Forwards and backwards analysis for functional programs

Andrei Sabelfeld
Chalmers University of Technology
Göteborg, Sweden
andrei@cs.chalmers.se

Viktor Sabelfeld
Karlsruhe University
Karlsruhe, Germany
sabelfel@ira.uka.de

Abstract

This paper presents an approach to static analysis of functional programs that lies between abstract interpretation and flow analysis approaches. Two general frameworks of semantic property analysis for recursion schemes are described. We present a marking technique for solving a rather powerful subset of these problems. Illustrative examples that are captured by this technique are *strictness analysis* and *needed/unneeded analysis*. However, some static property analyses of recursive programs described by either flow analysis or abstract interpretation approaches cannot be formulated as either forwards or backwards analysis problems. As an instance, we consider the *formal parameter dependence analysis*.

1 Introduction

To check applicability conditions of equivalent program transformations, to detect semantic errors in programs, and to specialise programs efficiently one should learn how to determine semantic properties of programs. Many authors have explored program analysis problems on program models. Most results have been obtained in flow analysis [KU77] and abstract interpretation [CC77, CC92] approaches.

We introduce a new method of semantic analysis for first order call-by-name functional programs (*recursion schemes* [GL71]). We describe two simple classes of semantic property analysis problems, *forwards* and *backwards property analysis problems* for recursion schemes, and use a marking technique for their solution. Our approach is close to Patrick and Radhia Cousot's abstract interpretation approach [CC77, CC92]. We consider the fix-point semantics of recursive programs. Namely, we associate a term sequence called *approximation sequence* to a recursion scheme. Given the properties of terms from the approximation sequence we define the program property as the limit of the constructed sequence of properties. Forwards analysis is applied if we are interested in properties that flow forwards from subexpressions to the big expression that contains them. On the contrary, backwards analysis is useful when properties flow from a large expression to its subexpressions.

Forwards and backwards property analysis problems prove to be undecidable in the general case. However, we give sufficient conditions for solving these problems and present marking algorithms that compute program properties under the conditions required. In Sections 4.2 and 5.2 we give examples of forwards (*strictness analysis*) and backwards (*needed/unneeded analysis*) property analysis problems.

Some static property analyses of recursive programs that are well described either in the flow analysis or abstract interpretation approaches cannot be formulated as either forwards or backwards analysis problems, unless computation traces are propagated. As an instance, in Section 6 we describe the *formal parameter dependence analysis* of recursion schemes as a flow analysis problem.

Section 7 discusses related work and concludes.

2 Recursion scheme definition

Let $\mathcal{X} = \{x, y, z, \ldots\}$ be a set of *variables*, $\mathcal{F}_b = \{\omega, f, g, h, \ldots\}$ be a set of *basic symbols*, $\mathcal{F}_d = \{F, F_1, F_2, \ldots\}$ be a set of *defined symbols*, $\mathcal{F} = \mathcal{F}_b \bigcup \mathcal{F}_d$, and r denote the *rank* function. Each symbol $\phi \in \mathcal{F}$ of rank $r(\phi) = n$ is said to have arity n. The symbol ω stands for "undefined", the least informative term; $r(\omega) = 0$. For a set X of variables, $X \subset \mathcal{X}$, let $\mathcal{T}(X)$ be the set of all *terms* under variables from X formed using basic and defined symbols from $\mathcal{F}$.

A *recursion scheme* is a pair $S = \langle e; DEF \rangle$, where e is a term called *scheme entry* and DEF is a finite set of symbol definitions in $\mathcal{F}_d$. A *definition* of a symbol $F \in \mathcal{F}_d$ has the form $F(x_1, \ldots, x_n) \Leftarrow t$, where $n = r(F), x_1, \ldots, x_n \in \mathcal{X}$ are different *formal parameters* of F and $t \in \mathcal{T}(x_1, \ldots, x_n)$ is the *body* of the symbol F. We assume that each formal parameter is distinct from all the variables appearing in the entry.

A term $F(t_1, \ldots, t_n)$, where $n = r(F)$ and $F \in \mathcal{F}_d$ is said to be a *call* to the symbol F. Its subterms $t_1, \ldots, t_n$ are *actual parameters* of this call. We require each symbol called in a scheme to be defined in it. Here is an example of a scheme:

$$\left\langle F_1(h,h) \; ; \; \begin{array}{ll} F_1(x,y) & \Leftarrow \mathbf{if}(px, F_2(fx, fy), x) \\ F_2(x,y) & \Leftarrow \mathbf{if}(py, F_1(fy, fx), y) \\ F_3(x) & \Leftarrow \mathbf{if}(px, F_3(fx), x) \end{array} \right\rangle$$

Recall the standard notions of *occurrence* and *substitution* from term rewriting theory [Klo91]. Any *occurrence* of a subterm in a term can be uniquely identified by the path from the root of the tree representation of the term to the subterm. Subterms of the entry and definition bodies in a scheme are identified as follows. The *address* of the the scheme entry (or a tree root if the tree is unique in the context under consideration) is the empty word Λ. The *address* of the body of a symbol F is the word $[F]$. If an occurrence of a term $f(t_1, \ldots, t_n)$ (or $F(t_1, \ldots, t_n)$) has the address a, then its subterm occurrences $t_1, \ldots, t_n$ have the addresses $a.1, \ldots, a.n$ respectively. For example, the first occurrence of the term fx in the scheme above has the address $[F_1].2.1$, and the first occurrence of the constant h has the address $\Lambda.1$, or just 1. Two addresses are *independent* iff neither is a prefix of the other.

