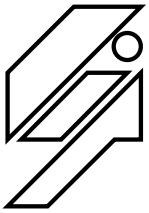
Detecting and Removing Dead Code using Rank 2 Intersection

Frederic Prost



Laboratoire de l'Informatique du Parallélisme

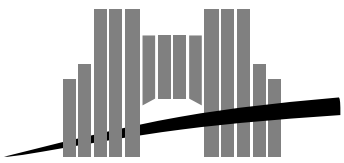
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Detecting and Removing Dead Code using Rank 2 Intersection

Ferrrucio Damiani
Frédéric Prost

Mai 97

Research Report N^{O} 97-10



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) (0)4.72.72.80.00 Télécopieur : (+33) (0)4.72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Detecting and Removing Dead Code using Rank 2 Intersection

Ferrrucio Damiani
Frédéric Prost

Mai 97

Abstract

In this paper we extend, by allowing rank 2 intersection types, the type assignment system for the detection and elimination of dead code in typed functional programs presented by Coppo et al Giannini and the first author in the *Static Analysis Symposium '96*. The main application of this method is the optimization of programs extracted from proofs in logical frameworks, but it could be used as well in the elimination of dead code determined by program specialization. This system rely on *annotated* types which allow to exploit the type structure of the language for the investigation of program properties. The detection of dead code is obtained via annotated type inference, which can be performed in a complete way, by reducing it to the solution of a system of inequalities between annotation variables. Even though the language considered in the paper is the simply typed λ-calculus with cartesian product, if-then-else, fixpoint, and arithmetic constants we can generalize our approach to polymorphic languages like Miranda, Haskell, and CAML.

Keywords: Intersection Types, Dead-Code Analysis, Annotated Types

Résumé

Dans ce papier nous étendons, en permettant des types intersections de rang 2, un système d'inférence de types pour la détection et l'élimination du code mort dans les programmes fonctionnels typés présenté par Coppo et al dans le *Static Analysis Symposium '96*. La principale application de cette méthode est l'optimisation de programmes extraits de preuves, mais il peut aussi bien être utilisé pour l'élimination du code mort produit par la spécialisation de programmes. Ce système repose sur des types *annotés* qui permettent d'exploiter la structure des types du langage pour trouver des propriétés sur un programme. La détection du code mort est obtenue via un système d'inférence de types. L'inférence peut être réalisé en réduisant le problème à la solution d'un système d'inégalités entre les variables d'annotations. Bien que le langage considéré soit le λ-calcul simplement typé étendu par le produit cartésien, le if-then-else, le point fixe et des constantes arithmétiques, nous pouvons généraliser notre approche aux langages polymorphes tels que Miranda, Haskell et CAML.

Mots-clés: Types intersection, analyse de code mort, types annotés

Detecting and Removing Dead Code using Rank 2 Intersection

Ferruccio Damiani[1] and Frédéric Prost[2]

[1] Dipartimento di Informatica, Università di Torino,
Corso Svizzera 185, 10149 Torino (Italy)
[2] Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon,
46 Allée d'Italie, 69364 Lyon Cedex 07 (France)

Abstract. In this paper we extend, by allowing rank 2 intersection types, the type assignment system for the detection and elimination of dead code in typed functional programs presented by Coppo et al in the *Static Analysis Symposium '96*. The main application of this method is the optimization of programs extracted from proofs in logical frameworks, but it could be used as well in the elimination of dead code determined by program specialization. This system rely on *annotated* types which allow to exploit the type structure of the language for the investigation of program properties. The detection of dead code is obtained via annotated type inference, which can be performed in a complete way, by reducing it to the solution of a system of inequalities between annotation variables. Even though the language considered in the paper is the simply typed λ-calculus with cartesian product, if-then-else, fixpoint, and arithmetic constants we can generalize our approach to polymorphic languages like Miranda, Haskell, and CAML.

Introduction

Types have been recognized as useful in programming languages because they provide a semantical (context dependent) analysis of programs. Such analysis can be incorporated in the compiling process. It is used on one side to check the consistency of programs and on the other to improve the efficiency of the code produced.

In addition to prevent run-time errors, type systems can characterize run-time properties of programs. For instance intersection types, see [11] (and also [1]), in their full generality, provide a characterization of normalization.

Type systems tailored to specific analysis, such as strictness, totality, binding time etc. have been introduced, see [18, 16, 2, 12, 15, 22, 25, 13]. In this perspective types represent program properties and their inference systems are systems for reasoning formally about them. In this paper we keep a clear distinction between the type structure of the language (types in the usual sense) and the *annotated* types ("non standard" types) which represent, inside the type structure of the language, particular properties. This distinction is very useful from a theoretical point of view, see [16, 2, 22], as well as in the design of both checking algorithms, see [15, 22], and inference algorithms, see [13, 14]. Type based analyzers rely on an implicit representation of types, either via type inequalities, see [19], or via lazy (implicit) types, see [15]. In this paper we pursue the first approach, reducing the annotated type inference problem to the solution of a system of inequalities between annotations on types.

Type analysis is also used in the area of program extraction from formal proof, see [7, 6, 23, 3, 5, 21]. The programs extracted from proofs are usually very inefficient, as they contain parts that are useless for the computation of the final result; they therefore require some sort of simplification. One of the more effective simplification techniques is the "pruning", and has been developed by Berardi, see [3]. In this technique useless terms (also called "dead code") are discovered by analyzing the type of terms. The

method was improved in [4] (see also [5] Chap. 4) with the use of type inclusion: an application is well typed if the argument has a type included in the input type of the corresponding function. The optimization algorithm proposed in [4] is rather difficult to understand and this makes its correctness proof even more difficult to follow.

In [10] is presented a type inference system for detecting "dead code", and an algorithm that simplifies λ-terms based on the system of [4]. The method presented in the paper is much more self-evident than the original one. The language considered is the simply typed λ-calculus with a primitive recursor over natural numbers, pairs and arithmetic constants. The idea is to start from a typed term and to decorate it by properties (called *refinement* or *annotated* types) that indicate whether or not a subterm is dead code. To this aim two annotations for the basic type nat (the type of natural numbers) are introduced. The first, δ^{nat} corresponds to the idea that the value may be used, and so could only be replaced with a term with the same behavior (observationally equivalent). The second, ω^{nat}, corresponds to the fact that the value is not used, and so it does not matter what the term is (it could be any closed term of the same type). These properties are propagated to higher types.
For instance, if a function of type $nat \to nat$ has the properties $\delta^{nat} \to \omega^{nat}$ or $\omega^{nat} \to \omega^{nat}$ then the whole term will not be used. The property $\omega^{nat} \to \delta^{nat}$, instead, informally represents the set of all the terms of type $nat \to nat$ which yield a useful output whenever applied to an argument which is not used for the computation of this output (like $\lambda x^{nat}.Q$ where x does not occur in Q). In other words, $\omega^{nat} \to \delta^{nat}$ characterizes all the functions of type $nat \to nat$ that don't use their argument. Finally, the property $\delta^{nat} \to \delta^{nat}$ does not contain any information about dead code.
The soundness of the system and of the optimizing transformation induced is proved via a partial equivalence relation semantics of the annotated types, showing that the optimized programs are observationally equivalent to the original ones.
Let us consider a simple example. Let $M = (\lambda x^{nat}.3)P$ where P is a term of type nat. Since x is not used in the body of the lambda we can assign the annotated type $\omega^{nat} \to \delta^{nat}$ to $\lambda x^{nat}.3$, so we discover that P is not useful for the computation of M and could be replaced by any constant of the right type.

In this paper, we extend the annotated type inference system of [10] by allowing rank 2 intersection (see [24]) of annotated types.
To see the usefulness of this extension, consider the term:

$$N = (\lambda f^{(nat\to nat)\to nat\to nat}.f(\lambda x^{nat}.3)P + f(\lambda y^{nat}.y)Q)\,(\lambda z^{nat\to nat}.z) \ ,$$

it is easy to see that the subterm P is dead code. To prove this by the annotated type assignment system we need to assign the annotated type $\phi_1 = (\omega^{nat} \to \delta^{nat}) \to \omega^{nat} \to \delta^{nat}$ to the first occurrence of f in the body of the λ-abstraction. On the other hand, since Q is useful to the computation of the final value of N, we are forced to assign the annotated type: $\phi_2 = (\delta^{nat} \to \delta^{nat}) \to \delta^{nat} \to \delta^{nat}$ to the the second occurrence of f in the body of the λ-abstraction.
The two annotated types ϕ_1 and ϕ_2 are not comparable using the type inclusion relation of [10], i.e., in the language of properties considered in [10] there is not a property ϕ that implies both them. So with the system of [10] it is not possible to prove that P is dead code, since for doing this is necessary to assume such a property ϕ for the λ-abstracted variable f. As we will see, the system proposed in the present paper allows to assume the intersection (or conjunction) of ϕ_1 and ϕ_2 for f, and so allows to prove that P is dead code.

The first section of this paper introduces the language we are dealing with and its semantics. Section 2 presents the rank 2 annotated type assignment system. In the third section we introduce a code simplification based on annotated type information, in particular we show that a term and its simplified version are observationally equivalent.

Section 4 presents an algorithm for inferring annotated typings of terms. The algorithm is complete, i.e., given a term, it allows to find all the dead code that can be detected by using the annotated type assignment system of Sect. 2.

1 A Typed Functional Language and its Semantics

In this section we introduce a typed functional language (basically the simply typed λ-calculus with cartesian product, if-then-else, fixpoint, and arithmetic constants) and its operational semantics. The set of types is defined assuming as basic types *nat* and *bool*: the set of naturals and the set of booleans. Types are ranged over by ρ, σ, ...

Definition1 (Types). The language of *types* (T) is defined by the following grammar: $\rho ::= \iota \mid \rho \to \rho \mid \rho \times \rho$, where $\iota \in \{nat, bool\}$.

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

$$\begin{aligned}\mathcal{K} = \{\ & 0^{nat},\ 1^{nat},\ \ldots\ succ^{nat\to nat},\ pred^{nat\to nat},\ +^{nat\times nat\to nat},\ *^{nat\times nat\to nat},\ \ldots \\ & true^{bool},\ false^{bool},\ not^{bool\to bool},\ =^{bool\times bool\to bool},\ and^{bool\times bool\to bool},\ \ldots \\ & =^{nat\times nat\to bool},\ <^{nat\times nat\to bool},\ \ldots \qquad\qquad \}\ ,\end{aligned}$$

(ranged over by C), and a set $\mathcal{V}$ of typed *term variables* (ranged over by $x^\rho, y^\sigma, \ldots$). The type of a constant C is denoted by $T(C)$. Typed terms, ranged over by M, N, ..., are defined as follows.

Definition2 (Typed terms). We write $\vdash_T M : \rho$, and say that M is a typed term of type ρ, if $\vdash M : \rho$ is derivable by the rules in Fig. 1.

$$\text{(Var)}\ \vdash x^\rho : \rho \qquad \text{(Con)}\ \vdash C^\rho : \rho$$

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

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

$$\text{(Fix)}\ \frac{\vdash M : \rho}{\vdash fix\, x^\rho.M : \rho} \qquad \text{(If)}\ \frac{\vdash N : bool \quad \vdash M_1 : \rho \quad \vdash M_2 : \rho}{\vdash if\ N\ then\ M_1\ else\ M_2 : \rho}$$

$$\text{(Case)}\ \frac{\vdash N : nat \quad \vdash M : \rho \quad \vdash F : nat \to \rho}{\vdash case(N, M, F) : \rho}$$

$$\text{(It)}\ \frac{\vdash N : nat \quad \vdash M : \rho \quad \vdash F : \rho \to \rho}{\vdash it(N, M, F) : \rho}$$

$$\text{(Rec)}\ \frac{\vdash N : nat \quad \vdash M : \rho \quad \vdash F : nat \to \rho \to \rho}{\vdash rec(N, M, F) : \rho}$$

Fig. 1. Rules for term formation

The program constructors *case*, *it* and *rec* have been included in view of an application to the optimization of terms extracted from proofs. Note that with this notation we explicitly mention in M the types of all its variables and constants. In the following we often omit to write types which are understood. The set of *free variables* of a term M, denoted by $FV(M)$, is defined in the standard way.

As usual a substitution is a finite function mapping term variables to terms, denoted by $[x_1 := N_1, \ldots, x_n := N_n]$, 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 supposed.

Let Λ_T be the set of the terms, i.e., $\Lambda_T = \{M \mid \vdash_T M : \rho \text{ for some type } \rho\}$, and Λ_T° be the set of the *closed* terms, i.e., $\Lambda_T^\circ = \{M \mid M \in \Lambda_T \text{ and } FV(M) = \emptyset\}$. Following Kahn, see [17], we define the values of terms in Λ_T° via a standard operational semantics described by judgments of the form $M \Downarrow K$, where M is a closed term and K is a closed canonical term, i.e., $K \in \mathcal{K} \cup \{\lambda x^\rho.N \mid \lambda x^\rho.N \in \Lambda_T^\circ\} \cup \{\langle M_1, M_2\rangle \mid \langle M_1, M_2\rangle \in \Lambda_T^\circ\}$. Assume that any functional constant has a type of the shape $\iota_1 \to \iota_2$ or $\iota_1 \times \iota_2 \to \iota_3$, for some $\iota_1, \iota_2, \iota_3 \in \{nat, bool\}$. The meaning of a functional constant C is given by a set $mean(C)$ of pairs, i.e., if $(P_1, P_2) \in mean(C)$ then CP_1 evaluates to P_2. For example $(5, 6) \in mean(succ)$ and $(\langle 1, 3\rangle, 4) \in mean(+)$.

Definition3 (Value of a term). We write $M \Downarrow K$ if this statement is derivable by using the rules in Fig. 2.

