langle \Theta, P'^{\theta}, \mathscr{E}\rangle$. The following results hold:

- $|P'^{\theta}|$ is linear in $|P|$.
- Since the set $\mathbf{tail}(\eta)$, for some η which occur in P'^{θ}, may contain at most $\mathbf{width}(P)$ elements, $|\mathscr{E}|$ has an upper bound of order $\mathbf{width}(P)|P|$. If we restrict to the language without the type constructor $\times$ (and therefore to terms without pairs and projections) then $|\mathscr{E}|$ is linear in $|P|$.
- The execution time of $\mathscr{W}_1(P)$ is at worst quadratic in $|P|$ (due to the linear unifications in the clauses for application, fix, and ifz).

7.4 An algorithm for inferring $\vdash_2$*-typings*

The only difference between $\vdash_1$ and $\vdash_2$ is in the use of the $\leqslant$ in rules (Var) and (Fix). This use is reflected in the inference algorithm $\mathscr{W}_1$ by generating the constraints $\mathbf{cs}_{\leqslant}$ for variables and recursive terms. The inference algorithm for the system $\vdash_2$, $\mathscr{W}_2$, is therefore obtained from the algorithm $\mathscr{W}_1$ in figure 17 by replacing (in the algorithm $\mathscr{W}$ in figure 18) the occurrence of $\mathbf{cs}_{\leqslant}$ with $\mathbf{cs}_{=}$, which is defined by the clauses in figure 19. It is easy to see that, for all instantiations $\mathbf{i}$: $\mathbf{i}(\theta) = \mathbf{i}(\eta)$ if and only if $\mathbf{i} \in \mathbf{sat}(\mathbf{cs}_{=}(\theta, \eta))$.

The inference algorithm $\mathscr{W}_2$ is correct and complete w.r.t. the $\vdash_2$ assignment system, and we can find the optimal faithful $\vdash_2$-typing in the same way as for $\vdash_1$.

$$\mathbf{cs}_{=}(\zeta_1^{\mathsf{nat}}, \zeta_2^{\mathsf{nat}}) = \{\zeta_1 \equiv \zeta_2\}, \text{ where } \zeta_1, \zeta_2 \in \{\delta\} \cup \mathscr{A}$$

$$\mathbf{cs}_{=}(\theta_1 \to \eta_1, \theta_2 \to \eta_2) = \mathbf{cs}_{=}(\theta_1, \theta_2) \cup \mathbf{cs}_{=}(\eta_1, \eta_2)$$

$$\mathbf{cs}_{=}(\theta_1 \times \eta_1, \theta_2 \times \eta_2) = \mathbf{cs}_{=}(\theta_1, \theta_2) \cup \mathbf{cs}_{=}(\eta_1, \eta_2)$$

Fig. 19. Function $\mathbf{cs}_{=}$.

Example 7.17
By applying algorithm $\mathscr{W}_2$ to the term N of Example 7.15 we get $\mathscr{W}_2(N) = \langle \Theta, N''^{\alpha'_1}, \mathscr{E}_2 \rangle$ where Θ and $N''^{\alpha'_1}$ are as in Example 7.15, while

$$\mathscr{E}_2 = \{(\delta \text{ in } \{\alpha'_1\}) \Rightarrow \{\alpha_1 \equiv \alpha'_1\}, (\delta \text{ in } \{\beta'_1\}) \Rightarrow \{\beta_1 \equiv \beta'_1\}, (\delta \text{ in } \{\beta_1\}) \Rightarrow \{\alpha_2 \equiv \beta_1\}\}.$$

The set $\mathbf{faithful}(\Theta, \alpha'_1)$ is as in Example 7.15, and by applying the algorithm of figure 13 we get $\mathscr{E}_2 \cup \mathbf{faithful}(\Theta, \alpha'_1) \hookrightarrow \mathscr{I}_2$, where $\mathscr{I}_2 = \{\alpha'_1, \alpha_1\}$. Since $\mathscr{I}_2$ is equal to the set $\mathscr{I}_1$ in Example 7.15 the faithful $\vdash_1$-typing $\mathbf{i}_1(\Theta) \vdash_1 \mathbf{i}_1(N''^{\alpha'_1})$ of Example 7.15 is also a faithful $\vdash_2$-typing.

By applying the simplification mapping $\mathbf{O}_2$ we get:

$$\begin{array}{lcl} \mathbf{O}_2(\mathbf{i}_1(\Theta)) & = & \{y : \delta\}, \quad \text{and} \\ \mathbf{O}_2(\mathbf{i}_1(N''^{\alpha'_1})) & = & y^{\delta} \ . \end{array}$$

8 A useless-code detection and elimination procedure

Let $\mathbf{O}_1^{\epsilon} : \Lambda_{\mathbf{T}} \to \Lambda_{\mathbf{T}}$ be the mapping that given a PCFP term M finds the optimal faithful $\vdash_1$-typing of M, $\Sigma_1 \vdash_1 M'^{\phi_1}$, and then returns $\epsilon(\mathbf{O}_1(M'^{\phi_1}))$. Similarly, let $\mathbf{O}_2^{\epsilon} : \Lambda_{\mathbf{T}} \to \Lambda_{\mathbf{T}}$ be the mapping that given a PCFP term M finds the optimal faithful $\vdash_2$-typing of M, $\Sigma_2 \vdash_2 M''^{\phi_2}$, and then returns $\epsilon(\mathbf{O}_2(M^{\phi_2}))$. The following property holds.

Proposition 8.1
For every PCFP term M,

1. $\mathbf{O}_1^{\epsilon}(M) = \mathbf{O}_1^{\epsilon}(\mathbf{O}_1^{\epsilon}(M))$,
2. $\mathbf{O}_2^{\epsilon}(M) = \mathbf{O}_2^{\epsilon}(\mathbf{O}_2^{\epsilon}(M))$, and
3. $\mathbf{O}_2^{\epsilon}(\mathbf{O}_1^{\epsilon}(M)) = \mathbf{O}_1^{\epsilon}(\mathbf{O}_2^{\epsilon}(\mathbf{O}_1^{\epsilon}(M))) = \mathbf{O}_2^{\epsilon}(\mathbf{O}_1^{\epsilon}(\mathbf{O}_2^{\epsilon}(M)))$.

Proof

1. We will show that, if $\Sigma \vdash_1 M'$ is the optimal faithful $\vdash_1$-typing of M, then $\Sigma_1 \vdash_1 \mathbf{O}_1(M')$, where $\Sigma_1 = \mathbf{O}_1(\Sigma) \cup \{\mathsf{d}^{\sigma} : \omega^{\sigma} \mid \mathsf{d}^{\sigma} \in \mathrm{DV}(\epsilon(\mathbf{O}_1(M')))\}$, is the optimal faithful $\vdash_1$-typing of $N = \epsilon(\mathbf{O}_1(M')) = \mathbf{O}_1^{\epsilon}(M)$.
   From Proposition 5.9.1 we have that $\Sigma_1 \vdash_1 \mathbf{O}_1(M')$ is a faithful $\vdash_1$-typing of N. Assume, by contraposition, that this is not the optimal one. This implies that the optimal faithful $\vdash_1$-typing of N, say $\Sigma' \vdash_1 N'$, is such that an ω-e-type is assigned to some subexpression of N different from d. Then it is possible to

define (starting from the derivation of $\Sigma' \vdash_1 N'$ and looking at the definition of $\mathbf{O}_1$) a faithful $\vdash_1$-typing of M, $\Sigma'' \vdash_1 M''$, with

$$\Sigma'' = \{x^\rho : \phi \in \Sigma' \mid x^\rho \in \mathrm{FV}(M)\} \cup \{x^\rho : \omega^\rho \mid x^\rho \in \mathrm{FV}(M) - \mathrm{FV}(N)\},$$

such that $\Sigma'' \vdash_1 M'' \not\sqsubseteq \Sigma \vdash_1 M'$. This contradicts the hypothesis that $\Sigma \vdash_1 M'$ is optimal.