Suppose a is an address and S is a scheme or a term. Let $S \downarrow a$ stand for the term at a in S, and $S[a \leftarrow t]$ denote the scheme (or term) obtained from S by replacing the term at a by t. If N is a set of mutually independent addresses of subterm occurrences of a term t, then $t[N \leftarrow \tau]$ denotes the term obtained from t by replacing all subterm occurrences at addresses from N by the term τ.

A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X})$ that satisfies the condition $\sigma x \neq x$ for only a finite number of variables x from $\mathcal{X}$. The substitution

σ that maps a variable x_i into a term t_i for $i = 1, \ldots, n$ is denoted by $[t_1/x_1, \ldots, t_n/x_n]$. The notion of substitution can be extended in a natural way to arbitrary terms: $\sigma\phi(t_1, \ldots, t_n) = \phi(\sigma t_1, \ldots, \sigma t_n)$ for $\phi \in \mathcal{F}$ and $n = r(\phi)$.

Every scheme S defines a mapping $\pi : \mathcal{T}(\mathcal{X}) \to \mathcal{T}(\mathcal{X})$ which corresponds to the "parallel outermost computation rule". It performs one step unfolding of all outermost calls in a term.

$$\pi t = \begin{cases} x, & \text{if } t = x, \text{ where } x \in \mathcal{X}, \\ f(\pi t_1, \ldots, \pi t_n), & \text{if } t = f(t_1, \ldots, t_n), \text{ where } n = r(f) \text{ and } f \in \mathcal{F}_b, \\ \sigma\tau, & \text{if } t = F(t_1, \ldots, t_n), \text{where } F \in \mathcal{F}_d, \\ & \sigma = [t_1/x_1, \ldots, t_n/x_n], \\ & \text{and } F(x_1, \ldots, x_n) \Leftarrow \tau \text{ is the definition of } F \text{ in } S. \end{cases}$$

3 Interpretation and equivalence

In this Section we define the algebraic semantics of recursive programs. A more detailed presentation can be found, for instance, in [Gue81]. A *domain* is a triple $\langle D, \leq, \perp \rangle$, where D is a set, $\perp \in D$ (the "bottom") and $\leq$ is a partial order on D that satisfies the following two conditions:

- $\forall\, d \in D \;\; \perp \leq d$.
- *Completeness condition:* for each chain C in D the *least upper bound* $lub(C)$ belongs to D. ($d \leq lub(C)$ for all d in C, and $lub(C) \leq u$ for all u in D such that $d \leq u$ for all d in C.)

The *universal term domain* $\mathcal{U} = \langle T_U, \sqsubseteq, \omega \rangle$ is constructed in the following way. Consider the set T of all terms under the signature $\langle \mathcal{X}, \mathcal{F}_b \rangle$ with the partial order $\sqsubseteq$ introduced by setting $t_1 \sqsubseteq t_2$ iff there exists a set N of mutually independent addresses in t_2, such that $t_1 = t_2[N \leftarrow \omega]$. Intuitively it means that $t_1 \sqsubseteq t_2$ iff t_1 can be obtained from t_2 by replacing 0 or more subterms by ω. Let T_U be the completion of the set T by least upper bounds of all infinite increasing chains of elements from T. The elements from T_U can be treated as (infinite) terms (trees). One can prove that $\mathcal{U}$ is a domain with the partial order $\sqsubseteq$: $t_1 \sqsubseteq t_2$ iff there is a (possibly infinite) set N of mutually independent addresses in t_2 such that $t_1 = t_2[N \leftarrow \omega]$, and bottom ω.

Suppose D and D' are domains. A function $\varphi : D \to D'$ is *monotone,* iff $\forall\ d, d' \in D\ d \leq_D d' \Rightarrow \varphi d \leq_{D'} \varphi d'$. A monotone function is *continuous* if it preserves the least upper bounds of non-empty linearly ordered subsets of D, i.e. $\varphi(lub(C)) = lub(\varphi(C))$.

An *interpretation* I fixes a domain D and assigns

- a member $I(x) \in D$ to each variable $x \in \mathcal{X}$,
- a continuous function $I(f) : D^{r(f)} \to D$ to each symbol $f \in \mathcal{F}_b$, the constant ω always gets $I(\omega) = \perp_D$.

Using the structural induction we can extend the notion of an interpretation I to arbitrary finite terms by considering all defined symbol applications in the terms to be "unknown":

$$I(t) = \begin{cases} I(x), & \text{if } t = x, \text{ where } x \in \mathcal{X}; \\ I(f)(I(t_1), \ldots, I(t_n)), & \text{if } t = f(t_1, \ldots, t_n), \text{ where } f \in \mathcal{F}_b; \\ \perp & \text{otherwise.} \end{cases}$$

An important instance of an interpretation is the *universal* (or *free*, or *initial*, or *Herbrand*) *interpretation* J on the domain $\mathcal{U}$. It assigns symbolic values to terms. Let $J(x) = x$ for variables $x \in \mathcal{X}$, and for $f \in \mathcal{F}_b, n = r(f)$, and $t_1, \ldots, t_n \in T_U$ set $J(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$.

Define the *approximation sequence* of t by $App_n(S, t) = J(\pi^n t)$, where $n \geq 0, \pi^0 \tau = \tau$, and $\pi^n = \pi^{n-1}\pi$ for $n > 0$. Let us now formally define the full unwinding of a term t with respect to a program S. The *determinant* $det(t)$ *of a term* t is the least upper bound of the approximation sequence $\{App_0(S, t), App_1(S, t), \ldots\}$, and the *determinant* $det(S)$ is the determinant of the entry of the scheme S. For example, the first elements of the approximation sequence of the entry of $\langle F(u); F(x) \Leftarrow f(x, F(gx))\rangle$ are

$$\omega \sqsubseteq f(u, \omega) \sqsubseteq f(u, f(gu, \omega)) \sqsubseteq f(u, f(gu, f(ggu, \omega))) \sqsubseteq \ldots, \text{ etc.}$$

The least upper bound of such a sequence is an element of T_U. This element is the algebraic semantics of the example program.