(CAN) $K \Downarrow K$ (FIX) $\dfrac{M[x := fix\, x.M] \Downarrow K}{fix\, x.M \Downarrow K}$

(APP) $\dfrac{M \Downarrow \lambda x.P \quad P[x := N] \Downarrow K}{MN \Downarrow K}$ (PROJ$_i$) $\dfrac{P \Downarrow \langle M_1, M_2\rangle \quad M_i \Downarrow K}{\pi_i P \Downarrow K}\ i \in \{1, 2\}$

(IF$_1$) $\dfrac{N \Downarrow true \quad M_1 \Downarrow K}{if\ N\ then\ M_1\ else\ M_2 \Downarrow K}$ (IF$_2$) $\dfrac{N \Downarrow false \quad M_2 \Downarrow K}{if\ N\ then\ M_1\ else\ M_2 \Downarrow K}$

(CASE$_1$) $\dfrac{N \Downarrow 0 \quad M \Downarrow K}{case(N, M, F) \Downarrow K}$ (CASE$_2$) $\dfrac{N \Downarrow n \quad F\, n \Downarrow K}{case(N, M, F) \Downarrow K}\ n \neq 0$

(IT$_1$) $\dfrac{N \Downarrow 0 \quad M \Downarrow K}{it(N, M, F) \Downarrow K}$ (IT$_2$) $\dfrac{N \Downarrow n \quad F(it(pred\, n, M, F)) \Downarrow K}{it(N, M, F) \Downarrow K}\ n \neq 0$

(REC$_1$) $\dfrac{N \Downarrow 0 \quad M \Downarrow K}{rec(N, M, F) \Downarrow K}$ (REC$_2$) $\dfrac{N \Downarrow n \quad F\, n\, (rec(pred\, n, M, F)) \Downarrow K}{rec(N, M, F) \Downarrow K}\ n \neq 0$

(APP$_1$) $\dfrac{M \Downarrow C \quad N \Downarrow C_1}{MN \Downarrow C_2}\ (C_1, C_2) \in mean(C)$

(APP$_2$) $\dfrac{M \Downarrow C \quad N \Downarrow \langle N_1, N_2\rangle \quad N_1 \Downarrow C_1 \quad N_2 \Downarrow C_2}{MN \Downarrow C_3}\ (\langle C_1, C_2\rangle, C_3) \in mean(C)$

Fig. 2. "Natural semantics" evaluation rules

Let $M \Downarrow$ mean that for some K, $M \Downarrow K$. We are interested in observing the behavior of terms at the ground level, so, as in Pitts [20], we consider the congruence on terms induced by the contextual preorder that compares the behavior of terms just at the ground type nat. Let $(\mathcal{C}[\ ]^\rho)^\sigma$ denote a typed context of type σ with a hole of type ρ in it. Let M and N be terms of type ρ. Define $M \preceq_{obs} N$ whenever, for all closed contexts $(\mathcal{C}[\ ]^\rho)^{nat}$, if $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are closed terms, then $\mathcal{C}[M] \Downarrow$ implies $\mathcal{C}[N] \Downarrow$. Let $\simeq_{obs}$ be the equivalence induced by $\preceq_{obs}$. (As shown in [20] such equivalence can also be defined directly as a bisimilarity.)

The *closed term model* $\mathcal{M}$ of Λ_T is defined by interpreting each type ρ as the set of the equivalence classes of the relation $\simeq_{obs}$ on the closed terms of type ρ. Let $I(\rho)$ denote the interpretation of type ρ in this model, and $[M]$ denote the equivalence class

of term M. For each type ρ, $[fix\, x^\rho.x]$ is the least element, w.r.t. $\preceq_{obs}$, of $I(\rho)$. An *environment* is a mapping $e : \mathcal{V} \to \bigcup_{\rho \in T} I(\rho)$ which respects types, i.e., such that, for each x^ρ, $e(x^\rho) \in I(\rho)$. 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 $\{x_1, \ldots, x_n\} = FV(M)$ and $[N_l] = e(x_l)$ $(1 \leq l \leq n)$.

1.1 Dummy Terms

For each type ρ, we consider a *dummy term* Ω^ρ of type ρ. Intuitively dummy terms should be considered as special terms without operational meaning. In fact, they are not present in the original programs, but (as we will show) they are introduced by the dead code elimination algorithm presented in Sect. 3, that replaces all the maximal subterms that are proved to be dead code by dummy terms of the proper type. So, each occurrence of a dummy term in a program is dead code, and this justifies the claim that dummy terms have not operational meaning: they are simply placeholders for some dead code removed.

To ensure that the output of the optimization algorithm is a well typed term, we extend the term formation rules of Fig. 1 by the following rule:

$$(\Omega)\ \frac{\rho \in T}{\vdash \Omega^\rho : \rho}\quad .$$

Remark. Despite to the claim above, for technical reasons, in the proof of the correctness of the dead code elimination algorithm O of Sect. 3 (see in particular Theorem 22), we will deal with terms containing occurrences of dummy terms that are *not* dead code. So we have to associate an operational meaning to dummy terms. This can be easily done. In fact, since the evaluation rules in Fig. 2 do not mention dummy terms, we get that, for every type ρ, $\Omega^\rho \not\Downarrow$. This means that the dummy term Ω^ρ is observationally equivalent to the divergent computation of type ρ, i.e., $[\Omega^\rho] = [fix\, x^\rho.x]$. □

2 A Type Assignment for Detecting Dead Code

In this section we introduce a (non standard) type assignment system for detecting useless code in typed terms. Starting from a typed term we want to be able to represent dead code information about this term. To this aim we define two *annotations* of the basic types: δ^ι and ω^ι $(\iota \in \{nat, bool\})$, which represent, respectively, the notion of values of type ι which are (possibly) necessary or (certainly) useless for the determination of the final value of a computation. I.e., we identify δ^ι with *(possibly) live* and ω^ι with *dead*. Annotated types are defined from $\{a^\iota \mid a \in \{\delta, \omega\} \text{ and } \iota \in \{nat, bool\}\}$ following the type construction rules. Moreover, to get more expressivity, we allow the use of intersection at rank 2.

2.1 Annotated Types

Definition 4 (Rank 0 annotated types). The language L^0 of *annotated rank 0 intersection types* (*a-0-types* for short), ranged over by ϕ, is defined by the following grammar: $\phi ::= a^\iota \mid \phi \to \phi \mid \phi \times \phi$, where $a \in \{\delta, \omega\}$ and $\iota \in \{nat, bool\}$.

Let $\epsilon(\phi)$ denote the T type obtained from the annotated type ϕ by removing all the annotations $a \in \{\delta, \omega\}$, i.e., by replacing each occurrence of δ^ι and ω^ι with ι. Moreover, if ρ is a type and $a \in \{\delta, \omega\}$, let $a(\rho)$ denote the annotated type obtained from ρ by replacing each occurrence of any basic type ι by a^ι. For instance:

$$\delta(((nat \to nat) \to nat \to nat) \to nat) = ((\delta^{nat} \to \delta^{nat}) \to \delta^{nat} \to \delta^{nat}) \to \delta^{nat}\quad .$$

Definition 5 (Rank 1 annotated types). The language L^1 of *annotated rank 1 intersection types* (*a-1-types* for short), ranged over by ξ, is defined by:

$$L^1 = \bigcup_{\rho \in T} \{\phi_1 \wedge \cdots \wedge \phi_n \mid n \geq 1,\ \phi_1, \ldots, \phi_n \in L^0,\ \text{and } \epsilon(\phi_1) = \cdots = \epsilon(\phi_n) = \rho\} \ .$$

One can note the restriction $\epsilon(\phi_1) = \cdots = \epsilon(\phi_n)$, which is not usual for standard intersection types. It intuitively corresponds to the fact that each ϕ_i represents a property of a same term. For example, the term $I = \lambda x^{nat \to nat}.x$, of type $(nat \to nat) \to nat \to nat$, can be assigned both the a-0-types $\phi_1 = (\omega^{nat} \to \delta^{nat}) \to \omega^{nat} \to \delta^{nat}$ and $\phi_2 = (\delta^{nat} \to \delta^{nat}) \to \delta^{nat} \to \delta^{nat}$. So it can be passed as argument to a function requiring an input satisfying the property $\phi_1 \wedge \phi_2$.

Definition 6 (Rank 2 annotated types). The language L^2 of *annotated rank 2 intersection types* (*a-2-types* for short), ranged over by ψ, is inductively defined by:

- $\phi \in L^2$, if $\phi \in L^0$.
- $\xi \to \psi \in L^2$, if $\xi \in L^1$ and and $\psi \in L^2$.
- $\psi_1 \times \psi_2 \in L^2$, if $\psi_1, \psi_2 \in L^2$.

Notice that $L^0 \subseteq L^1$, $L^0 \subseteq L^2$, and $L^1 \cap L^2 = L^0$.
Since a-types are properties of terms, in the following we will use the words a-type and property interchangeably. The notation $\epsilon(\cdot)$ introduced above naturally extends to a-1-types and a-2-types: $\epsilon(\xi)$ and $\epsilon(\psi)$ denote respectively the (standard) type obtained from the a-1-type ξ and the a-2-type ψ by removing all the annotations $a \in \{\delta, \omega\}$ and by keeping just the first component of each intersection. For instance:

$$\begin{aligned}\epsilon((((\delta^{nat} \to \delta^{nat}) \to \delta^{nat} \to \delta^{nat}) \wedge ((\omega^{nat} \to \delta^{nat}) \to \omega^{nat} \to \delta^{nat})) \to \delta^{nat}) = \\ ((nat \to nat) \to nat \to nat) \to nat \ .\end{aligned}$$

Intuitively, an a-2-type $\psi = \phi_1 \wedge \cdots \wedge \phi_n \to \psi' \in L^2$ such that $\epsilon(\psi) = \rho \to \rho'$ represents the set of all functional terms of type $\rho \to \rho'$ sending an input satisfying $\phi_1 \wedge \cdots \wedge \phi_n$ into an output satisfying ψ'.

The informal meaning of a-types is formalized by interpreting each a-type ψ as a partial equivalence relation (p.e.r. for short) over the interpretation of the type $\epsilon(\psi)$, i.e., the set of equivalence classes of closed terms of type $\epsilon(\psi)$ with respect to $\simeq_{obs}$. Let $\times$ denote the cartesian product of sets and $[M]$ denote the equivalence class of M in $\simeq_{obs}$.

Definition 7 (Semantics of annotated types). 1. The interpretation $[\![\psi]\!]$ of an a-2-type is defined by:

$$[\![\delta^\iota]\!] = \{\langle [N], [N] \rangle \mid [N] \in I(\iota)\} \qquad [\![\omega^\iota]\!] = I(\iota) \times I(\iota) \qquad [\![\psi_1 \times \psi_2]\!] = [\![\psi_1]\!] \times [\![\psi_2]\!]$$

$$[\![\xi \to \psi]\!] = \{\langle [M], [N] \rangle \mid \forall \langle [P], [Q] \rangle \in [\![\xi]\!].\langle [MP], [NQ] \rangle \in [\![\psi]\!]\} \ ,$$

where the interpretation $[\![\xi]\!]$ of an a-1-type $\xi = \phi_1 \wedge \cdots \wedge \phi_n$ is defined by:

$$[\![\xi]\!] = \textstyle\bigcap_{1 \leq i \leq n} [\![\phi_i]\!] \ .$$

2. By $\sim_\psi$ we denote the p.e.r. $[\![\psi]\!]$ on $I(\epsilon(\psi))$ and by $\sim_\xi$ we denote the p.e.r. $[\![\xi]\!]$ on $I(\epsilon(\xi))$.

ω-annotated types (*ω-a-types* for short) and *δ-annotated types* (*δ-a-types*) respectively formalize the notions of not being and of (possibly) being relevant to the computation, i.e., of being or (possibly) not being dead code, at higher types.

Definition 8 (δ-a-types and ω-a-types). 1. The set L^2_δ of δ-a-2-types is the subset of L^2 containing only δ annotations.
The sets L^0_δ of δ-a-0-types and L^1_δ of δ-a-1-types are defined in the same way.

2. The set L^2_ω of ω-a-2-types is inductively defined by:
   - $\omega^\iota \in L^2_\omega$, if $\iota \in \{nat, bool\}$,
   - $\xi \to \psi \in L^2_\omega$, if $\xi \in L^1$ and $\psi \in L^2_\omega$,
   - $\psi_1 \times \psi_2 \in L^2_\omega$, if $\psi_1, \psi_2 \in L^2_\omega$.

   The sets L^0_ω of ω-a-0-types is defined by $L^0_\omega = L^2_\omega \cap L^0$, and the set L^1_ω of ω-a-1-types is defined by:

$$L^1_\omega = \bigcup_{\rho \in T} \{\phi_1 \wedge \cdots \wedge \phi_n \mid n \geq 1,\ \phi_1, \ldots, \phi_n \in L^0_\omega,\ \text{and } \epsilon(\phi_1) = \cdots = \epsilon(\phi_n) = \rho\} \ .$$

Note that, if ψ is an ω-a-2-type, then $[\![\psi]\!] = I(\epsilon(\psi)) \times I(\epsilon(\psi))$, i.e., $[\![\psi]\!]$ is the p.e.r. which relates all pairs of elements of $I(\epsilon(\psi))$. The same holds for ω-1-types.

We now introduce a notion of inclusion between a-2-types, denoted $\leq_2$; $\psi_1 \leq_2 \psi_2$ means that ψ_1 is less informative then ψ_2, i.e., that $[\![\psi_1]\!] \subseteq [\![\psi_2]\!]$. The $\leq_2$ inclusion relation is defined on the top of the inclusion relation for a-0-types, $\leq_0$. This choice is justified by the key role played by the $\leq_0$ inclusion in the syntax directed a-type assignment system in Sect. 4.

Definition 9 (Inclusion relations $\leq_0$ and $\leq_2$). 1. Let $\phi_1, \phi_2 \in L^0$. We write $\phi_1 \leq_0 \phi_2$ to mean that $\phi_1 \leq_0 \phi_2$ is derivable by the rules in Fig. 3, and we write $\phi_1 \cong_0 \phi_2$ if both $\phi_1 \leq_0 \phi_2$ and $\phi_2 \leq_0 \phi_1$ hold.