2. We will show that, if $\Sigma \vdash_2 M'$ is the optimal faithful $\vdash_2$-typing of M, then $\mathbf{O}_2(\Sigma) \vdash_2 \mathbf{O}_2(M')$ is the optimal faithful $\vdash_2$-typing of $N = \epsilon(\mathbf{O}_2(M')) = \mathbf{O}_2^\epsilon(M)$. From Proposition 6.8.1 we have that $\mathbf{O}_2(\Sigma) \vdash_2 \mathbf{O}_2(M')$ is a faithful $\vdash_2$-typing of N. Assume, by absurd, that this is not the optimal one. This implies that the optimal faithful $\vdash_2$-typing of N, say $\Sigma' \vdash_2 N'$, is such that an ω-e-type is assigned to some subexpression of N. Then it is possible to define (starting from the derivation of $\Sigma' \vdash_2 N'$ and looking at the definition of $\mathbf{O}_2$) a faithful $\vdash_2$-typing of M, $\Sigma'' \vdash_2 M''$ (where $\Sigma'' = \Sigma' \cup \{x^\rho : \omega^\rho \mid x^\rho \in \epsilon(\Sigma) - \epsilon(\Sigma')\}$), such that $\Sigma'' \vdash_2 M'' \not\sqsubseteq \Sigma \vdash_2 M'$. This contradicts the hypothesis that $\Sigma \vdash_2 M'$ is optimal.
3. **$\mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(M)) = \mathbf{O}_1^\epsilon(\mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(M)))$.** Let $\Sigma \vdash_1 M'$ be the optimal faithful $\vdash_1$-typing of M, and let $N' = \mathbf{O}_1(M')$ and $N = \epsilon(N') = \mathbf{O}_1^\epsilon(M)$, then $\Sigma' \vdash_1 N'$, where $\Sigma' = \mathbf{O}_1(\Sigma) \cup \{\mathsf{d}^\sigma : \omega^\sigma \mid \mathsf{d}^\sigma \in \mathrm{DV}(N)\}$, is the optimal faithful $\vdash_1$-typing of N. Let $\Sigma'' \vdash_2 N''$ be the optimal faithful $\vdash_2$-typing of N, and let $P = \epsilon(\mathbf{O}_2(N'')) = \mathbf{O}_2^\epsilon(N)$. Let $\Sigma''' \vdash_1 P'$ be the optimal faithful $\vdash_1$-typing of P. Suppose, by absurd, that an ω-e-type is assigned to some subexpression of P different from d. Then it is possible to define[6] (starting from the derivation of $\Sigma_1'' \vdash_1 P'$ and looking at the definition of $\mathbf{O}_2$) a faithful $\vdash_1$-typing of N, $\Sigma'''' \vdash_1 N''''$, such that $\Sigma'''' \vdash_1 N'''' \not\sqsubseteq \Sigma' \vdash_1 N'$. This contradicts the fact that $\Sigma' \vdash_1 N'$ is optimal.

   $\mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(M)) = \mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(\mathbf{O}_2^\epsilon(M)))$. We first consider the optimal faithful typings involved on the left and right-hand side of the equality.

   (L) Let $\Sigma_\mathrm{L} \vdash_1 M'_\mathrm{L}$ be the optimal faithful $\vdash_1$-typing of M, let $N_\mathrm{L} = \mathbf{O}_1^\epsilon(M)$, and let $\Sigma'_\mathrm{L} \vdash_2 N'_\mathrm{L}$ be the optimal faithful $\vdash_2$-typing of N_L.

   (R) Let $\Sigma_\mathrm{R} \vdash_2 M'_\mathrm{R}$ be the optimal faithful $\vdash_2$-typing of M, and let $N_\mathrm{R} = \mathbf{O}_2^\epsilon(M)$. Let $\Sigma'_\mathrm{R} \vdash_1 N'_\mathrm{R}$ be the optimal faithful $\vdash_1$-typing of N_R, and let $P_\mathrm{R} = \mathbf{O}_1^\epsilon(N_\mathrm{R})$). Let $\Sigma''_\mathrm{R} \vdash_2 P'_\mathrm{R}$ be the optimal faithful $\vdash_2$-typing of P_R.

   Since $\vdash_2$ is a subsystem of $\vdash_1$, any subexpression of M that is labeled by an ω-e-type in $\Sigma_\mathrm{R} \vdash_2 M'_\mathrm{R}$ is also labeled by an ω-e-type in $\Sigma_\mathrm{L} \vdash_1 M'_\mathrm{L}$. Moreover, any subexpression of M that is not labeled by an ω-e-type in $\Sigma_\mathrm{R} \vdash_2 M'_\mathrm{R}$ corresponds to a subexpression of N_R, and if such a subexpression is labeled by an ω-e-type in $\Sigma'_\mathrm{R} \vdash_1 N'_\mathrm{R}$, then the original subexpression of M is labeled by an ω-e-type in $\Sigma_\mathrm{L} \vdash_1 M'_\mathrm{L}$. This implies that

   - any subexpression of P_R that is labeled by an ω-e-type in $\Sigma''_\mathrm{R} \vdash_2 P'_\mathrm{R}$ is labeled by an ω-e-type in $\Sigma'_\mathrm{L} \vdash_2 N'_\mathrm{L}$,

[6] Observe that, since $\vdash_2$ is a subsystem of $\vdash_1$, any subexpression of N that is labeled by an ω-e-type in $\Sigma'_1 \vdash_2 N''$ is also labeled by an ω-e-type in $\Sigma_1 \vdash_1 N'$.

- any subexpression of N_{L} that does not correspond to a subexpression of P_{R} (since the corresponding one has been removed by the application of $\mathbf{O}_2^{\epsilon}$ to M) is labeled by an ω-e-type in $\Sigma'_{\mathrm{L}} \vdash_2 N'_{\mathrm{L}}$, and
- any subexpression of N_{L} that is labeled by an ω-e-type in $\Sigma'_{\mathrm{L}} \vdash_2 N'_{\mathrm{L}}$ and that corresponds to a subexpression of P_{R} (since it has not been removed by the application of $\mathbf{O}_2^{\epsilon}$ to M) is labeled by an ω-e-type in $\Sigma''_{\mathrm{R}} \vdash_2 P'_{\mathrm{R}}$.

So we have that $\mathbf{O}_2^{\epsilon}(N_{\mathrm{L}}) = \mathbf{O}_2^{\epsilon}(P_{\mathrm{R}})$.

□

This shows that the more convenient way of using the useless-code detection and elimination techniques described in sections 5 and 6 is to apply first the simplification mapping $\mathbf{O}_1^{\epsilon}$ and then the simplification mapping $\mathbf{O}_2^{\epsilon}$ (as done in Example 6.12).

In particular, if we apply the simplification mappings in the reverse order, i.e. first $\mathbf{O}_2^{\epsilon}$ and then $\mathbf{O}_1^{\epsilon}$, we may loose some simplifications, as the following example shows.

Example 8.2

Consider the term of Example 6.12,

$$M = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, gx\rangle, (\lambda y^{\mathsf{nat}}.1)P\rangle)(\lambda z^{\mathsf{nat}}.3)Q.$$

We have that:

$$\mathbf{O}_1^{\epsilon}(M) = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, g\mathsf{d}_1^{\mathsf{nat}}\rangle, (\lambda y^{\mathsf{nat}}.1)\mathsf{d}_2^{\mathsf{nat}}\rangle)(\lambda z^{\mathsf{nat}}.3)\mathsf{d}_3^{\mathsf{nat}}$$
(*which is the term N_1 in Example 6.12*),

$$\mathbf{O}_2^{\epsilon}(M) = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, gx\rangle, 1\rangle)(\lambda z^{\mathsf{nat}}.3)Q$$
(*which is the term N_2 in Example 6.11*),

$$\mathbf{O}_1^{\epsilon}(\mathbf{O}_2^{\epsilon}(M)) = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.\lambda x^{\mathsf{nat}}.+\langle +\langle fg, g\mathsf{d}_1^{\mathsf{nat}}\rangle, 1\rangle)(\lambda z^{\mathsf{nat}}.3)\mathsf{d}_3^{\mathsf{nat}},\ \textit{and}$$

$$\mathbf{O}_2^{\epsilon}(\mathbf{O}_1^{\epsilon}(M)) = (\lambda g^{\mathsf{nat}\to\mathsf{nat}}.+\langle +\langle fg, g\mathsf{d}_1^{\mathsf{nat}}\rangle, 1\rangle)(\lambda z^{\mathsf{nat}}.3)$$
(*which is the term N in Example 6.12*).

9 Functional languages with other evaluation strategies

The useless-code detection and elimination techniques described in Section 8 is in some sense independent from the evaluation strategy of the language and so they can be applied also to call-by-value languages[7]. The key property is that in general, for any evaluation strategy, a simplified program is always observationally greater or equal than the original one.

To see that under some evaluation strategy observational equivalence is not preserved look at the following example.

[7] Like ML, Objective Caml and Scheme.

Example 9.1
Take the PCFP term

$$M = (\lambda z^{\mathsf{nat}}.3)(\mathsf{fix}\, x^{\mathsf{nat}}.x).$$

By applying the simplification mappings $\mathbf{O}_1^\epsilon$ and $\mathbf{O}_2^\epsilon$ of Section 8 we get

1. $\mathbf{O}_1^\epsilon(M) = (\lambda z^{\mathsf{nat}}.3)\mathsf{d}^{\mathsf{nat}}$, and
2. $\mathbf{O}_2^\epsilon(M) = 3$.

It is easy to see that, under the call-by-value evaluation strategy, we have that[8] the original program M is observationally less or equal than (and not equivalent to) both $\mathbf{O}_1^\epsilon(M)$ and $\mathbf{O}_2^\epsilon(M)$.

The fact that the simplified program are observationally greater or equal than the original ones is guaranteed from the fact that, for any PCFP term M, if $N = \mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(M))$, then we have that

- the evaluation of N under the call-by-name evaluation strategy never requires the evaluation of a dummy variable, and
- the evaluation of N under any evaluation strategy never requires the evaluation of an application of a dummy variable (i.e. $\mathsf{d}Q_1 \cdots Q_n$ with $n \geqslant 1$) or of a projection of a dummy variable (i.e. $\mathsf{prj}_1\mathsf{d}$ or $\mathsf{prj}_2\mathsf{d}$).