Two schemes S_1 and S_2 with the entries t_1 and t_2 are *equivalent* ($S_1 \sim S_2$ for short) iff $I(t_1) = I(t_2)$ for all interpretations I.

4 Forwards semantic analysis framework

In this section we will formulate the forwards semantic analysis framework, exemplify it (by considering strictness analysis as a forwards analysis problem), show its undecidability in the general case, present a solution under certain conditions, and, finally, apply the solution to the example.

4.1 Flowing forwards

Let $\mathcal{L} = \langle L, \sqcup, \perp \rangle$ be an upper semi-lattice, where L is a set with a partial order $\sqsubseteq$ on it, $\perp$ is the *bottom* element, and $\sqcup$ is a binary *union* operation that satisfies $\forall x, y, z \in L\ x \sqcup y = y \sqcup x, (x \sqcup y) \sqcup z = x \sqcup (y \sqcup z), x \sqcup x = x, x \sqcup \perp = x$. Define $x \sqsubset y$ iff $x \sqsubseteq y$ and $x \neq y$. A function $f : L \to L$ is *distributive* iff $f(x \sqcup y) = f(x) \sqcup f(y)$. We will consider only semi-lattices satisfying the ascending chain condition. (No infinitely ascending chains are allowed.)

We have an instance of a *forwards semantic analysis problem* for a recursion scheme S by giving:

- A property semi-lattice $\mathcal{L}$ that satisfies the chain condition.
- A *deductive semantic function* $Sem^{\uparrow}$ that assigns a monotone property transformer $Sem^{\uparrow}(f) : L^n \to L, n = r(f)$, to each symbol $f \in \mathcal{F}_b$. Such a transformer defines the property of a term $f(t_1, \ldots, t_n)$ using the properties $s_1, \ldots, s_n$ of its subterms $t_1, \ldots, t_n \in L$ as $Sem^{\uparrow}(f)(s_1, \ldots, s_n)$.

For an address a in a scheme S define "variables in the scope" $Var(a)$ as follows. If a is an address of a subterm of the body of some F, let $Var(a)$ be F's formal parameter set. If a is an address of a subterm of the scheme entry, then define $Var(a)$ to be the set of all variables occurring in the entry. Let $\overline{s}$ denote the tuple $s_1, \ldots, s_n$ for some n clear from the context. By Pr_i we denote the *projection function* $Pr_i \in (L^n \to L), Pr_i\overline{s} = s_i$. The deductive semantic function can be extended to arbitrary terms without calls. For such a term t set

$$Sem^*(t)\overline{s} = \begin{cases} s_i, & \text{if } t = x_i, \\ Sem^{\uparrow}(f)(Sem^*(t_1)\overline{s}, \ldots, Sem^*(t_n)\overline{s}), & \text{if } f \in \mathcal{F}_b \ \&\ t = f(t_1, \ldots, t_n), \end{cases}$$

where $|\overline{s}|$ is the number of free variables of t. For an address a in a scheme S define $d(a, n) = Sem^*(App_n(S, S \downarrow a))$ as an element of the semi-lattice $L^{|Var(a)|} \to L$.

The forwards semantic analysis problem for the scheme S consists of finding the property

$$D(a) = \bigsqcup_{n=0}^{\infty} d(a, n)$$

for all addresses a of the scheme S.

4.2 Example: strictness analysis

This analysis is similar to Mycroft's strictness analysis [Myc80, Myc81]. Suppose that a subset $Strict(f)$ of the set $2^{\{1,\ldots,r(f)\}}$ is associated to each basic symbol f, call $Strict(f)$ the set of *strict parameter collections* of the symbol f. An interpretation I is *strict*, if for any $d_1, \ldots, d_n \in D_I$ the condition

$$(\forall \Delta \in Strict(f)\ \exists i \in \Delta\ d_i = \perp) \Rightarrow I(f)(d_1, \ldots, d_n) = \perp$$

holds for each basic symbol $f \in \mathcal{F}_b$. For example, the natural way to bound the interpretations of the ternary symbol **if** is to settle $Strict(\mathbf{if}) = \{\{1, 2\}, \{1, 3\}\}$. This means that we restrict all possible interpretations of the symbol **if** to functions *cond* with the condition

$$\forall\ d, d' \in D\ (cond(\perp, d, d') = \perp)\ \&\ (cond(d, \perp, \perp) = \perp)$$

imposed. Another example is the "vote" function symbol Γ^3_2 with the set $Strict(\Gamma^3_2) = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$ of strict parameter collection, so that

$$\forall\ d \in D\ I(\Gamma^3_2)(\perp, \perp, d) = \perp\ \&\ I(\Gamma^3_2)(\perp, d, \perp) = \perp\ \&\ I(\Gamma^3_2)(d, \perp, \perp) = \perp$$

for all strict interpretations I. An address a of a scheme S is called *hopeless*, iff $I(S \downarrow a) = \perp$ for all strict interpretations I. For example, if $Strict(f) = Strict(h) = \{\{1\}\}$ and $Strict(g) = \{\{2\}\}$, then in the scheme

$$\langle\ F(u, \omega); F(x, y) \Leftarrow g(hy, F(fx, hy))\ \rangle$$

the address Λ is hopeless.