2. Let $\psi_1, \psi_2 \in L^2$. We write $\psi_1 \leq_2 \psi_2$ to mean that $\psi_1 \leq_2 \psi_2$ is derivable by the rules in Fig. 4, and we write $\psi_1 \cong_2 \psi_2$ if both $\psi_1 \leq_2 \psi_2$ and $\psi_2 \leq_2 \psi_1$ hold.

(Ref_0) $\phi \leq_0 \phi$ $\qquad$ (ω_0) $\dfrac{\phi_2 \in L^0_\omega \quad \epsilon(\phi_1) = \epsilon(\phi_2)}{\phi_1 \leq_0 \phi_2}$

$(\to_0)$ $\dfrac{\phi_2 \leq_0 \phi_2' \quad \phi_1' \leq_0 \phi_1}{\phi_1 \to \phi_2 \leq_0 \phi_1' \to \phi_2'}$ $\qquad$ $(\times_0)$ $\dfrac{\phi_1 \leq_0 \phi_1' \quad \phi_2 \leq_0 \phi_2'}{\phi_1 \times \phi_2 \leq_0 \phi_1' \times \phi_2'}$

Fig. 3. Inclusion rules for a-0-types

(Ref_2) $\psi \leq_2 \psi$ $\qquad$ (ω_2) $\dfrac{\psi_2 \in L^2_\omega \quad \epsilon(\psi_1) = \epsilon(\psi_2)}{\psi_1 \leq_2 \psi_2}$

$(\to_2)$ $\dfrac{\psi \leq_2 \psi' \quad \forall i \in \{1, \ldots, m\}. \exists j \in \{1, \ldots, n\}. \phi_j' \leq_0 \phi_i}{\phi_1 \wedge \cdots \wedge \phi_m \to \psi \leq_2 \phi_1' \wedge \cdots \wedge \phi_n' \to \psi'}$ $\qquad$ $(\times_2)$ $\dfrac{\psi_1 \leq_2 \psi_1' \quad \psi_2 \leq_2 \psi_2'}{\psi_1 \times \psi_2 \leq_2 \psi_1' \times \psi_2'}$

Fig. 4. Inclusion rules for a-2-types

It is immediate to show that both $\leq_0$ and $\leq_2$ are reflexive and transitive, and that they behave in the same way on L^0, i.e., for all $\phi, \phi' \in L^0$, $\phi \leq_0 \phi'$ if and only if $\phi \leq_2 \phi'$. With $[\phi]_{\cong_0}$ we the denote the $\cong_0$-equivalence class of the a-0-type ϕ, similarly for $\cong_2$. Notice that, if ψ_1 and ψ_2 are ω-a-2-types such that $\epsilon(\psi_1) = \epsilon(\psi_2)$, then $\psi_1 \cong_2 \psi_2$. Moreover, for all $\psi_1, \psi_2 \in L^2$, $\psi_1 \leq_2 \psi_2$ implies $\epsilon(\psi_1) = \epsilon(\psi_2)$.

The $\leq_2$ relation between annotated types is sound w.r.t. the interpretation, indeed, the following theorem holds.

Theorem 10 (Soundness of $\leq_2$). *$\psi_1 \leq_2 \psi_2$ implies $[\![\psi_1]\!] \subseteq [\![\psi_2]\!]$.*

2.2 Annotated Type Assignment System

Annotated types are assigned to Λ_T terms by a set of type inference rules. If x^ρ is a term variable of type ρ, an assumption for x^ρ is an expression of the shape $x^\rho : \xi$, or $x : \xi$ for short, where $\xi \in L^1$, and $\epsilon(\xi) = \rho$. A basis is a set Σ of a-types assumptions for term variables. The functions $\epsilon(\cdot)$, $\delta(\cdot)$ and $\omega(\cdot)$ defined above are extended to bases. More precisely: $\epsilon(\Sigma) = \{x^\rho \mid x^\rho \in \Sigma\}$ is the set of term variables which occur in Σ and, for any finite set Γ of term variables, $\delta(\Gamma)$ and $\omega(\Gamma)$ denote respectively the basis $\{x^\sigma : \delta(\sigma) \mid x^\sigma \in \Gamma\}$ and $\{x^\sigma : \omega(\sigma) \mid x^\sigma \in \Gamma\}$. We will prove judgments of the form $\Sigma \vdash_L M^\psi$ where $\epsilon(M^\psi)$ is a typed term of type $\epsilon(\psi)$ whose free variables are in Σ, i.e., such that $\vdash_T \epsilon(M^\psi) : \epsilon(\psi)$ and $\epsilon(\Sigma) \supseteq FV(M)$. We use this notation since it allows to attach an a-type to all subterms of M. Note the difference with the more usual notation $\Sigma \vdash_L M : \psi$ in which this is not possible.
For each constant C an a-0-type $L(C)$, such that $\epsilon(L(C)) = T(C)$, is specified. For example, for all integers n, $L(n) = \delta^{nat}$ and $L(+) = \delta^{nat} \times \delta^{nat} \to \delta^{nat}$. In the following we require, as it is indeed natural, that $L(C) \leq_0 \phi$ implies either $\phi \cong_0 L(C)$ or $\phi \in L^0_\omega$.

Definition11 (A-type assignment system $\vdash_L$). An a-typing statement is an expression $\Sigma \vdash_L M^\psi$ where Σ is a basis containing an assumption for each free variable of M. $\Sigma, x : \phi_1 \wedge \cdots \wedge \phi_n$ denotes the basis $\Sigma \cup \{x : \phi_1 \wedge \cdots \wedge \phi_n\}$ where it is assumed that x does not appear in Σ. We write $\Sigma \vdash_L M^\psi$ to mean that $\Sigma \vdash M^\psi$ can be derived by the rules in Fig. 5.

If $\Sigma \vdash_L M^\psi$ then M^ψ has written in it the a-types assigned to its subterms. We say that M^ψ is an *annotated* term. Note that, being $\vdash_L$ an inference system, the same terms can have different annotations.

Remark. 1. Note that the $\leq_2$ inclusion relation is only used in the rules (If) and (Case). In all the other rules the $\leq_0$ inclusion suffices.
2. It is worth mentioning that, in the rule ($\to$ E), the condition

$$\psi \notin L^2_\omega \text{ implies } \forall i \in \{1, \ldots, n\}.\phi'_i \leq_0 \phi_i \ ,$$

is used instead of

$$\forall i \in \{1, \ldots, n\}.\phi'_i \leq_0 \phi_i \ .$$

This is done to take into account the fact that: if $\phi_1 \wedge \cdots \wedge \phi_n \to \psi$ is an ω-a-type then $\phi_1, \ldots, \phi_n$ can be any a-0-types such that $\epsilon(\phi_1) = \cdots \epsilon(\phi_n) = \epsilon(\phi'_1) = \cdots = \epsilon(\phi'_n)$.
3. The $\bullet$-sequence $N^{\phi'_1} \bullet \cdots \bullet N^{\phi'_n}$, in the rule ($\to$ E), is just a way of storing n decorations of the argument of an application. These decorations correspond to different uses of the argument in the function. Indeed, as pointed out in the remark at the end of Sect. 4.2, the code duplication is not necessary and can easily be avoided in the implementation of the a-type inference algorithm of Sect. 4. $\square$

The functions $\epsilon(\cdot)$, $\delta(\cdot)$, and $\omega(\cdot)$, defined for annotated types in Sect. 2.1, can naturally be extended to annotated terms. $\epsilon(M^\psi)$ in particular is simply the term M^ψ in which each $\bullet$-sequence has been replaced by its first component and all the a-type annotations have been erased. The proof of the following fact is immediate.

$$\text{(Var)}\ \frac{\phi_i \leq_0 \phi_i'}{\Sigma, x : \phi_1 \wedge \cdots \wedge \phi_n \vdash x^{\phi_i'}}\ 1 \leq i \leq n \qquad \text{(Con)}\ \frac{L(C) \leq_0 \phi}{\Sigma \vdash C^{\phi}}$$

$$(\to \text{I})\ \frac{\Sigma, x : \phi_1 \wedge \cdots \wedge \phi_n \vdash M^{\psi}}{\Sigma \vdash (\lambda x^{\phi_1 \wedge \cdots \wedge \phi_n}.M^{\psi})^{\phi_1 \wedge \cdots \wedge \phi_n \to \psi}}$$

$$(\to \text{E})\ \frac{\Sigma \vdash M^{\phi_1 \wedge \cdots \wedge \phi_n \to \psi} \quad \Sigma \vdash N^{\phi_1'} \ \cdots\ \Sigma \vdash N^{\phi_n'} \quad \forall i \in \{1, \ldots, n\}.\psi \notin L_\omega^2 \text{ implies } \phi_i' \leq_0 \phi_i}{\Sigma \vdash (M^{\phi_1 \wedge \cdots \wedge \phi_n \to \psi}(N^{\phi_1'} \bullet \cdots \bullet N^{\phi_n'}))^{\psi}}$$

$$(\times \text{I})\ \frac{\Sigma \vdash M_1{}^{\psi_1} \quad \Sigma \vdash M_2{}^{\psi_2}}{\Sigma \vdash \langle M_1{}^{\psi_1}, M_2{}^{\psi_2} \rangle^{\psi_1 \times \psi_2}} \qquad (\times \text{E}_i)\ \frac{\Sigma \vdash M^{\psi_1 \times \psi_2}}{\Sigma \vdash (\pi_i M^{\psi_1 \times \psi_2})^{\psi_i}}\ i \in \{1, 2\}$$

$$\text{(Fix)}\ \frac{\Sigma, x : \phi \vdash M^{\phi}}{\Sigma \vdash (fix\, x^{\phi}.M^{\phi})^{\phi}} \qquad \text{(If)}\ \frac{\Sigma \vdash N^{\delta^{bool}} \quad \Sigma \vdash M_1{}^{\psi_1} \quad \Sigma \vdash M_2{}^{\psi_2} \quad \psi_1 \leq_2 \psi \quad \psi_2 \leq_2 \psi}{\Sigma \vdash (if\, N^{\delta^{bool}}\, then\, M_1{}^{\psi_1}\, else\, M_2{}^{\psi_2})^{\psi}}$$

$$\text{(Case)}\ \frac{\Sigma \vdash N^{\delta^{nat}} \quad \Sigma \vdash M^{\psi_1} \quad \Sigma \vdash F^{a^{nat} \to \psi_2} \quad \psi_1 \leq_2 \psi \quad \psi_2 \leq_2 \psi}{\Sigma \vdash case(N^{\delta^{nat}}, M^{\psi_1}, F^{a^{nat} \to \psi_2})^{\psi}}$$

$$\text{(It)}\ \frac{\Sigma \vdash N^{\delta^{nat}} \quad \Sigma \vdash M^{\phi_1} \quad \Sigma \vdash F^{\phi_2 \to \phi_3} \quad \phi_1 \leq_0 \phi \quad \phi_2 \to \phi_3 \leq_0 \phi \to \phi}{\Sigma \vdash it(N^{\delta^{nat}}, M^{\phi_1}, F^{\phi_2 \to \phi_3})^{\phi}}$$

$$\text{(Rec)}\ \frac{\Sigma \vdash N^{\delta^{nat}} \quad \Sigma \vdash M^{\phi_1} \quad \Sigma \vdash F^{a^{nat} \to \phi_2 \to \phi_3} \quad \phi_1 \leq_0 \phi \quad a^{nat} \to \phi_2 \to \phi_3 \leq_0 \delta^{nat} \to \phi \to \phi}{\Sigma \vdash rec(N^{\delta^{nat}}, M^{\phi_1}, F^{a^{nat} \to \phi_2 \to \phi_3})^{\phi}}$$

Fig. 5. Rules for a-type assignment

Fact 12. *1. $\Sigma \vdash_L M^{\psi}$ implies $\vdash_T \epsilon(M^{\psi}) : \epsilon(\psi)$ and $\epsilon(\Sigma) \supseteq FV(M)$.*
2. $\vdash_T M : \rho$ implies, for $a \in \{\delta, \omega\}$, $a(FV(M)) \vdash_L a(M)$.

To state the soundness of the a-type assignment system w.r.t. the semantics we introduce the following definition.

Definition 13. 1. Two environments e_1, e_2 are Σ-related if and only if, for all $x : \phi_1 \wedge \cdots \wedge \phi_n \in \Sigma$, $e_1(x) \sim_{\phi_1 \wedge \cdots \wedge \phi_n} e_2(x)$.
2. Let $\Sigma \vdash_L M^{\psi}$ and $\Sigma \vdash_L N^{\psi}$. We write $\epsilon(M^{\psi}) \sim_{\psi}^{\Sigma} \epsilon(N^{\psi})$ to mean that for all e_1, e_2, if e_1 and e_2 are Σ-related, then $[\![\epsilon(M^{\psi})]\!]_{e_1} \sim_{\psi} [\![\epsilon(N^{\psi})]\!]_{e_2}$.

Now we can state the main theorem for p.e.r. interpretation, which is standard (in various forms) in the literature. The proof of the following theorem is by induction on terms.

Theorem 14 (Soundness of $\vdash_L$). *Let $\Sigma \vdash_L M^{\psi}$. Then $\epsilon(M^{\psi}) \sim_{\psi}^{\Sigma} \epsilon(M^{\psi})$.*

Let us now identify a subset of a-typings for which the $\sim_{\psi}^{\Sigma}$ relation implies the $\simeq_{obs}$ relation.

Definition 15 (Faithful a-type assignment). $\Sigma \vdash_L M^{\phi}$ is a *faithful* a-type assignment statement if $\phi \in L_\delta^0$, and for all $x : \phi_1 \wedge \cdots \wedge \phi_n \in \Sigma$, $n = 1$ and $\phi_1 \in L_\omega^0 \cup L_\delta^0$.