Note that, when considering terminating programs (which is the typical case for programs extracted from proof) we have that, independently from the evaluation strategy of the language, the simplified programs are observationally equivalent to the original ones.

10 Related work

In this section we give a brief account of the use of PERs in program analysis and discuss the relation between our approach and other useless-code elimination techniques.

10.1 About PERs and program analysis

The first use of PERs in program analysis is, to our knowledge, in (Hunt and Sands, 1991), where the authors present a binding-time analysis for a functional programming language which is a version of PCFP with lists. In particular, they pointed out that PERs can be used for *live-variable analysis* (which is a form of useless-code analysis). The PER model in Hunt and Sands (1991) is built on a model of the language based on Scott domains, and the analysis is specified via abstract interpretation. The PhD thesis (Hunt, 1991) shows how recursive domain equations can be solved by PERs, allowing the construction of finite lattices expressing properties of algebraic data structures.

[8] Considering the dummy variables d^σ as special constants having only the property of being convergent: $\mathsf{d}^\sigma \Downarrow \mathsf{d}^\sigma$.

The key idea in using PERs for specifying a binding-time analysis (as described in Hunt and Sands (1991)) is that of understanding Δ^ρ as the property of values that are available at compile-time, and Ω^ρ as the property of values that may not be available at compile-time. The binding-time analysis of Hunt and Sands (1991) has been expressed as a type system by a number of authors (Jensen, 1995; Hankin and Le Métayer, 1995). The corresponding type system makes use of the conjunction type operator, which is necessary to model the analysis of Hunt and Sands (1991) (see Jensen (1995) for a discussion about the relation between conjunctive types and abstract interpretation). More recently, in Abadi *et al.* (1999), PER models are used for dependency-based type systems.

We got the idea for our use of PERs from Berardi (1996). The main difference between the use of PERs of Berardi (1996) and the one of Hunt and Sands (1991) is in the underlying model of the language: Berardi (1996) consider PERs on a term model of the language, whereas Hunt and Sands (1991) use a model based on a Scott domain.

10.2 About useless-code elimination

The useless-code elimination presented in this paper is inspired by that of Berardi (1996) and Berardi and Boerio (1995). The main differences between our approach and that of Berardi (1996) and Berardi and Boerio (1995) are the programming language considered, and the algorithm that finds the useless-code in a given term. The language considered in these papers is strongly normalizing: it can be obtained from PCFP by removing the constructor fix and adding, for every type $\tau \in \mathbf{T}$, an operator rec_τ for primitive recursion from nat to τ. The algorithm described in Berardi (1996) and Berardi and Boerio (1995) is a backtracking algorithm: it finds the best simplification of a given term by trying to type simplified versions of the term without changing the original type and context. Such algorithm is rather difficult understand, and this makes its proof of correctness even more difficult to follow. In our approach instead we assign to each term a set of constraints representing all the faithful e-type assignment of the term[9]. This system has always a maximal solution corresponding to the e-typing of the term showing all the useless-code that can be proved by the corresponding e-type assignment system (the optimal faithful e-typing). An important feature of our algorithm is that it is naturally compositional (see Appendix B) while that of Berardi (1996) and Berardi and Boerio (1995) is not.

The paper by Boerio (1994) (see also Boerio (1995, Chap. 5)) Extends the technique of Berardi (1996) to the Polymorphic Second Order λ-calculus. This analysis does not consider type entailment. In Prost (2000) this work is extended to the higher order polymorphisms present in the lambda-cube systems. In Berardi and Boerio

[9] The condition of being a faithful e-type assignment (Definition 5.10) is simply the translation in our framework of the condition introduced in Berardi (1996) to find useless-code. Namely, in Berardi's type assignment system a subterm is useless-code if once removed (replaced by a dummy constant having a special type, corresponding to our ω-e-types) the type of the term is unchanged. So the fact that (in a faithful e-type assignment) the e-type of the term is in $\mathbf{L}_\delta$, reflects exactly Berardi's requirement that the global type of the term is unchanged.

(1997) (see also Boerio (1995, Chap. 3) the technique of Berardi (1996) is extended to deal with algebraic data types. Also this analysis considers a strongly normalizing language and does not make use of type entailment. The problem of extending the analysis of Berardi and Boerio (1997) by adding a type entailment relation is considered in Damiani (1999), which extends system $\vdash_1$ (in section 5 of this paper) to a version of PCF with algebraic data types.

In Damiani (2000) (see also Damiani (1998, Chap. 8)) the system $\vdash_1$ is extended by adding conjunction. A version of the system in which conjunction is restricted to rank 2 is presented in Damiani and Prost (1998) (see also Damiani (1998, Chap. 9). The addition of conjunction increases significantly the power of the analysis, but makes the inference problem more difficult.

The paper by Xi (1999) describes a technique, based on dependent types, for detecting unreachable matching clauses in typed functional programs. This work is primarily concerned with error detection, while our approach is mainly designed for program optimization. The kind of useless-code considered in Xi (1999) is orthogonal with respect to the useless-code detected by the techniques mentioned above. It could be interesting to integrate the useless-code detection for PCF with algebraic data types of Damiani (1999) with the approach based on dependent types.

Useless-variable elimination (Shivers, 1991; Wand and Siveroni, 1999; Kobayashi, 2000; Fischbach and Hannan, 1999) is a particular useless-code elimination that focuses on useless function's formal parameters. This particular form of useless-code can be removed without introducing dummy place-holders: the key idea is that *a function's actual parameter can be removed if and only if the function's formal parameter can be removed*. Both the simplification mapping $\mathbf{O}_2^\epsilon$ (see section 8 of this paper) and the algorithm described in Berardi (1996) perform essentially a useless-variable elimination. The useless-code elimination performed by the simplification mapping $\mathbf{O}_1^\epsilon$ removes more useless-code, but requires the introduction of dummy place-holders in the simplified code. As explained in section 8, the best result is obtained by applying both the two mappings (first $\mathbf{O}_1^\epsilon$ and then $\mathbf{O}_2^\epsilon$). Take, for instance the term M in Example 8.2. All the useless-code eliminations in Shivers (1991), Wand and Siveroni (1999), Kobayashi (2000) and Fishbach and Hannan (1999) can simplify M to $\mathbf{O}_2^\epsilon(M)$, but they cannot simplify it to $\mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(M))$.

The technique described in Shivers (1991) is based on control flow analysis and works on higher-order untyped functional programs. This approach has been recently reformulated and proved correct in Wand and Siveroni (1999). This method has cubic time cost in the worst case.

Both Kobayashi (2000) and Fishbach and Hannan (1999) propose type-based useless-variable elimination techniques which are closely related to ours. The useless-variable elimination proposed in Kobayashi (2000) can be described as the extension to a language with let-polymorphism of the one performed by $\mathbf{O}_2^\epsilon$. As explained in Kobayashi (2000), let-polymorphism has interesting interactions with useless-variable elimination: on one side polymorphism provides more opportunity for useless-variable elimination and, on the other, the simplified program may be more polymorphic than the original one. We are working on the extensions of both $\mathbf{O}_2^\epsilon$ and $\mathbf{O}_1^\epsilon$ to a language with let-polymorphism.

- If $\phi \in \mathbf{L}_\omega$, then $\mathbf{Ext}^\phi = \emptyset$
- If $\phi = \phi_1 \to \cdots \to \phi_n \to \delta^{\mathsf{nat}}$, $n \geqslant 0$, then $\mathbf{Ext}^\phi = \{(\lambda x^{\epsilon(\phi)}.x, \phi_1 \cdots \phi_n)\}$
- If $\phi = \phi_1 \to \cdots \to \phi_n \to \psi_1 \times \psi_2$, $n \geqslant 0$, then

$$\mathbf{Ext}^\phi = \bigcup_{i\in\{1,2\}} \{\ (\lambda x^{\epsilon(\phi)}.\lambda y_1^{\epsilon(\phi_1)}.\cdots\lambda y_n^{\epsilon(\phi_n)}.E(\mathsf{prj}_i(xy_1\cdots y_n)), \phi_1\cdots\phi_n\overrightarrow{\psi}) \mid (E,\overrightarrow{\psi}) \in \mathbf{Ext}^{\psi_i}\ \}$$

Fig. A 1. Set $\mathbf{Ext}^\phi$

The simplification in Fishbach and Hannan (1999) can be described as the extension to a language with effects of the one performed by $\mathbf{O}_2^\epsilon$. The key idea is that of integrating a simple effect analysis in the type system for detecting useless-code, marking as 'useful' the pieces of code containing side effects. Therefore, the program simplification preserves such effects.