Now we formulate a forwards analysis problem in order to find all hopeless addresses of a scheme. According to the four dimensional strictness analysis classification [DW90] this algorithm is first-order, flat, high fidelity, and forwards. Take the property lattice $L_{str} = \{\perp, \top\}$, $\perp \sqsubset \top$. $\top$ stands for "might be defined", $\perp$ means "certainly undefined". For such a simple lattice $x \sqcap y$

is always either x or y, and similarly $x \sqcup y$ is either x or y. Let the deductive semantic function $Sem^{\uparrow}_{hope}$ assign the monotone transformer

$$Sem^{\uparrow}_{hope}(f)(s_1, \ldots, s_n) = \bigsqcup_{\Delta \in Strict(f)} \bigsqcap_{i \in \Delta} s_i$$

to each symbol $f \in \mathcal{F}_b$. Then an address a of a scheme S is hopeless if $D_{hope}(a)(\top, \ldots, \top) = \bot$. The general result of the strictness analysis for a term t at some address a is described by the function $Sem^*(t)$ that has type $L_{str}^{|Var(a)|} \to L_{str}$. Let $P \cup Q$ be a partition of $\{1, \ldots, |Var(a)|\}$. If $Sem^*(t)$ applied to a combination $\overline{s}$, where $s_i = \bot$ for $i \in P$ and $s_i = \top$ for $i \in Q$, results in $\bot$, it means that in any strict interpretation if the arguments to t indexed by P are undefined then t applied to those arguments along with arbitrary arguments indexed by Q will be undefined as well.

4.3 Undecidability of the general case

Theorem 1. *The forwards semantic analysis problem described is undecidable in the general case.*

Proof. We modify the construction used in [KU77] to reduce Post's correspondence problem [Pos46] for a system (X, Y):

$$X = (\alpha_1, \ldots, \alpha_n), Y = (\beta_1, \ldots, \beta_n),$$

where $i = 1, \ldots, n$; α_i and β_i are words in the alphabet $\{0, 1\}$, to the forwards semantic analysis problem for

$$\left\langle \begin{array}{l} f_1(F(g_1 d), \ldots, F(g_n d)); \\ F(x) \Leftarrow f_2(hx, F(g_1 x), \ldots, F(g_n x)) \end{array} \right\rangle.$$

Let us consider the property semi-lattice

$$L = \{\bot, \top\} \cup \{0,1\}^* \times \{0,1\}^*, x \sqsubseteq y \text{ iff } x = y \text{ or } x = \bot \text{ or } y = \top.$$

This union $x \sqcup y$ is defined to be y, if $x \sqsubseteq y$, or $\top$ otherwise. The lengths of all increasing chains are, apparently, bounded by 3. The semantic function is defined as follows: for $i = 1, \ldots, n$ set $Sem^{\uparrow}(g_i)(\bot) = \bot, Sem^{\uparrow}(g_i)(\top) = \top, Sem^{\uparrow}(g_i)((\alpha, \beta)) = (\alpha\alpha_i, \beta\beta_i)$,

$$Sem^{\uparrow}(f_1)(s_1, \ldots, s_n) = s_1 \sqcup \ldots \sqcup s_n,$$

$$Sem^{\uparrow}(f_2)(s_0, s_1, \ldots, s_n) = s_0 \sqcup s_1 \sqcup \ldots \sqcup s_n,$$
$$Sem^{\uparrow}(h)(\perp) = \perp, Sem^{\uparrow}(h)(\top) = \top,$$
$$Sem^{\uparrow}(h)((\alpha, \beta)) = \begin{cases} \top, & \text{if } \alpha = \beta, \\ \perp, & \text{if } \alpha \neq \beta, \end{cases}$$

$Sem^{\uparrow}(d) = (\Lambda, \Lambda)$, where Λ is the empty word in the alphabet $\{0, 1\}$. In order to prove the monotony of the functions $Sem^{\uparrow}(h)$ and $Sem^{\uparrow}(g_i)$ $(i = 1, \ldots, n)$ we use the fact that $x \neq \perp$, and $x \sqsubset y$ for $x, y \in L$ implies $y = \top$. Thus if $x \neq \perp$,

$$x \sqsubset y \Rightarrow Sem^{\uparrow}(h)(y) = Sem^{\uparrow}(h)(\top) = \top \sqsupseteq Sem^{\uparrow}(h)(x), \text{ and}$$

$$x \sqsubset y \Rightarrow Sem^{\uparrow}(g_i)(y) = Sem^{\uparrow}(g_i)(\top) = \top \sqsupseteq Sem^{\uparrow}(g_i)(x).$$

Otherwise,

$$\perp \sqsubset y \Rightarrow Sem^{\uparrow}(h)(\perp) = Sem^{\uparrow}(g_i)(\perp) = \perp \sqsubseteq z \text{ for any } z \in L.$$

The monotony of the functions $Sem^{\uparrow}(d), Sem^{\uparrow}(f_1)$ and $Sem^{\uparrow}(f_2)$ is obvious, they are even distributive.