The correctness proof of the optimization mappings of Sect. 3 rely on the following theorem.

Theorem 16. *Let $\Sigma \vdash_L M^\phi$ and $\Sigma \vdash_L N^\phi$ be faithful a-typings. Then $\epsilon(M^\phi) \sim_\phi^\Sigma \epsilon(M^\phi)$ implies $\epsilon(M^\phi) \simeq_{obs} \epsilon(N^\phi)$.*

Remark. The condition of being a faithful a-type assignment is simply the translation in our framework of the condition introduced by Berardi in [3] to find dead code. Namely, in the Berardi's type assignment system a subterm is dead code if once removed (replaced by a dummy constant having a special type, corresponding to our ω-a-types) the global type of the term is unchanged. More precisely, in a faithful a-type assignment, the fact that the global a-type of the term is in L^0_δ, reflects the Berardi's requirement that all the basic types that occurs in the global type are considered as useful. □

3 Dead Code Elimination

In this section we introduce an optimization mapping O that, given an annotated term M^ψ, returns an optimized version of $\epsilon(M^\psi)$.

To define the optimization mapping we introduce, following [3], a notion of pruning and an operation of least upper bound on the set of terms Λ_T.

Definition 17 (Pruning relation). Let $\vdash_T M : \rho$ and $\vdash_T N : \rho$. We say that M is a *pruning* of N, and write $M \preceq_{prune} N$, if M can be obtained from N by replacing some subterms by dummy constants of the corresponding type.

Definition 18 (Operation *sup*). 1. Let $\vdash_T M : \rho$, $M_1 \preceq_{prune} M$, and $M_2 \preceq_{prune} M$. Then $sup(M_1, M_2)$ is the term defined by the clauses in Fig. 6.
2. Let $\vdash_T M : \rho$, $M_1 \preceq_{prune} M, \ldots, M_n \preceq_{prune} M$, $(n \geq 1)$ Then $sup(M_1)$ is M_1 and, for $n \geq 2$, $sup(M_1, \ldots, M_n)$ is short for $sup(\cdots sup(sup(M_1, M_2), M_3) \cdots, M_n)$.

Theorem 19. *Let $\vdash_T M : \rho$. The set*

$$\{M' \mid M' \preceq_{prune} M\} \ ,$$

with the order relation $\preceq_{prune}$ is a finite lattice with bottom Ω^ρ and top M. The operation sup of Definition 18 is the join of the lattice.

Let Λ_L be the set of all annotated terms which are defined according to Definition 11, i.e., $\Lambda_L = \{M^\psi \mid \Sigma \vdash_L M^\psi$ for some a-2-type ψ and basis $\Sigma\}$.

Definition 20 (Optimization mapping O on terms). 1. The function

$$O : \Lambda_L \to \Lambda_T$$

is defined by the clauses in Fig. 7.
2. If Σ is a basis then

$$O(\Sigma) = \{x^{\epsilon(\phi_1)} \mid x : \phi_1 \wedge \cdots \wedge \phi_n \in \Sigma,\ n \geq 1 \text{ and } \exists i \in \{1, \ldots, n\}.\phi_i \notin L^0_\omega\}.$$

The fact that the optimization mapping produces well typed terms is stated by the following proposition.

Proposition 21. *If $\Sigma \vdash_L M^\psi$ then $\vdash_T O(M^\psi)$ and $O(\Sigma) \supseteq FV(O(M^\psi))$.*

The following result can be proved using the a-type semantics.

$sup(M, \Omega^{\rho}) = sup(\Omega^{\rho}, M) = M^{\rho}$

$sup(C^{\rho}, C^{\rho}) = C^{\rho}$
$sup(x^{\rho}, x^{\rho}) = x^{\rho}$
$sup(\langle M_1, M_2\rangle, \langle N_1, N_2\rangle) = \langle sup(M_1, N_1), sup(M_2, N_2)\rangle$
$sup(\pi_i M, \pi_i N) = \pi_i sup(M, N), \quad$ where $i \in \{1, 2\}$
$sup(M_1 M_2, N_1 N_2) = sup(M_1, N_1) sup(M_2, N_2)$
$sup(\lambda x^{\rho}.M, \lambda x^{\rho}.N) = \lambda x^{\rho}.sup(M, N)$
$sup(fix\, x^{\rho}.M, fix\, x^{\rho}.N) = fix\, x^{\rho}.sup(M, N)$
$sup(if\, M\, then\, M_1\, else\, M_2,\ if\, N\, then\, N_1\, else\, N_2) =$
$\quad if\, sup(M, N)\, then\, sup(M_1, N_1)\, else\, sup(M_2, N_2)$
$sup(case(M, P, F), case(N, Q, G)) = case(sup(M, N), sup(P, Q), sup(F, G))$
$sup(rec(M, P, F), rec(N, Q, G)) = rec(sup(M, N), sup(P, Q), sup(F, G))$

Fig. 6. Operation *sup*

$O(M^{\psi}) = \Omega^{\epsilon(\psi)}, \quad$ if $\psi \in L^2_{\omega}$

otherwise:

$O(C^{\psi}) = C^{\epsilon(\psi)}$
$O(x^{\psi}) = x^{\epsilon(\psi)}$
$O(\langle {M_1}^{\psi_1}, {M_2}^{\psi_2}\rangle^{\psi_1 \times \psi_2}) = \langle O({M_1}^{\psi_1}), O({M_2}^{\psi_2})\rangle$
$O((\pi_i M^{\psi_1 \times \psi_2})^{\psi_i}) = \pi_i O(M^{\psi_1 \times \psi_2}), \quad$ where $i \in \{1, 2\}$
$O((M^{\phi_1 \wedge \cdots \wedge \phi_n \to \psi}(N^{\phi'_1} \bullet \cdots \bullet N^{\phi'_n}))^{\psi}) = O(M^{\phi_1 \wedge \cdots \wedge \phi_n \to \psi}) sup(O(N^{\phi'_1}), \ldots, O(N^{\phi'_n}))$
$O((\lambda x^{\phi_1 \wedge \cdots \wedge \phi_n}.M^{\psi})^{\phi_1 \wedge \cdots \wedge \phi_n \to \psi}) = \lambda x^{\epsilon(\phi_1)}.O(M^{\psi})$
$O((fix\, x^{\phi}.M^{\phi})^{\phi}) = fix\, x^{\epsilon(\phi)}.O(M^{\phi})$
$O((if\, N^{\delta^{bool}}\, then\, {M_1}^{\psi_1}\, else\, {M_2}^{\psi_2})^{\psi}) = if\, O(N^{\delta^{bool}})\, then\, O({M_1}^{\psi_1})\, else\, O({M_2}^{\psi_2})$
$O(case(N^{\delta^{nat}}, M^{\psi_1}, F^{a^{nat} \to \psi_2})^{\psi}) = case(O(N^{\delta^{nat}}), O(M^{\psi_1}), O(F^{a^{nat} \to \psi_2}))$
$O(it(N^{\delta^{nat}}, M^{\phi_1}, F^{\phi_2 \to \phi_3})^{\phi}) = it(O(N^{\delta^{nat}}), O(M^{\phi_1}), O(F^{\phi_2 \to \phi_3}))$
$O(rec(N^{\delta^{nat}}, M^{\phi_1}, F^{a^{nat} \to \phi_2 \to \phi_3})^{\phi}) = rec(O(N^{\delta^{nat}}), O(M^{\phi_1}), O(F^{a^{nat} \to \phi_2 \to \phi_3}))$

Fig. 7. Mapping O on terms

Theorem 22. *If $\Sigma \vdash_L M^{\psi}$ then, for each term N, $O(M^{\psi}) \preceq_{prune} N$ implies $\epsilon(M^{\psi}) \sim^{\Sigma}_{\psi} N$.*

Note that, since the $\preceq_{prune}$ relation is reflexive, we have in particular that $\epsilon(M^{\psi}) \sim^{\Sigma}_{\psi} O(M^{\psi})$. This result is especially interesting when the typing of M is faithful since, from the above theorem and Theorem 16, we get that if $\Sigma \vdash_L M^{\psi}$ is a faithful a-typing statement then $\epsilon(M^{\psi})$ and $O(M^{\psi})$ are observationally equivalent.

Theorem 23. *Let $\Sigma \vdash_L M^{\psi}$ be a faithful typing. Then $\epsilon(M^{\psi}) \simeq_{obs} O(M^{\psi})$.*

Example 1. Let $\vdash_T M : nat$ where $FV(M) = \{{u_1}^{nat}, {u_2}^{nat}\}$ and $M =$

$$(\lambda f^{(nat \to nat) \to nat \to nat}. \\ +\langle f\,(\lambda x^{nat}.3)\,u_1,\ f\,(\lambda y^{nat}.y)\,u_2\rangle) \\ (\lambda z^{nat \to nat}.z)\ .$$

Note that M is very similar to the term N considered in the Introduction, the only

differences are the use of the prefix notation for the operator + and the replacement of the subterm P and Q by the free variables u_1 and u_2.

Let $\phi_1 = (\omega^{nat} \to \delta^{nat}) \to \omega^{nat} \to \delta^{nat}$ and $\phi_2 = (\delta^{nat} \to \delta^{nat}) \to \delta^{nat} \to \delta^{nat}$. It is easy to check that $\Sigma \vdash_L {M'}^{\delta^{nat}}$ is a faithful a-typing, where (writing, for short, δ and ω instead of δ^{nat} and ω^{nat}): $\Sigma = \{{u_1}^{\omega}, {u_2}^{\delta}\}$ and ${M'}^{\delta} =$

$$
\begin{array}{l}
((\lambda f^{\phi_1 \wedge \phi_2}. \\
\quad (+^{\delta\times\delta\to\delta}\langle(((f^{\phi_1}(\lambda x^{\omega}.3^{\delta})^{\omega\to\delta})^{\omega\to\delta} u_1^{\omega})^{\delta}, (((f^{\phi_2}(\lambda y^{\omega}.y^{\delta})^{\delta\to\delta})^{\delta\to\delta} u_2^{\delta})^{\delta}\rangle^{\delta\times\delta})^{\delta})^{\phi_1\wedge\phi_2\to\delta} \\
\qquad\qquad ((\lambda z^{\omega\to\delta}.z^{\omega\to\delta})^{\phi_1} \\
\qquad\qquad\qquad \bullet \\
\qquad\qquad (\lambda z^{\delta\to\delta}.z^{\delta\to\delta})^{\phi_2}))^{\delta} \quad .
\end{array}
$$

Applying the O optimization mapping we get $O({M'}^{\delta}) =$

$$
\begin{array}{l}
(\lambda f^{(nat\to nat)\to nat\to nat}. \\
\qquad +\langle f\,(\lambda x^{nat}.3)\;\Omega^{nat},\; f\,(\lambda y^{nat}.y)\;u_2\rangle) \\
\qquad\qquad (\lambda z^{nat\to nat}.z) \quad ,
\end{array}
$$

where $\vdash_T O({M'}^{\delta}) : nat$, and $FV(\epsilon(O({M'}^{\delta}))) = O(\Sigma) = \{u_2^{nat}\}$. □

4 An Algorithm for Annotated Type Inference

In this section we deal with the problem of defining a complete inference algorithm for the annotated type assignment system $\vdash_L$. To this aim the main problem is to use the inference rules to detect a faithful decoration showing the maximum amount of dead code, i.e., assigning an ω-a-type to all the maximal subterms that can be proved to be dead code by the system. The application of the optimization function O is then trivial.

The algorithm rely on a syntax directed version of the a-type assignment system $\vdash_L$ which avoids free use of the assumptions and uses only the $\leq_0$ inclusion relation. To define the new system we need some preliminary notations.

Definition24 (Operation $\uplus$). Let Σ, Σ' be two basis, then $\Sigma \uplus \Sigma'$ denotes the basis

$$
\begin{array}{l}
\{x : \xi \wedge \xi' \mid x : \xi \in \Sigma \text{ and } x : \xi' \in \Sigma'\} \\
\qquad \cup \{x : \xi \mid x : \xi \in \Sigma \text{ and } x \notin \Sigma'\} \cup \{x : \xi' \mid x : \xi' \in \Sigma' \text{ and } x \notin \Sigma\} \quad .
\end{array}
$$

Definition25 (Sets $L(p)$). For every natural number p, let $L(p)$ denote the set of the a-2-types of the shape

$$\xi_1 \to \cdots \to \xi_p \to \phi \quad ,$$

where $\xi_1, \ldots, \xi_p \in L^1$ and $\phi \in L^0$.

In the judgments of the syntax directed a-type assignment system there are two basis: the first contains a set of variables for which it is allowed to assume only a-0-types (and not a-1-types), while the second contains exactly the free variables of the term that does not occur in the first one. Moreover each judgment is parameterized by a natural number p. The idea is that, if the judgment $\Upsilon; \Sigma \vdash^{(p)} M^{\psi}$ holds, then $\psi \in L(p)$.

Definition26 (Syntax directed a-type assignment system). Let Υ denote a basis containing only assumption of the shape $x : \phi$, where $\phi \in L^0$, and let p be a natural number. We write $\Upsilon; \Sigma \vdash^{(p)} M^{\psi}$ if $\Upsilon; \Sigma \vdash^{(p)} M^{\psi}$ can be derived by the rules in Fig. 8.

Fact 27. *Let $\Upsilon; \Sigma \vdash^{(p)} M^{\psi}$. Then $\psi \in L(p)$, $\epsilon(\Upsilon) \cap \epsilon(\Sigma) = \emptyset$, and $\epsilon(\Sigma) \subseteq FV(\epsilon(M^{\psi})) \subseteq \epsilon(\Upsilon) \cup \epsilon(\Sigma)$.*

The notion of faithful typing for the system $\vdash^{(p)}$ is given by the following definition.