A Proof of Lemma 4.6

In this appendix we complete the proof of completeness for the entailment relation between e-types by presenting the construction, for each e-type ϕ, of a set of pairs of terms, $\mathbf{Chr}^\phi$, characterizing membership in the set $[\![\phi]\!]$ as specified by Lemma 4.6. This construction is done by defining at the same time the set $\mathbf{Ext}^\phi$, which is a set of pairs $(E, \overrightarrow{\psi})$ where E is a closed PCFP term, and $\overrightarrow{\psi}$ is a sequence of $n \geqslant 0$ e-types $\phi_1 \cdots \phi_n$. Each pair $(E, \phi_1 \cdots \phi_n) \in \mathbf{Ext}^\phi$ identifies an occurrence of δ^{nat} in ϕ that does not occur in the left-hand side of some arrow operator. In particular E is such that, whenever applied to a term M of type $\epsilon(\phi)$ and to n terms N_i of type $\epsilon(\phi_i)$ $(1 \leqslant i \leqslant n)$, returns a value of type nat.

Both $\mathbf{Ext}^\phi$ and $\mathbf{Chr}^\phi$ are defined by structural induction on ϕ.

Definition A.1
Let ϕ be an e-type. The set of pairs $\mathbf{Ext}^\phi$ is defined according to the clauses in Fig. A 1.

It is easy to prove that: $(E, \phi_1 \cdots \phi_n) \in \mathbf{Ext}^\phi$ implies $E \in \Lambda_{\epsilon(\phi)\to\epsilon(\phi_1)\to\cdots\to\epsilon(\phi_n)\to\mathsf{nat}}$.

Example A.2
Let $\phi = (\delta^{\mathsf{nat}} \to \omega^{\mathsf{nat}} \to \delta^{\mathsf{nat}}) \times (\omega^{\mathsf{nat}} \to (\delta^{\mathsf{nat}} \times \omega^{\mathsf{nat}}))$. $\mathbf{Ext}^\phi$ is the set

$$\{\ (\lambda x^{\epsilon(\phi)}.(\lambda x'^{\mathsf{nat}\to\mathsf{nat}\to\mathsf{nat}}.x')(\mathsf{prj}_1 x),\ \delta^{\mathsf{nat}}\omega^{\mathsf{nat}})\ ,\ (\lambda x^{\epsilon(\phi)}.(\lambda x'^{\mathsf{nat}\to(\mathsf{nat}\times\mathsf{nat})}.\lambda y_1^{\mathsf{nat}}.(\lambda x''^{\mathsf{nat}}.x'')(\mathsf{prj}_1(x'y_1)))(\mathsf{prj}_2 x),\ \omega^{\mathsf{nat}})\ \}\ .$$

The set $\mathbf{Ext}^\phi$ characterizes ϕ in the following sense.

Proposition A.3
Let $\phi \in \mathbf{L}$. $([P],[Q]) \in [\![\phi]\!]$ if and only if, for all $(E, \psi_1 \cdots \psi_n) \in \mathbf{Ext}^\phi$ and for all $([M_i],[N_i]) \in [\![\psi_i]\!]$ $(1 \leqslant i \leqslant n)$,

$$EPM_1 \cdots M_n \simeq_{\mathrm{obs}} EQN_1 \cdots N_n.$$

Proof
The case $\phi \in \mathbf{L}_\omega$ is immediate. If $\phi \notin \mathbf{L}_\omega$ the proof is by structural induction on ϕ.

$\underline{\phi = \phi_1 \to \cdots \phi_n \to \delta^{\mathsf{nat}},\ n \geqslant 0}$. By Definition A.1, $\mathbf{Ext}^\phi = \{(\lambda x^{\epsilon(\phi)}.x, \phi_1 \cdots \phi_n)\}$, and, by Definition 4.2, $([P],[Q]) \in [\![\phi]\!]$ if and only if, for all $([M_i],[N_i]) \in [\![\phi_i]\!]$ $(1 \leqslant i \leqslant n)$, $([P\ M_1 \cdots M_n],[Q\ N_1 \cdots N_n]) \in [\![\delta^{\mathsf{nat}}]\!]$. Again, from Definition 4.2, this is if and only if $P\ M_1 \cdots M_n \simeq_{\mathrm{obs}} Q\ N_1 \cdots N_n$. Therefore also

$$(\lambda x^{\epsilon(\phi)}.x) P\ M_1 \cdots M_n \simeq_{\mathrm{obs}} (\lambda x^{\epsilon(\phi)}.x) Q\ N_1 \cdots N_n.$$

$\underline{\phi = \phi_1 \to \cdots \phi_n \to \psi_1 \times \psi_2,\ n \geqslant 0}$. By Definition 4.2, $([P],[Q]) \in [\![\phi]\!]$ if and only if, for all for all $([M_i],[N_i]) \in [\![\phi_i]\!]$ $(1 \leqslant i \leqslant n)$, $([P\ M_1 \cdots M_n],[Q\ N_1 \cdots N_n]) \in [\![\psi_1 \times \psi_2]\!]$. Moreover, again by Definition 4.2, $([P M_1 \cdots M_n],[QN_1 \cdots N_n]) \in [\![\psi_1 \times \psi_2]\!]$ if and only if both $[\mathsf{prj}_1(P\ M_1 \cdots M_n)] \in [\![\psi_1]\!]$ and $[\mathsf{prj}_2(Q\ N_1 \cdots N_n)] \in [\![\psi_2]\!]$. So the result follows by induction hypotheses and Definition A.1.

□

The set of characteristic pairs $\mathbf{Chr}^\phi$ is defined by induction on ϕ.

Definition A.4
Let ϕ be an e-type. The set of pairs of closed terms $\mathbf{Chr}^\phi$ is defined according to the clauses in Fig. A 2.

It is easy to prove that: $(P,Q) \in \mathbf{Chr}^\phi$ implies $P \in \Lambda_{\epsilon(\phi)}$ and $Q \in \Lambda_{\epsilon(\phi)}$.

The proof of completeness relies on the following lemma.

Lemma A.5
$\phi \not\leqslant \psi$ implies that there are $(P,Q) \in \mathbf{Chr}^\phi$ and $(E, \psi_1 \cdots \psi_n) \in \mathbf{Ext}^\psi$, such that for some $(M_1,N_1) \in \mathbf{Chr}^{\psi_1}$, …, $(M_n,N_n) \in \mathbf{Chr}^{\psi_n}$,

$$EPM_1 \cdots M_n \not\simeq_{\mathrm{obs}} EQN_1 \cdots N_n.$$

Proof
By structural induction on ϕ.

$\underline{\phi = \omega^{\rho_1 \to \cdots \to \rho_n \to \mathsf{nat}}}$. Since $\phi \not\leqslant \psi$, it must be that $\psi = \psi_1 \to \cdots \to \psi_n \to \delta^{\mathsf{nat}}$. By definition

- $\mathbf{Chr}^\phi = \{(P,Q)\}$, where $P = \lambda x_1^{\rho_1} \cdots x_n^{\rho_n}.0$ and $Q = \lambda x_1^{\rho_1} \cdots x_n^{\rho_n}.1$, and
- $\mathbf{Ext}^\psi = \{(E, \psi_1 \cdots \psi_n)\}$, where $E = \lambda x^{\epsilon(\phi)}.x$.

Therefore, for all $(M_i,N_i) \in \mathbf{Chr}^{\psi_i}$ $(1 \leqslant i \leqslant n)$, $EPM_1 \cdots M_n \not\simeq_{\mathrm{obs}} EQN_1 \cdots N_n$.

$\underline{\phi = \phi_1 \to \cdots \to \phi_n \to \delta^{\mathsf{nat}},\ n \geqslant 0}$. Since $\phi \not\leqslant \psi$ it must be that $n \geqslant 1$, $\psi = \psi_1 \to \cdots \to \psi_n \to \delta^{\mathsf{nat}}$, and, for some $j \in \{1,\ldots,n\}$, $\psi_j \not\leqslant \phi_j$. By induction hypothesis there are $(P',Q') \in \mathbf{Chr}^{\psi_j}$ and $(E', \phi_1' \cdots \phi_k') \in \mathbf{Ext}^{\phi_i}$ such that, for some $(M_1',N_1') \in \mathbf{Chr}^{\phi_1'}$, …, $(M_k',N_k') \in \mathbf{Chr}^{\phi_k'}$,

$$E'P'M_1' \cdots M_k' \not\simeq_{\mathrm{obs}} E'Q'N_1' \cdots N_k'.$$

Let $P = \lambda x_1^{\epsilon(\phi_1)} \cdots x_n^{\epsilon(\phi_n)}.E'x_iM_1' \cdots M_k'$ and $Q = \lambda x_1^{\epsilon(\phi_1)} \cdots x_n^{\epsilon(\phi_n)}.E'x_iN_1' \cdots N_k'$. It is immediate to see that $(P,Q) \in \mathbf{Chr}^\phi$. Since $\mathbf{Ext}^\psi = (\lambda x^{\epsilon(\psi)}.x, \psi_1 \cdots \psi_n)$, we have that, for any $(M_i,N_i) \in \mathbf{Chr}^{\psi_i}$ $(1 \leqslant i \neq j \leqslant n)$,

$$(\lambda x^{\epsilon(\psi)}.x) PM_1 \cdots M_{j-1} P' M_{j+1} \cdots M_n \not\simeq_{\mathrm{obs}} (\lambda x^{\epsilon(\psi)}.x) QN_1 \cdots N_{j-1} Q' N_{j+1} \cdots N_n,$$