To complete the proof observe that for the formulated forwards analysis problem the condition $D(\Lambda) = \top$ holds iff the system (X, Y) has at least one solution. If we had an algorithm that solves the forwards analysis problem we would be able to solve Post's correspondence problem. □

4.4 Solution

Theorem 2. *If the semi-lattice L is finite and all property transformers are distributive in each argument, then the forwards analysis problem can be solved using the following marking algorithm.*

A *d-marking* of a scheme S is a mapping μ that assigns a mapping (*d-mark*) $\mu a \in (L^n \to L)$, $n = |Var(a)|$, to each address a in S. The initial d-marking μ_0 assigns the d-mark Pr_i to all addresses of the occurrences of the variable x_i and the d-mark $\perp\!\!\!\perp$ ($\perp\!\!\!\perp(x_1, \ldots, x_n) = \perp$) to all the other addresses in S. The initial d-marking and all d-markings which can be obtained from it by serial application of *d-marking rules* are called *reachable*. An application of a d-marking rule to an address a consists of the following:

Let $S \downarrow a = \phi(t_1, \ldots, t_n)$, where $\phi \in \mathcal{F}, n = r(\phi)$. We replace the d-mark μa of the address a by the d-mark $\mu a \sqcup Q$, where the mapping Q is defined by

$$Q = \begin{cases} Sem^{\uparrow}(\phi) \circ (\mu a.1, \ldots, \mu a.n), & \text{if } \phi \in \mathcal{F}_b, \\ \mu[\phi] \circ (\mu a.1, \ldots, \mu a.n), & \text{if } \phi \in \mathcal{F}_d. \end{cases}$$

Here and later we omit the parenthesis in marks $\mu a.i$ to mean $\mu(a.i)$. A *stationary d-marking* is the reachable d-marking that cannot be changed by any application of the d-marking rule. This stationary marking gives the precise solution of the forwards analysis problem. The premises of the theorem, the chain condition, and the monotony of all property transformers guarantee the existence and uniqueness of the stationary d-marking. A formal proof is sketched in Appendix A.2.

4.5 Application to the example

Note that *cond* is not distributive in all arguments, but is distributive in each argument. Moreover, all monotone functions on the two element domain L_{str} are distributive in each argument. This makes Theorem 2 applicable to the strictness analysis. According to the described above main algorithm for the solution of the forwards analysis problem obtain the following procedure.

The initial marking: $\mu_0 a = Pr_i$ for addresses a of the occurrences of a variable x_i, and $\mu_0 a = \perp\!\!\!\perp$ for all the other addresses a in S.

The marking rule: if $S \downarrow a = \phi(t_1, \ldots, t_n)$, where $\phi \in \mathcal{F}, n = r(\phi)$, then the mark μa of an address a is replaced by the mark $\mu' a$:

$$\mu' a = \mu a \sqcup \begin{cases} \bigsqcup_{\Delta \in Strict(\phi)} \bigsqcap_{i \in \Delta} \mu a.i, & \text{if } \phi \in \mathcal{F}_b, \\ \mu[\phi] \circ (\mu a.1, \ldots, \mu a.n), & \text{if } \phi \in \mathcal{F}_d. \end{cases}$$

5 Backwards semantic analysis framework

5.1 Flowing backwards

The *image set* $Image(a)$ of an address a of a scheme S is defined as the minimal subset of addresses of subterms occurring in $det(S)$ that satisfies the following conditions:

- $\Lambda \in Image(\Lambda)$.
- If $b \in Image(a), S \downarrow a = f(t_1, \ldots, t_n)$ and $f \in \mathcal{F}_b$, then $b.i \in Image(a.i)$ for all $i = 1, \ldots, n$.

- If $b \in Image(a)$ and $S \downarrow a = F(t_1, \ldots, t_n)$ for a symbol F defined in S, then $b \in Image([F])$.

- If $Image(a) \neq \emptyset, S \downarrow a = F(t_1, \ldots, t_n)$ for a symbol F defined in S, c is an address of an occurrence of the i-th formal parameter in the body of F, and $b \in Image(c)$, then $b \in Image(a.i)$.

Such a subset exists by the Knaster-Tarski theorem, since the corresponding defining operator is monotone.

Informally, the image of a program term's address is the set of the addresses of those terms in the determinant that "stem from" the use of the term. The image set can be empty. For instance, for

$$\langle F(u,v); F(x,y) \Leftarrow f(x, F(gx, gy)), F'(x) \Leftarrow f(x) \rangle$$

we get $Image([F']) = \emptyset$ as none of the terms $\pi^n F(u,v)$ invokes F'. Besides, $Image([F].2.2) = \emptyset$ since F's second argument is "unneeded". We will investigate how to find such addresses later on, in Section 5.2.

An instance of a *backwards semantic analysis problem* for a recursion scheme S is given by:

- A property semi-lattice $\mathcal{L}$ that satisfies the chain condition.

- An initial property $h_0 \in L$.

- A *semantic heritage function* $Sem^{\downarrow}$ that assigns a monotone property transformer $Sem^{\downarrow}(f,i) : L \to L$ to each symbol $f \in \mathcal{F}_b$ and $i \in \{1, \ldots, r(f)\}$. Such a transformer defines the property of the i-th subterm of a term $f(\ldots)$ with a given property $s \in L$ as $Sem^{\downarrow}(f,i)(s)$.

The property $h^{\downarrow}(a)$ of an address a in $det(S)$ is defined by

$$h^{\downarrow}(a) = \begin{cases} h_0, & \text{if } a = \Lambda, \\ Sem^{\downarrow}(f,i)(h^{\downarrow}(b)), & \text{if } f \in \mathcal{F}_b \;\&\; \exists i \; a = b.i \;\&\; det(S) \downarrow b = f(t_1, \ldots, t_n). \end{cases}$$

The backwards semantic analysis problem for a scheme S consists of finding the property

$$H^{\downarrow}(a) = \bigsqcup_{b \in Image(a)} h^{\downarrow}(b)$$

for all addresses a in S.