Definition 28 (Faithful $\vdash^{(0)}$-type assignment). $\Upsilon;\emptyset \vdash^{(0)} M^{\phi}$ is a *faithful* $\vdash^{(0)}$-type assignment statement if $\phi \in L^0_\delta$, and for all $x:\phi' \in \Upsilon$, $\phi' \in L^0_\omega \cup L^0_\delta$.

The relation between the a-type assignment system $\vdash_L$ of Definition 11 and its syntact directed formulation $\vdash^{(p)}$ is stated by the following theorem.

Theorem 29. *1. $\Upsilon;\Sigma \vdash^{(p)} M^{\psi}$ implies $\Upsilon \cup \Sigma \vdash_L M^{\psi}$.*
2. Let $\vdash_T M:\rho$. Then, for each faithful a-typing of M, $\Upsilon \vdash_L M'^{\phi}$, there is a faithful $\vdash^{(0)}$-typing of M, $\Upsilon;\emptyset \vdash^{(0)} M''^{\phi}$, such that $O(M'^{\phi}) = O(M''^{\phi})$.

$$(\mathrm{Var}_1^{p\geq 0})\ \frac{\phi \leq_0 \phi'}{\Upsilon, x:\phi;\emptyset \vdash^{(p)} x^{\phi'}} \qquad (\mathrm{Var}_2^{p\geq 0})\ \frac{\phi \leq_0 \phi'}{\Upsilon;\{x:\phi\} \vdash^{(p)} x^{\phi'}}\ x \notin \Upsilon$$

$$(\mathrm{Con}^{p\geq 0})\ \frac{L(C) \leq_0 \phi}{\Upsilon;\emptyset \vdash^{(p)} C^{\phi}} \qquad (\to \mathrm{I}_1^{p=0})\ \frac{\Upsilon, x:\phi;\Sigma \vdash^{(0)} M^{\phi'}}{\Upsilon;\Sigma \vdash^{(0)} (\lambda x^{\phi}.M^{\phi'})^{\phi\to\phi'}}$$

$$(\to \mathrm{I}_2^{p\geq 1})\ \frac{\Upsilon;\Sigma, x:\phi_1\wedge\cdots\wedge\phi_n \vdash^{(p-1)} M^{\psi}}{\Upsilon;\Sigma \vdash^{(p)} (\lambda x^{\phi_1\wedge\cdots\wedge\phi_n}.M^{\psi})^{\phi_1\wedge\cdots\wedge\phi_n\to\psi}}$$

$$(\to \mathrm{I}_3^{p\geq 1})\ \frac{\Upsilon;\Sigma \vdash^{(p-1)} M^{\psi}}{\Upsilon;\Sigma \vdash^{(p)} (\lambda x^{\phi}.M^{\psi})^{\phi\to\psi}}\ x \notin \Upsilon \cup \Sigma$$

$$(\to \mathrm{E}^{p\geq 0})\ \frac{\begin{array}{c}\Upsilon;\Sigma \vdash^{(p+1)} M^{\phi_1\wedge\cdots\wedge\phi_n\to\psi} \quad \Upsilon;\Sigma_1 \vdash^{(0)} N^{\phi'_1} \ \cdots\ \Upsilon;\Sigma_n \vdash^{(0)} N^{\phi'_n} \\ \psi \notin L^2_\omega \text{ implies } \forall i\in\{1,\ldots,n\}.\phi'_i \leq_0 \phi_i\end{array}}{\Upsilon;\Sigma \uplus \Sigma_1 \uplus \cdots \uplus \Sigma_n \vdash^{(p)} (M^{\phi_1\wedge\cdots\wedge\phi_n\to\psi}(N^{\phi'_1} \bullet \cdots \bullet N^{\phi'_n}))^{\psi}}$$

$$(\times \mathrm{I}^{p\geq 0})\ \frac{\Upsilon;\Sigma_1 \vdash^{(p)} M_1^{\psi_1} \quad \Upsilon;\Sigma_1 \vdash^{(p)} M_2^{\psi_2}}{\Upsilon;\Sigma_1 \uplus \Sigma_2 \vdash^{(p)} \langle M_1^{\psi_1}, M_2^{\psi_2}\rangle^{\psi_1\times\psi_2}} \qquad (\times \mathrm{E}_i^{p\geq 0})\ \frac{\Upsilon;\Sigma \vdash^{(p)} M^{\psi_1\times\psi_2}}{\Upsilon;\Sigma \vdash^{(p)} (\pi_i M^{\psi_1\times\psi_2})^{\psi_i}}\ i\in\{1,2\}$$

$$(\mathrm{Fix}^{p\geq 0})\ \frac{\Upsilon, x:\phi;\Sigma \vdash^{(0)} M^{\phi}}{\Upsilon;\Sigma \vdash^{(p)} (fix\, x^{\phi}.M^{\phi})^{\phi}}$$

$$(\mathrm{If}^{p\geq 0})\ \frac{\begin{array}{c}\Upsilon;\Sigma \vdash^{(0)} N^{\delta^{bool}} \quad \Upsilon;\Sigma_1 \vdash^{(p)} M_1^{\xi_1\to\cdots\to\xi_p\to\phi_1} \quad \Upsilon;\Sigma_2 \vdash^{(p)} M_2^{\xi'_1\to\cdots\to\xi'_p\to\phi_2} \\ \phi_1 \leq_0 \phi \quad \phi_2 \leq_0 \phi\end{array}}{\Upsilon;\Sigma \uplus \Sigma_1 \uplus \Sigma_2 \vdash^{(p)} (if\, N^{\delta^{bool}}\, then\, M_1^{\psi_1}\, else\, M_2^{\psi_2})^{\xi_1\wedge\xi'_1\to\cdots\to\xi_p\wedge\xi'_p\to\phi}}$$

$$(\mathrm{Case}^{p\geq 0})\ \frac{\begin{array}{c}\Upsilon;\Sigma_1 \vdash^{(p)} M^{\xi_1\to\cdots\to\xi_p\to\phi_1} \quad \Upsilon;\Sigma_2 \vdash^{(p)} F^{a_1{}^{nat}\wedge\cdots\wedge a_n{}^{nat}\to\xi'_1\to\cdots\to\xi'_p\to\phi_2} \\ \Upsilon;\Sigma \vdash^{(0)} N^{\delta^{nat}} \quad a^{nat} \leq_0 a_1{}^{nat} \ \cdots\ a^{nat} \leq_0 a_n{}^{nat} \quad \phi_1 \leq_0 \phi \quad \phi_2 \leq_0 \phi\end{array}}{\Upsilon;\Sigma \uplus \Sigma_1 \uplus \Sigma_2 \vdash^{(p)} case(N^{\delta^{nat}}, M^{\psi_1}, F^{a^{nat}\to\psi_2})^{\xi_1\wedge\xi'_1\to\cdots\to\xi_p\wedge\xi'_p\to\phi}}$$

$$(\mathrm{It}^{p\geq 0})\ \frac{\begin{array}{c}\Upsilon;\Sigma \vdash^{(0)} N^{\delta^{nat}} \quad \Upsilon;\Sigma_1 \vdash^{(0)} M^{\phi_1} \quad \Upsilon;\Sigma_2 \vdash^{(0)} F^{\phi_2\to\phi_3} \\ \phi_1 \leq_0 \phi \quad \phi_2\to\phi_3 \leq_0 \phi\to\phi\end{array}}{\Upsilon;\Sigma \uplus \Sigma_1 \uplus \Sigma_2 \vdash^{(p)} it(N^{\delta^{nat}}, M^{\phi_1}, F^{\phi_2\to\phi_3})^{\phi}}$$

$$(\mathrm{Rec}^{p\geq 0})\ \frac{\begin{array}{c}\Upsilon;\Sigma \vdash^{(0)} N^{\delta^{nat}} \quad \Upsilon;\Sigma_1 \vdash^{(0)} M^{\phi_1} \quad \Upsilon;\Sigma_2 \vdash^{(0)} F^{a^{nat}\to\phi_2\to\phi_3} \\ \phi_1 \leq_0 \phi \quad a^{nat}\to\phi_2\to\phi_3 \leq_0 \delta^{nat}\to\phi\to\phi\end{array}}{\Upsilon;\Sigma \uplus \Sigma_1 \uplus \Sigma_2 \vdash^{(p)} rec(N^{\delta^{nat}}, M^{\phi_1}, F^{a^{nat}\to\phi_2\to\phi_3})^{\phi}}$$

Fig. 8. Rules for $\vdash^{(p)}$-type assignment

Using the technique described in [10], we can develop an algorithm that, given a well typed term, returns a decoration of the term containing annotation variables and a set of constraints involving annotation variables. The output of the algorithm characterizes all the possible faithful $\vdash^{(0)}$-typings of the term, more precisely: any solution of the set of constraints corresponds to a faithful $\vdash^{(0)}$-typing, and vice versa. Moreover, the set of constraints has a maximal solution, i.e., a solution corresponding to a $\vdash^{(0)}$-typing showing all the dead code that can be proved using the type assignment system $\vdash^{(0)}$. This solution can be found in an effective way.

We start by defining the notions of a-type pattern and a-type scheme.

4.1 Annotated Type Schemes

Definition 30 (Annotated type patterns). Let $\mathcal{A}$ be the set of *annotation variables*, ranged by α, β, γ, ...

1. The language P^0 of *a-0-type patterns* (*a-0-patterns* for short), ranged over by θ, is defined from the grammar of Definition 4 by replacing $a \in \{\delta, \omega\}$ by $\alpha \in \mathcal{A}$, i.e. $\theta ::= \alpha^\iota \mid \theta \to \theta \mid \theta \times \theta$, where $\alpha \in \mathcal{A}$ and $\iota \in \{nat, bool\}$.
2. The language P^1 of *a-1-type patterns* (*a-1-patterns* for short), ranged over by χ, is defined according to the clauses of Definition 5 by replacing a-0-types by a-0-patterns.
3. The language P^2 of *a-2-type patterns* (*a-2-patterns* for short), ranged over by η, is defined according to the clauses of Definition 6 by replacing a-0-types and a-1-types by a-0-patterns and a-1-patterns.

The function $\epsilon : L^0 \cup L^1 \cup L^2 \to T$ is extended in the obvious way to a-patterns.

Definition 31 (Constraints). A *constraint* is a formula of one of the following shapes:

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

where $\zeta_1, \zeta_2 \in \{\delta\} \cup \mathcal{A}$, $\mathcal{G}$ is a finite not empty subset of $\{\delta\} \cup \mathcal{A}$ and $\mathcal{E}$ is a finite set of constraints.

The symbol $\equiv$ denotes the equality on the set of annotations $\{\delta, \omega\}$, while $\sqsubseteq$ denotes the order relation defined by: $\delta \sqsubseteq \delta$, $\delta \sqsubseteq \omega$ and $\omega \sqsubseteq \omega$. A constraint is simply an equality or an inequality (between annotation variables or the constant δ), or a guarded set of constraints. For instance, the set of constraints

$$\{\; \alpha_3 \sqsubseteq \alpha_1,\; (\delta \; in \; \{\alpha_1, \alpha_2\}) \Rightarrow \{\alpha_3 \sqsubseteq \alpha_4, \alpha_5 \sqsubseteq \delta\} \;\}$$

can be read as "$\alpha_3 \sqsubseteq \alpha_1$ and if $\alpha_1 = \delta$ or $\alpha_2 = \delta$, then $\alpha_3 \sqsubseteq \alpha_4$ and $\alpha_5 \sqsubseteq \delta$".

Definition 32 (Annotated type schemes). An *a-2-type scheme* is a pair $\langle \eta, \mathcal{E} \rangle$ where η is an a-2-pattern and $\mathcal{E}$ is a finite set of constraints.

An a-2-type scheme $\langle \eta, \mathcal{E} \rangle$ represents the set of a-2-types that can be obtained from the pattern η by replacing annotation variables with annotations in such a way that the constraints in $\mathcal{E}$ are satisfied. A-types and a-typings can be obtained from patterns by instantiation.

Definition 33 (Renamings and instantiations).
1. A *renaming* is a one–to–one mapping $r : \mathcal{A} \to \mathcal{A}$.
2. An *instantiation* is a mapping $i : \mathcal{A} \to \{\delta, \omega\}$.

Both renaming and instantiation can be extended to annotation constants (by defining $i(a) = a$ and $r(a) = a$, for $a \in \{\delta, \omega\}$) and to a-types and patterns (in the obvious way). For example: $i(\alpha^{nat} \to \beta^{nat}) = i(\alpha)^{nat} \to i(\beta)^{nat}$. Of course, for any a-type $\psi \in L^2$, $i(\psi) = \psi$ and $r(\psi) = \psi$.

Definition 34. Let $\langle \eta, \mathcal{E} \rangle$ be an a-2-scheme. An instantiation i satisfies $\mathcal{E}$ if

- $\zeta_1 \equiv \zeta_2 \in \mathcal{E}$ implies $i(\zeta_1) \equiv i(\zeta_2)$, and
- $\zeta_1 \sqsubseteq \zeta_2 \in \mathcal{E}$ implies $i(\zeta_1) \sqsubseteq i(\zeta_2)$, and
- $(\delta \; in \; \mathcal{G}) \Rightarrow \mathcal{E}' \in \mathcal{E}$ implies that, if $\delta \in i(\mathcal{G})$, then i satisfies $\mathcal{E}'$.

The set of all the instantiations that satisfy $\mathcal{E}$ is denoted by $sat(\mathcal{E})$. An a-2-scheme $\langle \eta, \mathcal{E} \rangle$ represents all the a-2-types $i(\eta)$, for any $i \in sat(\mathcal{E})$.

Definition 35. Let i_1, i_2 be instantiations. We write $i_1 \sqsubseteq i_2$ if, for all $\alpha \in \mathcal{A}$, $i_1(\alpha) \sqsubseteq i_2(\alpha)$.