- If $\phi = \omega^{\rho_1 \to \cdots \to \rho_n \to \mathsf{nat}}$, $n \geqslant 0$, then

$$\mathbf{Chr}^{\phi} = \{(\lambda x_1^{\rho_1} \cdots \lambda x_n^{\rho_n}.\mathsf{0}, \lambda x_1^{\rho_1} \cdots \lambda x_n^{\rho_n}.\mathsf{1})\}$$

- If $\phi = \omega^{\rho_1 \to \cdots \to \rho_n \to \sigma \times \sigma'}$, $n \geqslant 0$, then

$$\begin{aligned}\mathbf{Chr}^{\phi} = \{\ &(\lambda x_1 \cdots \lambda x_n.\langle P\, x_1 \cdots x_n, P'\, x_1 \cdots x_n\rangle,\\ &\lambda x_1 \cdots \lambda x_n.\langle Q\, x_1 \cdots x_n, Q'\, x_1 \cdots x_n\rangle) \mid\\ &(P,Q) \in \mathbf{Chr}^{\omega^{\rho_1 \to \cdots \to \rho_n \to \sigma}} \text{ and } (P',Q') \in \mathbf{Chr}^{\omega^{\rho_1 \to \cdots \to \rho_n \to \sigma'}}\ \}\end{aligned}$$

- If $\phi = \phi_1 \to \cdots \to \phi_n \to \delta^{\mathsf{nat}}$, $n \geqslant 0$, then

$$\begin{aligned}\mathbf{Chr}^{\phi} = \{&(\lambda x_1^{\epsilon(\phi_1)} \cdots \lambda x_n^{\epsilon(\phi_n)}.\mathsf{0}, \lambda x_1^{\epsilon(\phi_1)} \cdots \lambda x_n^{\epsilon(\phi_n)}.\mathsf{0})\} \cup\\ &\cup_{i \in \{1,\ldots,n\}} \{(\lambda x_1^{\epsilon(\phi_1)} \cdots \lambda x_n^{\epsilon(\phi_n)}.Ex_i M_1 \cdots M_{k_i},\\ &\lambda x_1^{\epsilon(\phi_1)} \cdots \lambda x_n^{\epsilon(\phi_n)}.Ex_i N_1 \cdots N_{k_i}) \mid\\ &(E, \phi_1^i \cdots \phi_{k_i}^i) \in \mathbf{Ext}^{\phi_i} \text{ and } \forall j \in \{1,\ldots,k_i\}.(M_j,N_j) \in \mathbf{Chr}^{\phi_j^i}\}\end{aligned}$$

- If $\phi = \phi_1 \to \cdots \to \phi_n \to \psi \times \psi'$, $n \geqslant 0$, then

$$\begin{aligned}\mathbf{Chr}^{\phi} = \{\ &(\lambda x_1 \cdots \lambda x_n.\langle P\, x_1 \cdots x_n, P'\, x_1 \cdots x_n\rangle,\\ &\lambda x_1 \cdots \lambda x_n.\langle Q\, x_1 \cdots x_n, Q'\, x_1 \cdots x_n\rangle) \mid\\ &(P,Q) \in \mathbf{Chr}^{\chi} \text{ and } (P',Q') \in \mathbf{Chr}^{\chi'}\ \}\end{aligned}$$

where

$$\chi = \begin{cases} \phi_1 \to \cdots \to \phi_n \to \psi, & \text{if } \psi \notin \mathbf{L}_\omega\\ \omega^{\epsilon(\phi_1 \to \cdots \to \phi_n \to \psi)}, & \text{if } \psi \in \mathbf{L}_\omega \end{cases}$$

$$\chi' = \begin{cases} \phi_1 \to \cdots \to \phi_n \to \psi', & \text{if } \psi' \notin \mathbf{L}_\omega\\ \omega^{\epsilon(\phi_1 \to \cdots \to \phi_n \to \psi')}, & \text{if } \psi' \in \mathbf{L}_\omega \end{cases}$$

Fig. A 2. Set $\mathbf{Chr}^{\phi}$.

which proves the result.

$\phi = \phi_1 \to \cdots \to \phi_n \to \phi' \times \phi'', n \geqslant 0$. Since $\phi \not\leqslant \psi$ it must be that $\psi = \psi_1 \to \cdots \to \psi_n \to \psi' \times \psi''$, and:

1. either $\phi_1 \to \cdots \to \phi_n \to \phi' \not\leqslant \psi_1 \to \cdots \to \psi_n \to \psi'$,
2. or $\phi_1 \to \cdots \to \phi_n \to \phi'' \not\leqslant \psi_1 \to \cdots \to \psi_n \to \psi''$.

Assume that $\phi, \phi', \psi, \psi' \notin \mathbf{L}_\omega$ and that condition (1) holds (a similar proof can be given if (2) holds). By inductive hypothesis there are $(P',Q') \in \mathbf{Chr}^{\phi_1 \to \cdots \to \phi_n \to \phi'}$ and $(E', \psi_1 \cdots \psi_n \psi_1' \cdots \psi_k') \in \mathbf{Ext}^{\psi_1 \to \cdots \to \psi_n \to \psi'}$ $(n,k \geqslant 0)$, such that for some $(M_i,N_i) \in \mathbf{Chr}^{\psi_i}$ $(1 \leqslant i \leqslant n)$ and $(M_j',N_j') \in \mathbf{Chr}^{\psi_j'}$ $(1 \leqslant j \leqslant k)$

$$E'P'M_1 \cdots M_n M_1' \cdots M_k' \not\simeq_{\mathrm{obs}} E'Q'N_1 \cdots N_n N_1' \cdots N_k'.$$

Let

- $P = \lambda x_1^{\epsilon(\phi_1)} \cdots x_n^{\epsilon(\phi_n)}.\langle (P' x_1 \cdots x_n), (P'' x_1 \cdots x_n)\rangle$, and
- $Q = \lambda x_1^{\epsilon(\phi_1)} \cdots x_n^{\epsilon(\phi_n)}.\langle (Q' x_1 \cdots x_n), (Q'' x_1 \cdots x_n)\rangle$,

for some $(P'',Q'') \in \mathbf{Chr}^{\phi_1 \to \cdots \to \phi_n \to \phi''}$. It is easy to check that $(P,Q) \in \mathbf{Chr}^{\phi}$. Let

$$(E, \overrightarrow{\psi}) = (\lambda x^{\epsilon(\psi)}.\lambda y_1^{\epsilon(\psi_1)}.\cdots \lambda y_n^{\epsilon(\psi_n)}.E'(\mathsf{prj}_1(xy_1 \cdots y_n)), \psi_1 \cdots \psi_n \psi_1' \cdots \psi_k'),$$

then $(E, \overrightarrow{\psi}) \in \mathbf{Ext}^{\psi}$. However

$$\begin{array}{lll} EPM_1 \cdots M_n M'_1 \cdots M'_k & \simeq_{\text{obs}} & E'P'M_1 \cdots M_n M'_1 \cdots M'_k, \text{ and} \\ EQN_1 \cdots N_n N'_1 \cdots N'_k & \simeq_{\text{obs}} & E'Q'N_1 \cdots N_n N'_1 \cdots N'_k, \end{array}$$

so $EPM_1 \cdots M_n M'_1 \cdots M'_k \not\simeq_{\text{obs}} EQN_1 \cdots N_n N'_1 \cdots N'_k$, and the result is proved.
$\phi = \omega^{\rho_1 \to \cdots \to \rho_n \to \sigma \times \sigma'}$. Similar.

□

From Proposition A.3 and Lemma A.5 we can prove Lemma 4.6.

Proof of Lemma 4.6

1. By induction on ϕ.
2. By absurd. Assume $\phi \not\leqslant \psi$. By Lemma A.5 there are $(P, Q) \in \mathbf{Chr}^{\phi}$ and $(E, \psi_1 \cdots \psi_n) \in \mathbf{Ext}^{\psi}$, such that for some $(M_1, N_1) \in \mathbf{Chr}^{\psi_1}$, …, $(M_n, N_n) \in \mathbf{Chr}^{\psi_n}$,
$$EPM_1 \cdots M_n \not\simeq_{\text{obs}} EQN_1 \cdots N_n.$$
However, from Point 1. we get $([M_i], [N_i]) \in [\![\psi_i]\!]$ $(1 \leqslant i \leqslant n)$, which, by Proposition A.3, implies $(P, Q) \notin [\![\psi]\!]$.

□

B An incremental useless-code detection an elimination procedure

In this appendix we show how the useless-code detection algorithms developed in section 7 can be used to build an incremental useless-code detection and elimination procedure.