5.2 Example: needed/unneeded analysis

We will formulate the needed/unneeded analysis, which finds expressions that are not needed to be evaluated, as a backwards analysis problem. As Hughes mentions in [Hug87], programmers do not usually write functions with unneeded parameters, whereas partial evaluators might well introduce them in residual programs. Let us call an address a in a scheme S *unneeded* iff $Image(a) = \emptyset$. For example, in the scheme

$$\langle F(u,v); F(x,y) \Leftarrow \mathbf{if}(px, x, F(fx, fy))\rangle$$

the addresses 2, $[F]$.3.2 and $[F]$3.2.1 are unneeded. For an unneeded address a, $S \sim S[a \leftarrow t]$ holds for arbitrary terms $t \in \mathcal{T}(Var(a))$. For example,

$$\left\langle \begin{array}{l} F(u,v); \\ F(x,y) \Leftarrow \mathbf{if}(px, x, F(fx, fy)) \end{array} \right\rangle \sim \left\langle \begin{array}{l} F(u,v); \\ F(x,y) \Leftarrow \mathbf{if}(px, x, F(fx, b)) \end{array} \right\rangle.$$

We can formulate a backwards analysis problem in order to find all unneeded addresses of a scheme S. Take the property semi-lattice $L = \{\mathbf{O}, \mathbf{1}\}$; $\mathbf{O}$ stands for "unneeded", and $\mathbf{1}$ stands for "needed". Let

$$\mathbf{O} \sqsubseteq \mathbf{O}, \mathbf{1} \sqsubseteq \mathbf{1}, \mathbf{O} \sqsubseteq \mathbf{1}, p \sqcup p' = \mathbf{O} \Leftrightarrow p = \mathbf{O} \ \&\ p' = \mathbf{O}.$$

The initial property is $\mathbf{1}$. The semantic heritage function $Sem^{\downarrow}$ assigns the distributive property transformer $Sem^{\downarrow}(f,i) = id_L$ to each basic symbol $f \in \mathcal{F}_b$ and $i \in \{1, \ldots, r(f)\}$. Then

$$H^{\downarrow}_{need}(a) = \bigsqcup_{b \in Image(a)} h^{\downarrow}(b) = \mathbf{O} \text{ iff } Image(a) = \emptyset.$$

5.3 Undecidability of the general case

Theorem 3. *The backwards semantic analysis problem described is undecidable in the general case.*

Proof. In a similar, to the forwards problem, way we can prove that the backwards analysis problem is undecidable in the general case. In fact, the leaves of the determinant-tree for the program

$$\left\langle \begin{array}{l} f_1(F(g_1 d), \ldots, F(g_n d)); \\ F(x) \Leftarrow f_2(hx, F(g_1 x), \ldots, F(g_n x)) \end{array} \right\rangle$$

used in the proof of Theorem 4.3 have the form $hg_{i_1} \ldots g_{i_k} d$, where $i_1 \ldots i_k$ are all finite sequences of numbers from 1 to n. The job of the semantic transformer for h is to check if a given sequence gives a solution. Other determinant nodes just propagate all properties to the root.

We can go the other way letting the information flow backwards and accumulate in all the images of d's. Now h is needed to be the argument of g_i's:

$$\left\langle \begin{array}{l} f_1(F(g_1 hd), \ldots, F(g_n hd)); \\ F(x) \Leftarrow f_2(x, F(g_1 x), \ldots, F(g_n x)) \end{array} \right\rangle.$$

Take the same as in Theorem 4.3 semi-lattice L and the initial property $\perp$. Having the semantic transformers "reversed" we reduce Post's correspondence problem to a backwards analysis problem, and thus prove the latter to be undecidable. □

5.4 Solution

Theorem 4. *If the semi-lattice L is finite and all property transformers are distributive, the backwards analysis problem can be solved using the following marking algorithm.*

An *h-marking* of a scheme S is a mapping μ that assigns a property (called *h-mark*) $\mu a \in L$ to each address a in S. The initial h-marking μ_0 assigns the h-mark h_0 to the entry address and the h-mark $\perp$ to all other addresses. The initial h-marking and all h-markings which can be obtained from it by serial application of *h-marking rules* are called *reachable*. An application of an h-marking rule to an address a consists of the following:

1. If $S \downarrow a = f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}_b$, then for all $i \in \{1, \ldots, n\}$ the h-mark $\mu a.i$ of the address $a.i$ is replaced by the h-mark $\mu a.i \sqcup Sem^{\downarrow}(f, i)(\mu a)$.

2. If $S \downarrow a = F(t_1, \ldots, t_n)$, where $F \in \mathcal{F}_d$, then the h-mark $\mu[F]$ of the address $[F]$ is replaced by the h-mark $\mu[F] \sqcup \mu a$.

3. If a is an address of some occurrence of the i-th formal parameter in some F's body, then for each address b of a call to F, that satisfies $\mu b \neq \perp$, the h-mark $\mu b.i$ of the address $b.i$ is replaced by the h-mark $\mu b.i \sqcup \mu a$.

A *stationary h-marking* is a reachable h-marking that cannot be changed by any application of the h-marking rule. This stationary marking gives the precise solution of the backwards analysis problem. By the premises of the theorem, the chain condition, and the monotony of all property transformers the stationary h-marking exists and it is unique. See Appendix A.1 for a formal proof.

For any backwards problem with distributive transformers we have a solution algorithm for free. A natural marking technique comes out of it when we apply it to the example of the needed/unneeded analysis.

6 Formal parameter dependence

If the value of an actual parameter in each call to a symbol F equals the value of a term t, then any occurrence of the corresponding formal parameter in the body of F can be replaced by the term t. Let us describe and solve the formal parameter dependence analysis problem as a flow analysis problem. This problem can be formulated neither as forwards nor as backwards analysis problem in the presented framework since any term in the approximation sequence does not contain formal parameters and, therefore, no formal parameter properties can be taken into account.