Fact 36. *Let $\mathcal{E}$ be a finite set of constraints. The sets $sat(\mathcal{E})$ is not empty and has a maximum element.*

Example 2. Consider the sets of constraints:

$$
\begin{array}{l}
\mathcal{E} = \{\ (\delta in \gamma_5) \Rightarrow (\ \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2 \cup \\
\qquad \{(\delta in \{\alpha_4\}) \Rightarrow \{\alpha'_8 \sqsubseteq \alpha_4, \alpha_3 \sqsubseteq \alpha'_7,\ (\delta in \{\alpha_8\}) \Rightarrow \{\alpha_2 \sqsubseteq \alpha_8, \alpha_7 \sqsubseteq \alpha_1\}\}, \\
\qquad\ (\delta in \{\beta_4\}) \Rightarrow \{\beta'_8 \sqsubseteq \beta_4, \beta_3 \sqsubseteq \beta'_7,\ (\delta in \{\beta_8\}) \Rightarrow \{\beta_2 \sqsubseteq \beta_8, \beta_7 \sqsubseteq \beta_1\}\} \quad \}\ )\ \} \\
\mathcal{E}_0 = \{\ (\delta in \gamma_5) \Rightarrow \\
\qquad \{(\delta in \{\gamma_5\}) \Rightarrow \{ \quad \gamma_3 \sqsubseteq \delta, \gamma_4 \sqsubseteq \delta\}, \\
\qquad\ (\delta in \{\alpha'_4\}) \Rightarrow \{ \quad \gamma'_1 \sqsubseteq \alpha'_3, \\
\qquad\qquad (\delta in \{\alpha'_4\}) \Rightarrow \{\ (\delta in \{\alpha'_4\}) \Rightarrow \{\ \alpha_4 \sqsubseteq \alpha'_4, \alpha'_3 \sqsubseteq \alpha_3, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\delta in \{\alpha_2\}) \Rightarrow \{\alpha_2 \sqsubseteq \alpha'_2, \alpha'_1 \sqsubseteq \alpha_1\}\} \\
\qquad\qquad\qquad\qquad\qquad (\delta in \{\alpha_2\}) \Rightarrow \{\ \alpha_6 \sqsubseteq \alpha'_2, \alpha'_1 \sqsubseteq \alpha_5\}\}\} \\
\\
\qquad\ (\delta in \{\beta'_4\}) \Rightarrow \{ \quad \gamma'_2 \sqsubseteq \beta'_3, \\
\qquad\qquad (\delta in \{\beta'_4\}) \Rightarrow \{\ (\delta in \{\beta'_4\}) \Rightarrow \{\ \beta_4 \sqsubseteq \beta'_4, \beta'_3 \sqsubseteq \beta_3, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\delta in \{\beta_2\}) \Rightarrow \{\beta_2 \sqsubseteq \beta'_2, \beta'_1 \sqsubseteq \beta_1\}\} \\
\qquad\qquad\qquad\qquad\qquad \beta_5 \sqsubseteq \beta_6, \qquad\ (\delta in \{\beta_2\}) \Rightarrow \{\beta_6 \sqsubseteq \beta'_2, \beta'_1 \sqsubseteq \beta_5\}\}\} \\
\qquad\ \alpha'_4 \sqsubseteq \gamma_3, \beta'_4 \sqsubseteq \gamma_4 \\
\quad \}\ \} \\
\mathcal{E}_1 = \{\ (\delta in \alpha_{8'}) \Rightarrow \{\alpha_8 \sqsubseteq \alpha'_8,\ \alpha'_7 \sqsubseteq \alpha_7\}\ \} \\
\mathcal{E}_2 = \{\ (\delta in \beta_{8'}) \Rightarrow \{\beta_8 \sqsubseteq \beta'_8,\ \beta'_7 \sqsubseteq \beta_7\}\ \} \\
\mathcal{E}' = \{\ (\delta in \{\gamma_1\}) \Rightarrow \{\gamma_1 \equiv \delta\}, (\delta in \{\gamma_2\}) \Rightarrow \{\gamma_2 \equiv \delta\},\ \gamma_5 \equiv \delta\ \}\ .
\end{array}
$$

To find the maximum element i of $sat(\mathcal{E} \cup \mathcal{E}')$ observe that from the last constraint of $\mathcal{E}$ we get $i(\gamma_5) = \delta$. Then from the first constraint of $\mathcal{E}_0$ we get $i(\gamma_3) = i(\gamma_4) = \delta$. By proceeding in this way we finally get that, for each $\alpha \in \mathcal{A}$, $i(\alpha) = \delta$ if and only if $\alpha \in \mathcal{I}$, where

$$
\begin{array}{l}
\mathcal{I} = \{\alpha_2, \alpha'_2, \alpha_4, \alpha'_4, \alpha_6, \alpha_8, \alpha'_8, \\
\qquad \beta_1, \beta'_1, \beta_2, \beta'_2, \beta_3, \beta'_3, \beta_4, \beta'_4, \beta_5, \beta_6, \beta_7, \beta'_7, \beta_8, \beta'_8, \\
\qquad \gamma_2, \gamma'_2, \gamma_3, \gamma_4, \gamma_5 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \} \ .
\end{array}
$$

So i defined by: $i(\alpha) = \delta$ if $\alpha \in \mathcal{I}$ and $i(\alpha) = \omega$ otherwise, is the maximum instantiation in $sat(\mathcal{E} \cup \mathcal{E}')$. □

The $\vdash^{(p)}$-type inference of a term is reduced to the solution of a finite set of constraints. A maximal instantiation then corresponds to a faithful $\vdash^{(0)}$-typing that shows the maximal amount of dead code. The algorithm for finding the maximal instantiation i that satisfies a finite set of constraints $\mathcal{E}$ is presented in natural semantics style using

judgments $\mathcal{E} \rightsquigarrow \mathcal{I}$, where $\mathcal{I}$ is the set of annotation variables that *represents* i, i.e., such that $\alpha \in \mathcal{I}$ if and only if $i(\alpha) = \delta$. The idea is simply that of recognizing, following the equalities and the inequalities, all the annotation variables that are forced to represent δ. All other annotation variables are then replaced by ω in the maximal solution.

Definition 37 (Constraints solution). Let $\mathcal{E}$ be a finite non empty set of constraints. We write $\mathcal{E} \rightsquigarrow \mathcal{I}$ to mean that this judgment is derivable by the rules in Fig. 9.

$$(STOP)\ \frac{\text{no other rule can be applied}}{\mathcal{E} \rightsquigarrow \emptyset} \qquad (GUARD)\ \frac{\mathcal{E} \cup \mathcal{E}'' \rightsquigarrow \mathcal{I} \quad \delta \in \mathcal{G}}{\mathcal{E} \cup \{(\delta\ in\ \mathcal{G}) \Rightarrow \mathcal{E}''\} \rightsquigarrow \mathcal{I}}$$

$$(\equiv)\ \frac{\{\zeta_1, \zeta_2\} = \{\alpha, \delta\} \quad \mathcal{E}[\delta/\alpha] \rightsquigarrow \mathcal{I}}{\mathcal{E} \cup \{\zeta_1 \equiv \zeta_2\} \rightsquigarrow \mathcal{I} \cup \{\alpha\}} \qquad (\sqsubseteq)\ \frac{\mathcal{E}[\delta/\alpha] \rightsquigarrow \mathcal{I}}{\mathcal{E} \cup \{\alpha \sqsubseteq \delta\} \rightsquigarrow \mathcal{I} \cup \{\alpha\}}$$

Fig. 9. "Natural semantics" rules for constraints solution

It is easy to see that, given a finite set of constraints $\mathcal{E}$, we can find $\mathcal{I}$ such that $\mathcal{E} \rightsquigarrow \mathcal{I}$ in a time linear in the number of constraints which occur in $\mathcal{E}$.

Proposition 38. *Let $\mathcal{E}$ be a finite set of constraints. Then $\mathcal{E} \rightsquigarrow \mathcal{I}$ if and only if $\mathcal{I}$ represents the maximum of $sat(\mathcal{E})$.*

4.2 An Algorithm to Infer Annotated Types

To define the algorithm we need some preliminary notations. By $newa()$ we denote a 0-ary function that, whenever called, returns a fresh annotation variable.
Let ρ be a type. By $fresh(\rho)$ we denote an a-0-pattern obtained from ρ by annotating each occurrence of any basic type in ρ with a fresh annotation variable. For example: $fresh(nat \to nat) = \alpha^{nat} \to \beta^{nat}$. For a set of term variables Γ, $fresh(\Gamma) = \{x : fresh(\rho) \mid x^\rho \in \Gamma\}$.
The function $vars$ maps an a-2-pattern η to its finite set of annotation variables. For example: $vars(\alpha^{nat} \to \beta^{nat}) = \{\alpha, \beta\}$.
The function $tail$, that maps a-2-patterns and a-2-types (not containing ω) to finite subsets of $\{\delta\} \cup \mathcal{A}$, is inductively defined by: $tail(\zeta^\iota) = \{\zeta\}$ (for $\zeta \in \{\delta\} \cup \mathcal{A}$), $tail(\eta_1 \times \eta_2) = tail(\eta_1) \cup tail(\eta_2)$, and $tail(\chi \to \eta) = tail(\eta)$.
Let θ, θ' be a-0-patterns or a-0-types (not containing ω) such that $\epsilon(\theta) = \epsilon(\theta')$, $cs_=(\theta, \theta')$, $cs_{\leq_0}(\theta, \theta')$ and $ucs_{\leq_0}(\theta, \theta')$ denote the constraints sets inductively defined by the clauses in Fig. 10. We have that for all instances i:

- $i(\theta) = i(\theta')$ if and only if $i \in sat(cs_=(\theta, \theta'))$, and
- $i(\theta) \leq_0 i(\theta')$ if and only if $i \in sat(cs_{\leq_0}(\theta, \theta'))$, and
- $i(\theta') \notin L^0_\omega$ implies $i(\theta) \leq_0 i(\theta')$ if and only if $i \in sat(ucs_{\leq_0}(\theta, \theta'))$.

Note that $ucs_{\leq_0}$ is just an auxiliary function, it has been introduced to simplify the set of constraints generated by the function $cs_{\leq_0}$. More precisely the auxiliary function is used to avoid to introduce, in the right part of a guarded constraint, some guards that are always satisfied.
For each constant C an a-0-scheme $ats(C)$ is specified. For example, for any integer n, $ats(n) = \langle \omega^{nat}, \emptyset \rangle$ and $ats(+) = \langle {\alpha_1}^{nat} \to {\alpha_2}^{nat} \to \beta^{nat}, \{\{\beta\} \Rightarrow \{\{\alpha_1 \sqsubseteq \delta, \alpha_2 \sqsubseteq \delta\}\}\} \rangle$.

$$cs_{=}(\zeta_1{}^{\iota}, \zeta_2{}^{\iota}) = \{\zeta_1 \equiv \zeta_2\}, \ \textit{where } \zeta_1, \zeta_2 \in \{\delta\} \cup \mathcal{A}$$
$$cs_{=}(\theta_1 \times \theta_2, \theta_1' \times \theta_2') = cs_{=}(\theta_1, \theta_1') \cup cs_{=}(\theta_2, \theta_2')$$
$$cs_{=}(\theta_1 \to \theta_2, \theta_1' \to \theta_2') = cs_{=}(\theta_1, \theta_1') \cup cs_{=}(\theta_2, \theta_2')$$

$$cs_{\leq_0}(\zeta_1{}^{\iota}, \zeta_2{}^{\iota}) = \{\zeta_1 \sqsubseteq \zeta_2\}, \ \textit{where } \zeta_1, \zeta_2 \in \{\delta\} \cup \mathcal{A}$$
$$cs_{\leq_0}(\theta_1 \times \theta_2, \theta_1' \times \theta_2') = cs_{\leq_0}(\theta_1, \theta_1') \cup cs_{\leq_0}(\theta_2, \theta_2')$$
$$cs_{\leq_0}(\theta_1 \to \cdots \to \theta_n \to \theta, \theta_1' \to \cdots \to \theta_n' \to \theta') =$$
$$\{(\delta \textit{ in } tail(\theta')) \Rightarrow (ucs_{\leq_0}(\theta, \theta') \cup \textstyle\bigcup_{1 \leq l \leq n} cs_{\leq_0}(\theta_l', \theta_l))\},$$
$$\textit{where } n \geq 1 \textit{ and } \theta, \theta' \textit{ are not arrow a-patterns or arrow a-types.}$$

$$ucs_{\leq_0}(\theta_1 \to \cdots \to \theta_n \to \theta, \theta_1' \to \cdots \to \theta_n' \to \theta') =$$
$$cs_{\leq_0}(\theta, \theta') \cup \textstyle\bigcup_{1 \leq l \leq n} cs_{\leq_0}(\theta_l', \theta_l),$$
$$\textit{where } n \geq 0 \textit{ and } \theta, \theta' \textit{ are not arrow a-patterns or arrow a-types.}$$

Fig. 10. Functions $cs_{=}$, $cs_{\leq_0}$ and $ucs_{\leq_0}$

Definition 39 (Sets $P(p)$). For every natural number p, let $P(p)$ denote the set of the a-2-patterns of the shape

$$\chi_1 \to \cdots \to \chi_p \to \theta \ ,$$

where $\chi_1, \ldots, \chi_p \in P^1$ and $\theta \in P^0$.

Consider the rules (If$^{p \geq 0}$) and (Case$^{p \geq 0}$) in Fig. 8, and the a-2-types $\psi_1 = \xi_1 \to \cdots \to \xi_p \to \phi_1$, $\psi_2 = \xi_1' \to \cdots \to \xi_p' \to \phi_2$, and $\psi = \xi_1 \wedge \xi_1' \to \cdots \to \xi_p \wedge \xi_p' \to \phi$ that occur in these rules. Let $\eta_1, \eta_2 \in P(p)$ be a-2-patterns corresponding respectively to $\psi_1, \psi_2 \in L(p)$. Then $\mathcal{J}(p, \eta_1, \eta_2)$, where $\mathcal{J}$ is the algorithm in Fig. 11, returns an a-2-pattern $\eta \in P(p)$ and a set of constraints $\mathcal{E}$ that characterize the a-2-type ψ. More precisely, the following proposition holds.