B.1 $\vdash_1$ *and* $\vdash_2$ *inference in a single step*

We first show how to modify the algorithm $\mathscr{W}_1$ to infer a compact representation of both the $\vdash_1$-typings of M and the $\vdash_2$-typings of $\mathbf{O}_1^{\epsilon}(M)$ (see section 8). To do this we have to keep track of the constraints generated by uses of $\leqslant$ in the type rules for variables and recursive terms. We do it by replacing in the text of the algorithm $\mathscr{W}$ in figure 18 (but *not* in the definition of the function $\mathbf{cs}_{\leqslant}$ in figure 15) every occurrence of $\Rightarrow$ by $\Rightarrow^{\bullet}$. The 'marked connectives' $\Rightarrow^{\bullet}$ keep track of the conditional constraints associated with a potential application of the (ω) rule to some subterm, whereas the unmarked $\Rightarrow$ are associated with the uses of $\leqslant$ in rules (Var) and (Fix) of $\vdash_1$.

The e-constraints that may contain occurrences of $\Rightarrow$ marked by $\bullet$ are called $\bullet$-e-constraints. Define $\mathscr{W}^{\bullet}$ to be the algorithm obtained from the algorithm $\mathscr{W}_1$ in figure 17 by making to the algorithm $\mathscr{W}$ in figure 18 the change previously described.

As shown in section 8, the best way to use the simplifications induced by $\vdash_1$ and $\vdash_2$ on a term M is: first apply $\mathbf{O}_1^{\epsilon}$ to M and then $\mathbf{O}_2^{\epsilon}$ to the result. So the idea behind solving the $\bullet$-e-constraints associated to a term M is that, in the solution procedure, we first detect which variables are forced to be δ by the maximal solution for all the constraints (marked and unmarked), then all the marked conditional

$$(\equiv)\ \frac{\mathscr{E}[\delta/\alpha]\hookrightarrow^{\bullet}\mathscr{I}_1,\mathscr{I}_2}{\mathscr{E},\ \mathrm{cns}\hookrightarrow^{\bullet}\mathscr{I}_1\cup\{\alpha\},\mathscr{I}_2}\ \mathrm{cns}\in\{\alpha\equiv\delta,\delta\equiv\alpha\}\quad(\sqsubseteq)\ \frac{\mathscr{E}[\delta/\alpha]\hookrightarrow^{\bullet}\mathscr{I}_1,\mathscr{I}_2}{\mathscr{E},\ \alpha\sqsubseteq\delta\hookrightarrow^{\bullet}\mathscr{I}_1\cup\{\alpha\},\mathscr{I}_2}$$

$$(\Rightarrow)\ \frac{\mathscr{E}\cup\mathscr{E}''\hookrightarrow^{\bullet}\mathscr{I}_1,\mathscr{I}_2}{\mathscr{E},\ (\delta\ \mathsf{in}\ \mathscr{G})\Rightarrow^{?}\mathscr{E}''\hookrightarrow^{\bullet}\mathscr{I}_1,\mathscr{I}_2}\ \delta\in\mathscr{G}\ \text{and}\ \Rightarrow^{?}\in\{\Rightarrow,\Rightarrow^{\bullet}\}$$

$$(\mathrm{CHANGE})\ \frac{\text{no other }\hookrightarrow^{\bullet}\text{-rule can be applied}\qquad\mathbf{change}(\mathscr{E})\hookrightarrow\mathscr{I}}{\mathscr{E}\hookrightarrow^{\bullet}\emptyset,\mathscr{I}}$$

Fig. B 1. 'Natural semantics' rules for •-e-constraints solution (system $\hookrightarrow^{\bullet}$).

constraints that are trivially satisfied by the solution can be removed (in fact, since they corresponding to applications of the rule (ω), the corresponding subterm is removed by the mapping $\mathbf{O}_1^{\epsilon}$). After that, to identify which variables are forced to be δ by the optimal faithful $\vdash_2$-typing of $\mathbf{O}_1^{\epsilon}(M)$, we have to replace the constraints generated by $\mathbf{cs}_{\leqslant}$ with the constraints generated by $\mathbf{cs}_{=}$ (all the unmarked conditional constraints are generated by $\mathbf{cs}_{\leqslant}$). To this aim, define for a set of •-e-constraint $\mathscr{E}$, **change**($\mathscr{E}$) to be the set of e-constraint obtained from $\mathscr{E}$ by

- removing all the marked conditional constraints,
- 'opening' all the conditional constraints, i.e. repeatedly replacing a set of e-constraints of the form $\mathscr{E}', (\delta\ \mathsf{in}\ \mathscr{G})\Rightarrow\mathscr{E}''$ with $\mathscr{E}'\cup\mathscr{E}''$, and
- replacing every occurrence of $\sqsubseteq$ by $\equiv$.

Note that **change**($\mathscr{E}$) contains only e-constraints of the form $\zeta_1\equiv\zeta_2$, for $\zeta_1,\zeta_2\in\mathscr{A}\cup\{\delta\}$.

Let **faithful**$^{\bullet}(\Theta,\theta)$ be the set of •-e-constraints obtained from **faithful**(Θ,θ) (as defined in Theorem 7.14) by replacing each occurrence of $\Rightarrow$ by $\Rightarrow^{\bullet}$. We now define an algorithm for finding both the maximal solutions w.r.t. $\vdash_1$ and $\vdash_2$ of the •-e-constraints generated by the algorithm $\mathscr{W}^{\bullet}$. The algorithm is presented in natural semantics style, using judgements $\mathscr{E}\hookrightarrow^{\bullet}\mathscr{I}_1,\mathscr{I}_2$ (see figure B 1, where the symbol $\hookrightarrow$ in rule (CHANGE) refers to the derivation system described in figure 13.). We can see that all the rules except (CHANGE) correspond to the standard solution of the constraints of figure 13. Once finished the application of such rules the resulting set of constraints is changed and solved again. We have that, for every PCFP term M of type ρ, if $\mathscr{W}^{\bullet}(M)=\langle\Theta,M'^{\theta},\mathscr{E}\rangle$ and $\mathscr{E}\cup$ **faithful**$^{\bullet}(\Theta,\theta)\hookrightarrow^{\bullet}\mathscr{I}_1,\mathscr{I}_2$, then

1. the instantiation represented by $\mathscr{I}_1$, $\mathbf{i}_1$, is such that $\mathbf{i}_1(\Theta)\vdash_1\mathbf{i}_1(M'^{\theta})$ is the optimal faithful $\vdash_1$-typing of M, and
2. if M''^{θ} is the e-pattern decorated term obtained from M'^{θ} by replacing all subterms decorated by ω-e-types in $\mathbf{i}_1(M'^{\theta})$ with fresh dummy variables, i.e. $\epsilon(M''^{\theta})=\mathbf{O}_1(\mathbf{i}_1(M'^{\theta}))=\mathbf{O}_1^{\epsilon}(M)$, then the instantiation represented by $\mathscr{I}_1\cup\mathscr{I}_2$, $\mathbf{i}_2$, is such that

$$\mathbf{i}_2(\Theta-\{x^{\sigma}:\eta\mid\mathbf{i}_1(\eta)=\omega^{\sigma}\})\cup\{\mathsf{d}^{\sigma}:\omega^{\sigma}\mid\mathsf{d}^{\sigma}\in\mathrm{DV}(\epsilon(M''^{\theta}))\}\vdash_2\mathbf{i}_2(M''^{\theta})$$

is the optimal faithful $\vdash_2$-typing of $\epsilon(M''^{\theta})$.

Remark B.1 (*Complexity of algorithm* $\hookrightarrow^\bullet$)
According to Remark 7.11, given a finite set of $\bullet$-e-constraints $\mathscr{E}$ we can find $\mathscr{I}_1$ and $\mathscr{I}_2$ such that $\mathscr{E}\hookrightarrow^\bullet\mathscr{I}_1,\mathscr{I}_2$ in a time linear in $|\mathscr{E}|$.

B.2 Useless-code elimination on pattern-decorated terms

Let $M \in \Lambda_\rho$, $\mathscr{W}^\bullet(M) = \langle\Theta, M'^\theta, \mathscr{E}\rangle$ and $\mathscr{E} \cup \mathbf{faithful}^\bullet(\Theta,\theta)\hookrightarrow^\bullet\mathscr{I}_1,\mathscr{I}_2$. Let $\mathbf{i}_1$ and $\mathbf{i}_2$ be the instantiations represented by $\mathscr{I}_1$ and $\mathscr{I}_1 \cup \mathscr{I}_2$.

1. Define
   - $\mathbf{O}_1^{\mathscr{I}_1}(M'^\theta)$ to be the term obtained from M'^θ following Definition 5.8, that is, by *replacing* all the maximal subterms of M'^θ which are assigned an e-pattern η such that $\mathbf{i}_1(\eta) \in \mathbf{L}_\omega$ with a fresh dummy variable of type $\epsilon(\eta)$, and
   - $\mathbf{O}_1^{\mathscr{I}_1}(\Theta) = \{x : \eta \mid x : \eta \in \Sigma \text{ and } \mathbf{i}_1(\eta) \notin \mathbf{L}_\omega\}$.