We formulate a data flow analysis problem for the directed graph $Graph(S)$ obtained from a scheme S in the following way. The nodes of this graph will be the entry address and the addresses of bodies of the defined symbols of the scheme S. The edges will be the call addresses. We draw an edge a from an address b to an address $[F]$, if a is an address of a call to F that occurs in the term at address b.

Let X be a finite set of variables, $X \subset \mathcal{X}$. We use below the semi-lattice $\mathcal{L}(X)$ of context free grammars G that describes the finite languages $L(G)$ of term equalities of the form $x = t$, where $x \in X$ and $t \in \mathcal{T}(X)$. These grammars have the terminal set

$$\Sigma = \mathcal{F}_b \cup X \cup \{\equiv, (,), ,\}$$

and rules of the following three forms

1. $S \to x \equiv A$ with nonterminals S, A; S is the initial nonterminal of the grammar; $x \in X$;

2. $A \to x$ with a nonterminal A; $x \in X$;

3. $A \to f(A_1, \ldots, A_n)$ with nonterminals $A, A_1, \ldots, A_n$ and a terminal $f \in \mathcal{F}_b$.

In order to ensure the finiteness of a grammar's language let us require the graph of the dependencies of the grammar's nonterminals to be acyclic.

Example:

$$\begin{array}{l} S \to x \equiv A_1 | y \equiv A_1 | z \equiv A_2 \\ A_1 \to x | y | f(A_2, A_3) \\ A_2 \to z | g(A_3) \\ A_3 \to a \end{array} \quad \text{represents} \quad \left\{ \begin{array}{l} x \equiv x, y \equiv y, \\ x \equiv y, x \equiv f(z,a), y \equiv f(z,a), \\ x \equiv f(g(a),a), y \equiv f(g(a),a) \\ z \equiv z, z \equiv g(a) \end{array} \right\}.$$

Such a grammar G can be reduced in linear time to a *reduced* grammar $Red(G) = \langle N, \Sigma, S, P \rangle$ that satisfies the following conditions

- $L(G) = L(Red(G))$ is a finite language,
- $\forall A, B \in N \; L(A) \cap L(B) = \emptyset$,
- $\forall A \; \in N \; L(A) \neq \emptyset$,
- $\forall A \in N \; \exists \alpha, \beta \in \{N \cup \Sigma\}^* \; S \xrightarrow{*} \alpha A \beta$

The *meet* operation $\sqcap$ on reduced grammars corresponds to the language intersection and can be defined in the following way. Let $G_i = \langle N_i, \Sigma, S_i, P_i \rangle$ for $i = 1,2$ be two reduced grammars. We define $G = \langle N, \Sigma, S, P \rangle$, where $N = N_1 \times N_2, S = \langle S_1, S_2 \rangle, P = \{S \to x \equiv \langle A_1, A_2 \rangle | (S_i \to x \equiv A_i) \in P_i, i = 1,2\} \cup \{\langle A_1, A_2 \rangle \to f(\langle B_1^1, B_1^2 \rangle, \ldots, \langle B_n^1, B_n^2 \rangle) | (A_i \to f(B_1^i, \ldots, B_n^i)) \in P_i, i = 1,2\} \cup \{\langle A_1, A_2 \rangle \to x | (A_i \to x) \in P_i, i = 1,2, x \in X\}$. Finally, we define $G_1 \sqcap G_2 \stackrel{def}{=} Red(G)$.

Lemma 1. $L(G_1 \sqcap G_2) = L(G_1) \cap L(G_2)$.

We denote by $\mathcal{L}(X)$ the set of all reduced grammars augmented by a new element $\mathbf{1}$ satisfying $\mathbf{1} \sqcap G = G \sqcap \mathbf{1} = G$. The partial order $\sqsubseteq$ on $\mathcal{L}(X)$ is introduced by the definition $G_1 \sqsubseteq G_2 \stackrel{def}{=} G_1 \sqcap G_2 = G_1$. The grammar $\{S \to x \equiv A_x, A_x \to x \mid x \in X\}$ will be denoted by $\mathbf{O}$, $L(\mathbf{O}) = \{x \equiv x \mid x \in X\}, \forall\, G \in \mathcal{L}(X)\; \mathbf{O} \sqsubseteq G$. We say that a nonterminal A of a grammar *knows* a term t, if $A \xrightarrow{*} t$ holds. For a grammar G and a term t, the grammar $Add(G,t)$ that has a nonterminal A which knows t can be built in the following way.

If the grammar G already has a nonterminal that knows t, or $G = \mathbf{1}$, then $Add(G,t) \stackrel{def}{=} G$. Otherwise, if t is a variable x, then add a new nonterminal A and a rule $A \to x$ to the grammar G; if $t = f(t_1, \ldots, t_n)$, build the grammar $G' = Add(Add(\ldots Add(G, t_1), \ldots), t_n)$ and add a new nonterminal A and the rule $A \to f(A_1, \ldots, A_n)$ to the grammar G', where for $i = 1, \ldots, n$ A_i is the nonterminal in G' which knows the term t_i.

For an address a of a call $F(t_1, \ldots, t_n)$ in a scheme we define the grammar transformer

$$[\![call(a)]\!] : \mathcal{L}(Var(a) \cup Var(\Lambda)) \to \mathcal{L}(Var([F]) \cup Var(\Lambda)).$$

If $G = \mathbf{1}$, then $[\![call(a)]\!]G = G$. Otherwise, let $G' = Add(\ldots Add(G, t_1) \ldots t_n)$, let A_i be the nonterminal of G' that knows the term t_i, and let B_i be the nonterminal of G' that knows the i-th formal parameter x_i of the symbol $F (i = 1, \ldots, n)$. For all $i = 1, \ldots, n$, if $A_i \neq B_i$, then delete the rules $S \to x_i \equiv B_i$ and $B_i \to x_i$ from the grammar G' and add the new rules $S \to x_i \equiv A_i$ and $A_i \to x_i$. Finally, define $[\![call(a)]\!]G \stackrel{def}{=} Red(G')$.