Proposition 40. *Let $\eta_1, \eta_2 \in P(p)$, $\epsilon(\eta_1) = \epsilon(\eta_2) = \rho$, and $\langle \eta, \mathcal{E} \rangle = \mathcal{J}(p, \eta_1, \eta_2)$. Then*

1. *$\eta \in P(p)$, and*
2. *for every instantiation $i \in sat(\mathcal{E})$, $i(\eta_1) \leq_2 i(\eta)$ and $i(\eta_2) \leq_2 i(\eta)$, and*
3. *for every instantiation i and a-2-type ψ such that $i(\eta_1) \leq_2 \psi$ and $i(\eta_2) \leq_2 \psi$, there is an instantiation $i' \in sat(\mathcal{E})$ such that $i'(\eta_1) = i(\eta_1)$, $i'(\eta_2) = i(\eta_2)$, and $i'(\eta) \leq_2 \psi$.*

$\mathcal{J}(0, \theta, \theta') =$ let $\theta'' = fresh(\epsilon(\theta))$
in $\langle \theta'', cs_{\leq_0}(\theta, \theta'') \cup cs_{\leq_0}(\theta', \theta'') \rangle$ end

$\mathcal{J}(p, \eta, \eta') =$ case $\langle \eta, \eta' \rangle$ of
$\langle \alpha_1{}^{\iota}, \alpha_2{}^{\iota} \rangle$: let $\beta = newa()$
in $\langle \beta, \{\alpha_1 \sqsubseteq \beta, \ \alpha_2 \sqsubseteq \beta\} \rangle$ end
$\langle \chi \to \eta, \chi' \to \eta' \rangle$: let $\langle \eta'', \mathcal{E} \rangle = \mathcal{J}(p-1, \eta, \eta')$
in $\langle \chi \wedge \chi' \to \eta'', \mathcal{E} \rangle$ end
$\langle \eta_1 \times \eta_2, \eta_1' \times \eta_2' \rangle$: let $\langle \eta_1'', \mathcal{E}_1 \rangle = \mathcal{J}(p, \eta_1, \eta_1')$ and $\langle \eta_2'', \mathcal{E}_2 \rangle = \mathcal{J}(p, \eta_2, \eta_2')$
in $\langle \eta_1'' \times \eta_2'', \mathcal{E}_1 \cup \mathcal{E}_2 \rangle$ end

Fig. 11. Algorithm $\mathcal{J}$

We can now proceed to define the annotated type inference algorithm $\mathcal{W}$. This algorithm is presented in Fig. 12, 13 and 14. Let $\vdash_T M : \rho$, if $\mathcal{W}(M) = \langle \Theta, M'^{\theta}, \mathcal{E} \rangle$ then Θ is a basis that associates to each term variable in $FV(M)$ an a-0-pattern, M'^{θ} is a term annotated with a-patterns, and $\mathcal{E}$ is a finite set of constraints. We will prove that $\langle \Theta, M'^{\theta}, \mathcal{E} \rangle$ represents all the $\vdash^{(0)}$-typings of M. More precisely, for any Υ and M''^{ϕ} such that $\epsilon(\Upsilon) = FV(M)$ and $\epsilon(M''^{\phi}) = M$, we have that $\Upsilon; \emptyset \vdash^{(0)} M''^{\phi}$ implies $\Upsilon = i(\Theta)$ and $M''^{\phi} = i(M'^{\theta})$, for some i that satisfies $\mathcal{E}$.

$$
\begin{array}{lll}
\mathcal{W}(P) = & \text{let} & \Theta = fresh(FV(P)) \\
 & & \text{and } \langle \emptyset, P', \mathcal{E} \rangle = \mathcal{W}^{\star}(0, \Theta, P) \\
 & \text{in} & \langle \Theta, P', \mathcal{E} \rangle \ \text{ end}
\end{array}
$$

Fig. 12. Algorithm $\mathcal{W}$

Correctness and completeness of the inference w.r.t. $\vdash^{(p)}$-typings containing as many ω annotations as possible is expressed by the following lemma.

Lemma 41. *$\vdash_T M : \rho$, $\Gamma \subseteq FV(M)$, $\Theta = fresh(\Gamma)$, and $\mathcal{W}^{\star}(p, \Theta, M) = \langle \Xi, M'^{\eta}, \mathcal{E} \rangle$ implies*

1. *if i is the maximum of $sat(\mathcal{E})$, then $i(\Theta); i(\Xi) \vdash^{(p)} i(M'^{\eta})$, and*
2. *for all Υ, Σ and M''^{ψ} such that $\epsilon(\Upsilon) = \Gamma$, $\epsilon(\Sigma) = \epsilon(\Xi)$ and $\epsilon(M''^{\psi}) = M$, if $\Upsilon; \Sigma \vdash^{(p)} M''^{\psi}$ then exists $i \in sat(\mathcal{E})$ such that $i(\Theta) = \Upsilon$, $i(\Xi) = \Sigma$ and $i(M'^{\eta}) = M''^{\psi}$.*

We are interested in faithful $\vdash^{(0)}$-typings, so we want to restrict the set of solutions of the constraints generated by the algorithm to those that correspond to faithful a-typings. This can be done as shown by the following theorem.

Theorem 42. *Let $\vdash_T M : \rho$ and $\mathcal{W}(M) = \langle \Theta, M'^{\theta}, \mathcal{E} \rangle$. If i is the maximum of $sat(\mathcal{E} \cup faithful(\Theta, \theta))$ then $i(\Theta); \emptyset \vdash^{(0)} i(M'^{\theta})$ is a faithful assignment showing the maximum amount of dead code, where $faithful(\Theta, \theta) =$*

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

The constraint $(\delta \ in \ tail(\theta')) \Rightarrow \{\gamma \equiv \delta | \gamma \in vars(\theta')\}$ means that θ' must be instantiated either to an ω-a-0-type or to an δ-a-0-type.

Example 3. Let $\vdash_T M : \rho$ be the typed term of Example 1. Let $\theta_1 = (\alpha_1 \to \alpha_2) \to \alpha_3 \to \alpha_4$, $\theta_1' = (\alpha_1' \to \alpha_2') \to \alpha_3' \to \alpha_4'$, $\theta_2 = (\beta_1 \to \beta_2) \to \beta_3 \to \beta_4$, and $\theta_2' = (\beta_1' \to \beta_2') \to \beta_3' \to \beta_4'$. Then $\mathcal{W}(M) = \langle \Theta, M'^{\gamma_5 \ {}^{nat}}, \mathcal{E} \rangle$ where (writing, for short, ζ instead of ζ^{nat}): $\Theta = \{u_1 : \gamma_1, u_2 : \gamma_2\}$, $M'^{\theta} =$

$$
\begin{array}{l}
((\lambda f^{\theta_1 \wedge \theta_2}. \\
\quad (+^{\gamma_3 \times \gamma_4 \to \gamma_5} \langle (((f^{\theta_1'}(\lambda x^{\alpha_5}.3^{\alpha_6})^{\alpha_5 \to \alpha_6})^{\alpha_3' \to \alpha_4'} u_1^{\gamma_1})^{\alpha_4'}, \\
\qquad\qquad ((f^{\theta_2'}(\lambda y^{\beta_5}.y^{\beta_6})^{\beta_5 \to \beta_6})^{\beta_3' \to \beta_4'} u_2^{\gamma_2})^{\beta_4'})^{\alpha_4' \times \beta_4'})^{\gamma_5})^{(\theta_1 \wedge \theta_2) \to \gamma_5} \\
\qquad\qquad\qquad ((\lambda z^{\alpha_7 \to \alpha_8}.z^{\alpha_7' \to \alpha_8'})^{(\alpha_7 \to \alpha_8) \to \alpha_7' \to \alpha_8'} \\
\qquad\qquad\qquad\qquad \bullet \\
\qquad\qquad\qquad (\lambda z^{\beta_7 \to \beta_8}.z^{\beta_7' \to \beta_8'})^{(\beta_7 \to \beta_8) \to \beta_7' \to \beta_8'}))^{\gamma_5} \ ,
\end{array}
$$

$\mathcal{W}^\star(p, \Theta, C) =$
  let $\langle \theta, \mathcal{E} \rangle = ats(C)$
  in $\langle \emptyset, C^\theta, \mathcal{E} \rangle$ end
$\mathcal{W}^\star(p, \Theta, x^\rho) =$
  let $\theta = fresh(\rho)$ and $\theta' = fresh(\rho)$
  in if $x \in \epsilon(\Theta)$
    then $\langle \emptyset, x^{\theta'}, cs_{\leq_0}(\Theta(x), \theta') \rangle$
    else $\langle \{x : \theta\}, x^{\theta'}, cs_{\leq_0}(\theta, \theta') \rangle$
  end
$\mathcal{W}^\star(p, \Theta, \lambda\, x^\rho.M) =$
  let $\theta = fresh(\rho)$ in if $p = 0$
    then let $\langle \Xi, M'^{\theta'}, \mathcal{E} \rangle = \mathcal{W}^\star(0, \Theta \cup \{x : \theta\}, M)$
      in $\langle \Xi, (\lambda\, x^\theta.M'^{\theta'})^{\theta \to \theta'}, \mathcal{E} \rangle$ end
    else let $\langle \Xi, M'^\eta, \mathcal{E} \rangle = \mathcal{W}^\star(p-1, \Theta, M)$
QUI   in case Ξ of
        $\Xi', x : \chi$: $\langle \Xi', (\lambda\, x^\chi.M'^\eta)^{\chi \to \eta}, \mathcal{E} \rangle$
        _: $\langle \Xi, (\lambda\, x^\theta.M'^\eta)^{\theta \to \eta}, \mathcal{E} \rangle$ end
  end
$\mathcal{W}^\star(p, \Theta, MN) =$
  let $\langle \Xi_0, M'^{\theta_1 \wedge \cdots \wedge \theta_n \to \eta}, \mathcal{E}_0 \rangle = \mathcal{W}^\star(p+1, \Theta, M)$
    and $\langle \Xi, N'^\theta, \mathcal{E} \rangle = \mathcal{W}^\star(0, \Theta, N)$
    and, for each $l \in \{1, \ldots, n\}$, $\langle \Xi_l, N_l^{\theta'_l}, \mathcal{E}_l \rangle = r_l(\langle \Xi, N'^\theta, \mathcal{E} \rangle)$,
      where r_l is a fresh renaming of all the annotation variables not in Θ
  in $\langle \Xi_0 \uplus \Xi_1 \uplus \cdots \uplus \Xi_n,$
    $(M'^{\theta_1 \wedge \cdots \wedge \theta_n \to \eta}(N'^{\theta'_1} \bullet \cdots \bullet N'^{\theta'_n}))^\eta,$
    $\{(\delta\ in\ tail(\eta)) \Rightarrow (\mathcal{E}_0 \cup \bigcup_{1 \leq l \leq n}(\mathcal{E}_l \cup cs_{\leq_0}(\theta'_l, \theta_l)))\}\rangle$ end
$\mathcal{W}^\star(p, \Theta, \langle M_1, M_2 \rangle) =$
  let $\langle \Xi_1, M'^{\eta_1}_1, \mathcal{E}_1 \rangle = \mathcal{W}^\star(p, \Theta, M_1)$
    and $\langle \Xi_2, M'^{\eta_2}_2, \mathcal{E}_2 \rangle = \mathcal{W}^\star(p, \Theta, M_2)$
  in $\langle \Xi_1 \uplus \Xi_2, (\langle M'^{\eta_1}_1, M'^{\eta_2}_2 \rangle)^{\eta_1 \times \eta_2}, \mathcal{E}_1 \cup \mathcal{E}_2 \rangle$ end
$\mathcal{W}^\star(p, \Theta, \pi_i M) =$
  let $\langle \Xi, M'^{\eta_1 \times \eta_2}, \mathcal{E} \rangle = \mathcal{W}^\star(p, \Theta, M)$
  in $\langle \Xi, (\pi_i M'^{\eta_1 \times \eta_2})^{\eta_i}, \mathcal{E} \rangle$ end
$\mathcal{W}^\star(p, \Theta, fix\ x^\rho.M) =$
  let $\theta_1 = fresh(\rho)$
    and $\langle \Xi, M'^{\theta_2}, \mathcal{E} \rangle = \mathcal{W}^\star(0, \Theta \cup \{x : \theta_1\}, M)$
  in $\langle \Xi, (fix\, x^\theta.M'^\theta)^\theta, \mathcal{E} \cup cs_=(\theta_1, \theta_2) \rangle$ end
$\mathcal{W}^\star(p, \Theta, if\ N\ then\ M_1\ else\ M_2) =$
  let $\langle \Xi_0, N'^{\alpha^{bool}}, \mathcal{E}_0 \rangle = \mathcal{W}^\star(p, \Theta, N)$
    and $\langle \Xi_1, M'^{\eta_1}_1, \mathcal{E}_1 \rangle = \mathcal{W}^\star(p, \Theta, M_1)$
    and $\langle \Xi_2, M'^{\eta_2}_2, \mathcal{E}_2 \rangle = \mathcal{W}^\star(p, \Theta, M_2)$
    and $\langle \eta, \mathcal{E} \rangle = \mathcal{J}(p, \eta_1, \eta_2)$
  in $\langle \Xi_0 \uplus \Xi_1 \uplus \Xi_2,$
    $(if\ N'^{\alpha^{bool}}\ then\ M'^{\theta_1}_1\ else\ M'^{\theta_2}_2)^\eta,$
    $\{(\delta\ in\ tail(\eta)) \Rightarrow (\{\alpha \equiv \delta\} \cup \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E})\}\rangle$ end

Fig. 13. Algorithm $\mathcal{W}^\star$ (continue)