   We have that $\mathbf{i}_1(\mathbf{O}_1^{\mathscr{I}_1}(\Theta)) \cup \Sigma' \vdash_1 \mathbf{i}_1(\mathbf{O}_1^{\mathscr{I}_1}(M'^\theta))$, where $\Sigma' = \{\mathsf{d}^\sigma : \omega^\sigma \mid \mathsf{d}^\sigma \in \mathrm{DV}(\epsilon(\mathbf{O}_1^{\mathscr{I}_1}(M'^\theta)))\}$.
2. Similarly, define[10]
   - $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^\theta)$ to be the (pattern decorated) term obtained from $\mathbf{O}_1^{\mathscr{I}_1}(M'^\theta)$ by removing, following Definition 6.7, the subterms which are assigned an e-pattern η such that $\mathbf{i}_2(\eta) \in \mathbf{L}_\omega$, and
   - $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta) = \mathbf{O}_1^{\mathscr{I}_1}(\Theta)$.

   Since $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta)$ contain exactly all the free non-dummy variables in $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^\theta)$ we have that $\mathbf{i}_2(\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta)) \cup \Sigma'' \vdash_2 \mathbf{i}_2(\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^\theta))$, where $\Sigma'' = \{\mathsf{d}^\sigma : \omega^\sigma \mid \mathsf{d}^\sigma \in \mathrm{DV}(\epsilon(\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^\theta)))\}$.

The following result holds.

Theorem B.2 ($\mathbf{O}_3^{\cdot\cdot}$ *computes the best simplification w.r.t.* $\vdash_1$ *and* $\vdash_2$)
Let $M \in \Lambda_\rho$, $\mathscr{W}^\bullet(M) = \langle\Theta, M'^\theta, \mathscr{E}\rangle$ and $\mathscr{E} \cup \mathbf{faithful}^\bullet(\Theta,\theta)\hookrightarrow^\bullet\mathscr{I}_1,\mathscr{I}_2$. Then $\epsilon(\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^\theta))$ is the best simplification of M that can be obtained by a combined use of the mappings $\mathbf{O}_1^\epsilon$ and $\mathbf{O}_2^\epsilon$. That is: $\epsilon(\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^\theta)) = \mathbf{O}_2^\epsilon(\mathbf{O}_1^\epsilon(M))$.

Example B.3
By applying algorithm $\mathscr{W}^\bullet$ to the term

$$N = (\lambda x^{\mathsf{nat}}.\mathsf{prj}_1\langle y^{\mathsf{nat}}, x\rangle)z^{\mathsf{nat}}$$

of Examples 7.15 and 7.17 we get $\mathscr{W}^\bullet(N) = \langle\Theta, N''^{\alpha'_1}, \mathscr{E}\rangle$, where Θ and $N''^{\alpha'_1}$ are as in Example 7.15, while

$$\mathscr{E} = \{(\delta \text{ in } \{\alpha'_1\}) \Rightarrow^\bullet \{\alpha_1 \sqsubseteq \alpha'_1\},\ (\delta \text{ in } \{\beta'_1\}) \Rightarrow^\bullet \{\beta_1 \sqsubseteq \beta'_1\},\ (\delta \text{ in } \{\beta_1\}) \Rightarrow^\bullet \{\alpha_2 \sqsubseteq \beta_1\}\}.$$

We also have

$$\mathbf{faithful}^\bullet(\Theta, \alpha'_1) = \{(\delta \text{ in } \{\alpha_1\}) \Rightarrow^\bullet \{\alpha_1 \equiv \delta\},\ (\delta \text{ in } \{\alpha_2\}) \Rightarrow^\bullet \{\alpha_2 \equiv \delta\},\ \alpha'_1 \equiv \delta\}.$$

[10] We choose the name $\mathbf{O}_3^{\cdot\cdot}$ for this simplification mapping, since if performs the 'sum' of the simplifications of the mappings $\mathbf{O}_1$ and $\mathbf{O}_2$.

Compare the sets $\mathscr{E}$ and $\mathbf{faithful}^{\bullet}(\Theta, \alpha'_1)$ above with the sets $\mathscr{E}_1$ and $\mathbf{faithful}(\Theta, \alpha'_1)$ in Example 7.15. By applying the algorithm shown in figure B 1, we get $\mathscr{E} \cup \mathbf{faithful}^{\bullet}(\Theta, \alpha'_1) \hookrightarrow^{\bullet} \mathscr{I}_1, \mathscr{I}_2$, where $\mathscr{I}_1$ and $\mathscr{I}_2$ are as in Examples 7.15 and 7.17, respectively (i.e. $\mathscr{I}_1 = \mathscr{I}_2 = \{\alpha'_1, \alpha_1\}$)[11].

Because of Theorem B.2 we can use the mapping $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}$ to remove the useless-code directly from the e-pattern decorated term $N''^{\alpha'_1}$ (without instantiating the basic variables). So we get

1. $\mathbf{O}_1^{\mathscr{I}_1}(N''^{\alpha'_1}) = ((\lambda x.\mathsf{prj}_1\langle y, \mathsf{d}_1\rangle^{\beta'_1})\mathsf{d}_2^{\beta_1})^{\alpha'_1}$,
2. $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta) = \mathbf{O}_1^{\mathscr{I}_1}(\Theta) = \{y : \alpha_1\}$, and
   $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(N''^{\alpha'_1}) = y^{\alpha'_1}$.

B.3 The procedure

We have now all the components for the definition of an incremental useless-code detection and elimination procedure on PCFP terms. Let $M \in \Lambda_\rho$, $\mathscr{W}^{\bullet}(M) = \langle \Theta, M'^{\theta}, \mathscr{E}\rangle$, $\mathscr{E} \cup \mathbf{faithful}(\Theta, \theta) \hookrightarrow^{\bullet} \mathscr{I}_1, \mathscr{I}_2$. The simplification algorithm $\mathbf{O}(M)$ returns

$$\langle \mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta), \mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^{\theta}), \mathscr{E}\rangle.$$

The term simplified is $\epsilon(\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^{\theta}))$ and the triple $\langle \mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta), \mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(M'^{\theta}), \mathscr{E}\rangle$ is all we need to perform further analysis of a program containing M, since the simplification performed by $\mathbf{O}$ are those which are possible in any context. For instance M could be applied to a term or could be itself the argument of function. In such contexts further simplifications of M could be possible.

Remark B.4 (*Complexity of the useless-code detection and elimination procedure*)
According to Remarks 7.16 and B.1 we have that the execution time of the simplification procedure is of order $\mathbf{width}(M)\,|M|$, where M is the PCFP term in input.

B.4 Constraints simplification

We now give a constraints simplification algorithm that can be used to simplify the constraints $\mathscr{E}$ produced by the algorithm $\mathbf{O}$ applied to a term M. The simplification preserves the maximum solution of $\mathscr{E}$ w.r.t. $\vdash_1$ and $\vdash_2$. This implies that, for the purpose of eliminating the useless-code in any term containing M as a subexpression, the simplified set of constraints $\mathscr{E}'$ contains the same information of the original set $\mathscr{E}$.

Proposition B.5 (*Constraints simplification algorithm*)
Let $M \in \Lambda_\rho$, $\mathscr{W}^{\bullet}(M) = \langle \Theta, M'^{\theta}, \mathscr{E}\rangle$ and $\mathscr{E} \cup \mathbf{faithful}^{\bullet}(\Theta, \theta) \hookrightarrow^{\bullet} \mathscr{I}_1, \mathscr{I}_2$. Let $\mathscr{E}'$ be obtained from $\mathscr{E}$ by performing (at the top level and, recursively, in the right-hand-side of every conditional constraints in $\mathscr{E}$) the following three simplification steps:

[11] In the simple example that we are considering we have $\mathscr{I}_1 = \mathscr{I}_2$, but in general these sets are different since the system $\vdash_1$ allows to detect more useless-code than $\vdash_2$.

1. remove all the constraints of the form $(\delta \text{ in } \mathscr{G}) \Rightarrow \mathscr{E}_0$ with $\mathscr{I}_1 \cap (\mathscr{G} \cup \{\delta\}) = \emptyset$;
2. remove all the constraints of the form either $\zeta_1 \sqsubseteq \zeta_2$ or $\zeta_1 \equiv \zeta_2$ with $\zeta_1, \zeta_2 \notin \mathscr{I}_1 \cup \mathscr{I}_2 \cup \{\delta\}$;
3. remove all the constraints of the form $(\delta \text{ in } \mathscr{G}) \Rightarrow \emptyset$.

$\mathscr{E}'$ is such that for every finite set of constraints $\mathscr{E}_0$, if $\mathscr{E} \cup \mathscr{E}_0 \hookrightarrow^\bullet \mathscr{I}'_1, \mathscr{I}'_2$ and both $\mathscr{I}'_1 \subseteq \mathscr{I}_1$ and $\mathscr{I}'_2 \subseteq \mathscr{I}_2$, then $\mathscr{E}' \cup \mathscr{E}_0 \hookrightarrow^\bullet \mathscr{I}'_1, \mathscr{I}'_2$.

Proof
First observe that, by definition of $\hookrightarrow^\bullet$, we have that both $\mathscr{I}_1 \cap \mathscr{I}_2 = \emptyset$ and $\mathscr{I}'_1 \cap \mathscr{I}'_2 = \emptyset$. Moreover, if $\mathscr{E} \cup \mathscr{E}_0 \hookrightarrow^\bullet \mathscr{I}'_1, \mathscr{I}'_2$ then, for every set $\mathscr{E}''$ obtained from $\mathscr{E}$ by removing some constraints (either at the top level or in the in the right-hand-side of some conditional constraints), we have that $\mathscr{E}'' \cup \mathscr{E}_0 \hookrightarrow^\bullet \mathscr{I}''_1, \mathscr{I}''_2$ for some $\mathscr{I}''_1 \subseteq \mathscr{I}'_1$ and $\mathscr{I}''_2 \subseteq \mathscr{I}'_2$. Since by the hypotheses $\mathscr{I}'_1 \subseteq \mathscr{I}_1$ and $\mathscr{I}'_2 \subseteq \mathscr{I}_2$ we have that $\mathscr{E}'$ is obtained from $\mathscr{E}$ by removing only constraints that do not force any basic variable in $\mathscr{I}'_1 \cup \mathscr{I}'_2$ to δ, we can conclude that, if $\mathscr{E}'' = \mathscr{E}'$, then $\mathscr{I}''_1 = \mathscr{I}'_1$ and $\mathscr{I}''_2 = \mathscr{I}'_2$. □

Example B.6
Consider the output of the inference algorithm $\mathscr{W}^\bullet$ in Example B.3, i.e. the triple $\langle \Theta, N''^{\alpha'_1}, \mathscr{E} \rangle$, and the sets $\mathscr{I}_1$ and $\mathscr{I}_2$ such that $\mathscr{E} \cup \mathbf{faithful}^\bullet(\Theta, \alpha'_1) \hookrightarrow^\bullet \mathscr{I}_1, \mathscr{I}_2$. According to Proposition B.5 we can replace $\mathscr{E}$ by the simplified set of constraint

$$\mathscr{E}' \quad = \quad \{ \ (\delta \text{ in } \{\alpha'_1\}) \Rightarrow^\bullet \{\alpha_1 \sqsubseteq \alpha'_1\} \ \}$$

without loss of information.

The triple $\langle \mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta), \mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(N''^{\alpha'_1}), \mathscr{E}' \rangle$, where $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(\Theta)$ and $\mathbf{O}_3^{\mathscr{I}_1,\mathscr{I}_2}(N''^{\alpha'_1})$ are as in Example B.3, is all we need in the analysis of programs that use M.

Remark B.7 (*Complexity of the constraint simplification*)
The constraint simplification described in Proposition B.5 can be performed in quadratic time in the length, $|\mathscr{E}|$, of the set $\mathscr{E}$ of constraints to be simplified.

Acknowledgements

We thank Stefano Berardi and Mario Coppo for discussions and comments, and David Sands for giving us pointers to related work. The first author thanks also Jean-Christophe Filliâtre for stimulating discussions on the topic of program extraction.

During the preparation of a first draft of this paper Ferruccio Damiani was visiting the Laboratoire d'Informatique de l'École Polytechnique (LIX). He would like to thank his host Radhia Cousot and the whole LIX for the ideal working conditions they provided.

References

Abadi, M., Barnerjee, A., Heintze, N. and Riecke, J. G. (1999) A Core Calculus of Dependency. *POPL'99*. ACM.

Barendsen, E. and Smetsers, S. (1995). A Derivation System for Uniqueness Typing. *Segragra'95*. Elsevier.

Benton, P. N. (1992) *Strictness Analysis of Lazy Functional Programs*. PhD thesis, University of Cambridge, Pembroke College.

Berardi, S. (1996) Pruning Simply Typed Lambda Terms. *J. Logic and Computation*, **6**(5), 663–681.

Berardi, S. and Boerio, L. (1995) Using Subtyping in Program Optimization. *TLCA'95: Lecture Notes in Computer Science 902*. Springer-Verlag.

Berardi, S. and Boerio, L. (1997) Minimum Information Code in a Pure Functional Language with Data Types. *TLCA'97: Lecture Notes in Computer Science 788*, pp. 30–45. Springer-Verlag.

Boerio, L. (1994) Extending Pruning Techniques to Polymorphic Second Order λ-calculus. *ESOP'94: Lecture Notes in Computer Science 788*, pp. 120–134. Springer-Verlag.

Boerio, L. (1995) *Optimizing Programs Extracted from Proofs*. PhD thesis, Università di Torino.

Coppo, M., Damiani, F. and Giannini, P. (1996) Refinement Types for Program Analysis. *SAS'96: Lecture Notes in Computer Science 1145*, pp. 143–158. Springer-Verlag.

Damiani, F. (1998) *Non-standard Type Inference for Functional Programs*. PhD thesis, Dipartimento di Informatica, Università di Torino.

Damiani, F. (1999) Useless-code Detection and Elimination for PCF with Algebraic Datatypes. *TLCA'99: Lecture Notes in Computer Science 1581*, pp. 83–97. Springer-Verlag.

Damiani, F. (2000) Conjunctive Types and Useless-code Elimination. *ITRS'00 Workshop*. Carleton-Scientific.

Damiani, F. and Prost, F. (1998) Detecting and Removing Dead Code using Rank 2 Intersection. *TYPES'96: Lecture Notes in Computer Science 1512*, pp. 66–87. Springer-Verlag.

Dussart, D., Henglein, F. and Mossin, C. (1995) Polymorphic Recursion and Subtype Qualifications: Polymorphic Binding-Time Analysis in Polynomial Time. *SAS'95: Lecture Notes in Computer Science 983*, pp. 118–135. Springer-Verlag.

Fischbach, A. and Hannan, J. (1999) *Type Systems and Algoritms for Useless-Variable Elimination*. Available at http://www.cse.psu.edu/~hannan.

Hankin, C. and Le Métayer, D. (1995) Lazy Type Inference and Program Analysis. *Science of Computer Programming*, **25**, 219–249.

Hunt, L. S. and Sands, D. (1991) Binding Time Analysis: A New PERspective. *Partial Evaluation and Semantics-based Program Manipulation*. ACM.

Hunt, S. (1991) *Abstract Interpretation of Functional Languages: From Theory to Practice*. PhD thesis, University of London, Imperial College.

Jensen, T. P. (1992) *Abstract Interpretation in Logical Form*. PhD thesis, University of London, Imperial College.

Jensen, T. P. (1995) Conjunctive Type Systems and Abstract Interpretation of Higher-order Functional Programs. *J. Logic and Computation*, **5**(4), 397–421.

Kahn, G. (1988) Natural semantics. In: Fuchi, K. and Nivat, M. (eds), *Programming Of Future Generation Computer*. Elsevier Science.

Kobayashi, N. (2000) Type-Based Useless Variable Elimination. *Pepm'00*. ACM.

Kuo, T. M. and Mishra, P. (1989) Strictness Analysis: a New Perspective Based on Type Inference. *Functional Programming Languages and Computer Architecture*, pp. 260–272. ACM.

Mossin, C. (1997) *Flow analysis of Typed Higher-Order Programs*. PhD thesis, DIKU, University of Copenhagen. Revised version.

Paulin-Mohring, C. (1989a) Extracting F_ω's Programs from Proofs in the Calculus of Constructions. *POPL'89*. ACM.

Paulin-Mohring, C. (1989b) *Extraction de Programme dans le Calcul des Constructions*. PhD thesis, Université Paris VII.

Pfenning, F. (1996) The Practice of Logical Frameworks. In: Kirchner, H. (ed.), *Proceedings of the Colloquium on Trees in Algebra and Programming: Lecture Notes in Computer Science 1059*, pp. 119–134. Linköping, Sweden. Springer-Verlag.

Pitts, A. M. (1997) Operationally-based Theories of Program Equivalence. In: Pitts, A. M. and Dybjer, P. (eds.), *Semantics and Logics of Computation*, pp. 241–298. Cambridge University Press.

Plotkin, G. D. (1977) LCF Considered as a Programming Language. *Theoretical Computer Science*, **5**(3), 223–255.

Plotkin, G. D. (1981) *A Structural Approach to Operational Semantics*. Technical report DAIMI FN-19, Aarhus University.

Prost, F. (2000) *A Static Calculus of Dependencies for the λ-cube. LICS'00*. IEEE Press.

Shivers, O. (1991) *Control Flow Analysis of Higher-Order Languages*. PhD thesis, Carnegie-Mellon University.

Solberg, K. L. (1995) *Annotated Type Systems for Program Analysis*. PhD thesis, Aarhus University, Denmark. Revised version.

Takayama, Y. (1991) Extraction of Redundancy-free Programs from Constructive Natural Deduction Proofs. *J. Symbolic Computation*, **12**, 29–69.

Wand, M. and Siveroni, I. (1999) Constraints Systems for Useless Variable Elimination. *POPL'99*, pp. 291–302. ACM.

Wright, D. A. (1992) *Reduction Types and Intensionality in the Lambda-Calculus*. PhD thesis, University of Tasmania.

Xi, H. (1999) Dead code Elimination Through Dependent Types. *PADL'99*, pp. 228–242.