$\mathcal{W}^\star(p,\Theta,case(N,M,F)) =$
let $\langle \Xi_0, {N'^{\alpha_0}}^{nat}, \mathcal{E}_0\rangle = \mathcal{W}^\star(p,\Theta,N)$
and $\langle \Xi_1, {M'^{\eta_1}}, \mathcal{E}_1\rangle = \mathcal{W}^\star(p,\Theta,M)$
and $\langle \Xi_2, {F'^{\alpha_1}}^{nat\wedge\cdots\wedge{\alpha_n}^{nat}\to\eta_2}, \mathcal{E}_2\rangle = \mathcal{W}^\star(p,\Theta,F)$
and $\alpha = newa()$
and $\langle \eta, \mathcal{E}\rangle = \mathcal{J}(p,\eta_1,\eta_2)$
in $\langle \Xi_0 \uplus \Xi_1 \uplus \Xi_2,$
$(case({N'^{\alpha_0}}^{nat}, {M'^{\theta_1}}, {F'^{\alpha}}^{nat\to\theta_2})^\theta,$
$\{(\delta\ in\ tail(\eta)) \Rightarrow (\{\alpha_0 \equiv \delta, \alpha \sqsubseteq \alpha_1, \ldots, \alpha \sqsubseteq \alpha_n\} \cup \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E})\}\rangle$ end

$\mathcal{W}^\star(p,\Theta,it(N,M,F)) =$
let $\langle \Xi_0, {N'^{\alpha_0}}^{nat}, \mathcal{E}_0\rangle = \mathcal{W}^\star(0,\Theta,N)$
and $\langle \Xi_1, {M'^{\theta_1}}, \mathcal{E}_1\rangle = \mathcal{W}^\star(0,\Xi,M)$
and $\langle \Xi_3, {F'^{\theta_2\to\theta_3}}, \mathcal{E}_2\rangle = \mathcal{W}^\star(0,\Theta,F)$
and $\theta = fresh(\epsilon(\theta_1))$
in $\langle \Xi_0 \uplus \Xi_1 \uplus \Xi_2,$
$(it({N'^{\alpha_0}}^{nat}, {M'^{\theta_1}}, {F'^{\theta_2\to\theta_3}})^\theta,$
$\{(\delta\ in\ tail(\theta)) \supset (\{\alpha_0 \equiv \delta\} \cup \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2 \cup ucs_{\leq_0}(\theta_1,\theta) \cup$
$ucs_{\leq_0}(\theta_2\to\theta_3, \theta\to\theta))\}\rangle$ end

$\mathcal{W}^\star(p,\Theta,rec(N,M,F)) =$
let $\langle \Xi_0, {N'^{\alpha_0}}^{nat}, \mathcal{E}_0\rangle = \mathcal{W}^\star(0,\Theta,N)$
and $\langle \Xi_1, {M'^{\theta_1}}, \mathcal{E}_1\rangle = \mathcal{W}^\star(0,\Theta,M)$
and $\langle \Xi_3, {F'^{\alpha_1}}^{nat\to\theta_2\to\theta_3}, \mathcal{E}_2\rangle = \mathcal{W}^\star(0,\Theta,F)$
and $\theta = fresh(\epsilon(\theta_1))$
in $\langle \Xi_0 \uplus \Xi_1 \uplus \Xi_2,$
$(rec({N'^{\alpha_0}}^{nat}, {M'^{\theta_1}}, {F'^{\alpha_1}}^{nat\to\theta_2\to\theta_3})^\theta,$
$\{(\delta\ in\ tail(\theta)) \supset (\{\alpha_0 \equiv \delta\} \cup \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2 \cup ucs_{\leq_0}(\theta_1,\theta) \cup$
$ucs_{\leq_0}({\alpha_1}^{nat}\to\theta_2\to\theta_3, {\delta}^{nat}\to\theta\to\theta))\}\rangle$ end

Fig. 14. Algorithm $\mathcal{W}^\star$

and $\mathcal{E}$ is the first set of constraints introduced in Example 2.

The set $faithful(\Theta,\gamma_5)$ is the set $\mathcal{E}'$ in Example 2, so $\mathcal{E} \cup faithful(\Theta,\gamma_5) \rightsquigarrow \mathcal{I}$, where

$$\mathcal{I} = \{\alpha_2,\alpha_2',\alpha_4,\alpha_4',\alpha_6,\alpha_8,\alpha_8',\ \beta_1,\beta_1',\beta_2,\beta_2',\beta_3,\beta_3',\beta_4,\beta_4',\beta_5,\beta_6,\beta_7,\beta_7',\beta_8,\beta_8',\ \gamma_2,\gamma_2',\gamma_3,\gamma_4,\gamma_5\} .$$

Let i be defined by: $i(\alpha) = \delta$ if $\alpha \in \mathcal{I}$ and $i(\alpha) = \omega$ otherwise. Then $i(\Theta);\emptyset \vdash^{(0)} i({M'}^\theta)$ is the faithful $\vdash^{(0)}$-typing that shows all the dead code that can be detected by using the a-type assignment system $\vdash_L$.
Note that $i(\Theta) \vdash_L i({M'}^\theta)$ is the faithful a-typing used in Example 1. □

Remark. The algorithm $\mathcal{W}$ is presented it this form to make it as close to the $\vdash^{(0)}$-type assignment system as possible. Indeed it generates some constraints that can be avoided in a real implementation.
Moreover, an efficient implementation of the algorithm should avoid the use of $\bullet$-sequences, recording just the annotation that contain the relevant information w.r.t. the dead code elimination. In fact, as it is easy to see, for every a-2-pattern η associated to a subterm, it suffices to keep just the annotation variables in $tail(\eta)$. So it is possible to record all the relevant annotations by decorating the terms with sets of annotation variables. For instance, the decorated term of Example 3 could be replaced

by the following:

$$
\begin{array}{l}
((\lambda f^{\{\alpha_4,\beta_4\}}. \\
\quad (+^{\{\gamma_5\}}\langle(((f^{\{\alpha'_4\}}(\lambda x^{\{\alpha_5\}}.3^{\{\alpha_6\}})^{\{\alpha_6\}})^{\{\alpha'_4\}}u_1^{\{\gamma'_1\}})^{\{\alpha'_4\}}, \\
\qquad\quad ((f^{\{\beta'_4\}}(\lambda y^{\{\beta_5\}}.y^{\{\beta_6\}})^{\{\beta_6\}})^{\{\beta'_4\}}u_2^{\{\gamma'_2\}})^{\{\beta'_4\}}\rangle^{\{\alpha'_4,\beta'_4\}})^{\{\gamma_5\}})^{\{\gamma_5\}} \\
\qquad\qquad\quad ((\lambda z^{\{\alpha_8,\beta_8\}}.z^{\{\alpha'_8,\beta'_8\}})^{\{\alpha'_8,\beta'_8\}}))^{\{\gamma_5\}}\ .
\end{array}
$$

□

Conclusions and Future Work

In this paper we have presented an extension of the type assignment system for detecting dead code introduced in [10]. The main achievement over that system is the extension of the language of annotated types with rank 2 intersection. We have also presented an inference algorithm which is correct and complete, in the sense that it finds all the dead code that can be detected by using the annotated type assignment system.

The idea of using intersection types for dead code detection seems very natural. In fact they allow to handle some problem in the detection and elimination of dead code in applications. Take for instance the term $(\lambda f.M)\,N$. If we look at the different occurrences of the bound variable f in M (let us denote them by f_i), then it may happen that each f_i has a different annotated type.
Note that in the original framework of [3] this raise problems since, after the optimization process, the different occurrences f_i have different types. This problem can be partially handled by allowing subtyping, as done in [4] (see also [10]). But subtyping is contravariant in the left part of the arrow operator, whereas, to *specialize* a term (see [8]), covariance is needed.
As showed in the present paper, by using rank 2 intersection it is possible to deal with covariance.

The idea of specializing terms seems quite interesting for future works. Consider the following application:

$$
\begin{array}{l}
(\lambda f^{\rho}.\ +\langle +(f\ M\ N\ 0), \pi_1(fP\ Q\ 0)\rangle) \\
\qquad (\mathsf{fix}\ g^{\rho}.\lambda x^{nat}.\lambda y^{nat}.\lambda z^{nat}.\mathsf{if}\ \ > \langle x,y\rangle\ \mathsf{then}\ \ \langle x,z\rangle\ \ \mathsf{else}\ g\ (-\langle x,y\rangle)\ y\ (+\langle 1,z\rangle)\ \ ,
\end{array}
$$

where $\rho = nat \to nat \to nat \to (nat \times nat)$ and M, N, P, Q are terms of type nat. The lambda abstracted variable f is bounded to a function which, given 3 natural numbers x, y, and z, returns the pair formed by the remainder plus z and the quotient of the Euclidean division of x by y (thus when z is 0, it is just the standard Euclidean division). In the first occurrence of f in the body of the lambda abstraction, both the components of the pair computed are used, but in the second occurrence, the remainder is useless, and since z is only used to compute the remainder, it is dead code (in this occurrence). Indeed, it would be interesting to have two different version of the Euclidean division, the first one like the original version, and the second one for the cases when only the remainder is purchased. In this way an optimized version of the term above would look like:

$$
\begin{array}{l}
(\lambda f.\ +\langle +(f\ M\ N\ 0), \pi_1(f\ P\ Q)\rangle) \\
\qquad ((\mathsf{fix}\ g^{\rho}.\lambda x^{nat}.\lambda y^{nat}.\lambda z^{nat}.\mathsf{if}\ \ > \langle x,y\rangle\ \mathsf{then}\ \ \langle x,z\rangle\ \ \mathsf{else}\ g(-\langle x,y\rangle)y(+\langle 1,z\rangle) \\
\qquad\ \bullet \\
\qquad\ (\mathsf{fix}\ g^{\sigma}.\lambda x^{nat}.\lambda y^{nat}.\mathsf{if}\ \ < \langle x,y\rangle\ \mathsf{then}\ \ x\ \ \mathsf{else}\ g(-\langle x,y\rangle)\ y))) \quad ,
\end{array}
$$

where $\sigma = nat \to nat \to nat$.
If we allow these kind of optimization, we have to handle overloaded functions. Indeed, in this case f is bound to two different branches, and when it is used in the body of the lambda abstraction, we have to choose the right branch. This can be done by looking at the actual type of f in the body of the lambda abstraction. The $\lambda\&$ calculus of Castagna, see [9], seems a good candidate to explore further this idea.

References

1. H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48:931–940, 1983.
2. P. N. Benton. *Strictness Analysis of Lazy Functional Programs.* PhD thesis, University of Cambridge, Pembroke College, 1992.
3. S. Berardi. Pruning Simply Typed Lambda Terms. *Journal of Logic and Computation*, 6(5):663–681, 1996.
4. S. Berardi and L. Boerio. Using Subtyping in Program Optimization. In *Typed Lambda Calculus and Applications*, 1995.
5. L. Boerio. *Optimizing Programs Extracted from Proofs.* PhD thesis, Universitá di Torino, 1995.
6. C. Pauline-Möhring. Extracting F_ω's Programs from Proofs in the Calculus of Constructions. In *POPL'89*. ACM, 1989.
7. C. Pauline-Möhring. *Extraction des Programmes dans le Calcul des Constructions.* PhD thesis, Université Paris VII, 1989.
8. G. Castagna. Covariance and contravariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, 1995.
9. Giuseppe Castagna. *Object-Oriented Programming: A Unified Foundation.* Progress in Theoretical Computer Science. Birkäuser, Boston, 1996. To appear.
10. M. Coppo, F. Damiani, and P. Giannini. Refinement Types for Program Analysis. In *SAS'96*, LNCS 1145, pages 143–158. Springer, 1996.
11. M. Coppo and M. Dezani-Ciancaglini. An extension of basic functional theory for lambda-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.
12. M. Coppo and A. Ferrari. Type inference, abstract interpretation and strictness analysis. *Theoretical Computer Science*, 121:113–145, 1993.
13. D. Dussart and F. Henglein and C. Mossin. Polymorphic Recursion and Subtype Qualifications: Polymorphic Binding-Time Analysis in Polynomial Time. In *SAS'95*, LNCS 983, pages 118–135. Springer, 1995.
14. F. Damiani and P. Giannini. An Inference Algorithm for Strictness. In *TLCA'97*, To appear in LNCS. Springer, 1997.
15. C. Hankin and D. Le Métayer. Deriving algorithms for type inference systems: Applications to strictness analysis. In *POPL'94*, pages 202–212. ACM, 1994.
16. T. P. Jensen. *Abstract Interpretation in Logical Form.* PhD thesis, University of London, Imperial College, 1992.
17. G. Kahn. Natural semantics. In K. Fuchi and M. Nivat, editors, *Programming Of Future Generation Computer.* Elsevier Sciences B.V. (North-Holland), 1988.
18. T. M. Kuo and P. Mishra. Strictness analysis: a new perspective based on type inference. In *Functional Programming Languages and Computer Architecture*, pages 260–272. ACM, 1989.
19. J. Palsberg and P. O'Keefe. A Type System Equivalent to Flow Analysis. In *POPL'95*, pages 367–378. ACM, 1995.
20. A. M. Pitts. Operationally-Based Theories of Program Equivalence. Summer School on *Semantics and Logics of Computation*, Cambridge UK, 25-29 Sep 1995.
21. F. Prost. Marking techniques for extraction. Technical report, Ecole Normale Supérieure de Lyon, Lyon, December 1995.
22. K. L. Solberg. *Annotated Type Systems for Program Analysis.* PhD thesis, Aarhus University, Denmark, 1995. Revised version.

23. Y. Takayama. Extraction of Redundancy-free Programs from Constructive Natural Deduction Proofs. *Journal of Symbolic Computation*, 12:29–69, 1991.
24. S. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems*. PhD thesis, Katholieke Universiteit Nijmegen, 1993.
25. D. A. Wright. *Reduction Types and Intensionality in the Lambda-Calculus*. PhD thesis, University of Tasmania, 1992.