Lemma 2. *For all addresses a of a scheme, $[\![call(a)]\!]$ is a distributive grammar transformer, i.e.*

$$[\![call(a)]\!](G_1 \sqcap G_2) = [\![call(a)]\!]G_1 \sqcap [\![call(a)]\!]G_2.$$

Now we formulate a data flow analysis problem for $Graph(S)$ by stating

- the initial marking μ_0 that associates the grammar $\mathbf{O}$ to the entry node Λ and the grammar $\mathbf{1}$ to the other nodes of $Graph(S)$,
- the semantic function which associates a distributive grammar transformer $[\![call(a)]\!]$ to any edge a of $Graph(S)$.

Let w be a path in $Graph(S)$, i.e. a sequence of adjacent edges, and let

$$\Phi_w(G) = \begin{cases} G, & \text{if } w \text{ does not contain any edge,} \\ [\![call(a)]\!]\Phi_{w'}(G), & \text{if } w = w'a, \text{ where } a \text{ is an address of a call.} \end{cases}$$

Our data flow analysis problem consists of finding the 'meet over all path solution'

$$mop([F]) = \bigsqcap_{w \in W} \Phi_w(\mathbf{O})$$

for all nodes $[F]$ in $Graph(S)$, where W is the set of all paths in $Graph(S)$ from the entry node to the node $[F]$.

It is well known [KU77], that a stationary marking μ of $Graph(S)$ can be effectively constructed such that $\mu[F] = mop([F])$ for all nodes $[F]$, and hence, if the condition $(x \equiv t) \in L(\mu[F])$ is valid for a node $[F]$ and for a formal parameter x of F, then $S \sim S[a \leftarrow t]$ holds for all addresses a of occurrences of x in the body of the symbol F. For example, for the stationary marking μ of the scheme

$$\left\langle \begin{array}{l} F(h,h); \\ F(x,y) \Leftarrow \mathbf{if}(px, F(fx, fy), gx) \end{array} \right\rangle$$

$(x \equiv y) \in L(\mu[F])$ holds, which means that actual parameter values coincide in all calls to F.

7 Discussion and Conclusions

Our forwards analysis can be classified to be an abstract interpretation according to Patrick and Radhia Cousot [CC92]. In abstract interpretation all primitive functions get abstracted such that they safely approximate the concrete primitives. Abstract functions constitute the abstract domain. The abstract functions correspond to our deductive semantic functions for forwards analysis. Once approximations are defined, it is possible to find the abstract value for a defined function by computing the fix-point of its definition in the abstract domain. Hughes [Hug87] argued that when doing the full abstract interpretation forwards analysis it takes exponential time in the worst case. However, it is possible to make forwards analysis demand-driven, that is not to compute functions by fix-point iteration entirely, but rather check them only on certain values. Jones and Mycroft have introduced *minimal function graphs* [JM86] to implement the idea. In our framework we can simply choose the analysis that suits a particular problem better. The advantage is that in spite of a more operational nature of backwards analysis, both frameworks are formulated to be dual.

As for complexity, observe that both algorithms formulated are nondeterministic. The worst case is when for a scheme of the size n the algorithm (in either analysis) changes each mark as many times as the length of the longest chain in the semi-lattice $\langle L, \sqcup, \bot \rangle$. Since such an algorithm randomly applies the marking rules to the addresses until stabilisation, its complexity gets another factor of n. Let $l = |L|$. In case of forwards analysis the marks are elements of $L^k \to L$ $(k \leq r)$, where r is the maximal rank of all basic functions, and in case of backwards analysis the marks belong to L. This gives us $\mathcal{O}(n^2 * l^{l^r})$ as the worst case estimate for the forwards and $\mathcal{O}(n^2 * l)$ for the backwards analysis problem. The subexpressions l^{l^r} and l instantiate to 2^{2^r} and 2 in the examples of strictness analysis and needed/unneeded analysis respectively. These are rough estimates. In practice one factor of n disappears once a rule application strategy is settled.

The forwards and backwards analyses can be generalised to higher-order functions. It is possible by allowing the property transformers to be higher-order as well. However, the complexity of the marking algorithms becomes extremely high.

Acknowledgements

Thanks are due to David Sands and John Hughes for the constructive suggestions on both contents and presentation of the paper. Thank you to Sebastian Hunt for providing us with his unpublished note [Hun90] which helped us track other work on algebraic semantics.

A Appendix

A.1 Proof of Theorem 4

If the semi-lattice L is finite and all property transformers are distributive, the backwards analysis problem can be solved by the h-marking algorithm.

Proof. Let us first prove the stationary h-marking μ_s for any backwards analysis problem to exist and be unique (Theorem 5), and then show that it solves the problem by establishing $H^{\downarrow}(a) \sqsubseteq \mu_s a$ (Theorem 5) and $\mu_s a \sqsubseteq H^{\downarrow}(a)$ (Theorem 6) for all addresses a in a scheme S. $\Box$

Theorem 5. *The stationary h-marking μ_s defined in Section 5.4 exists and it is unique for any backwards analysis problem for a program S. $H^{\downarrow}(a) \sqsubseteq \mu_s a$ holds for all addresses a in S.*

Proof. Note that a marking rule application to an address might only increase the mark of an address. Since the set of addresses of S is finite and so are all strictly increasing chains, any marking terminates provided it changes a mark at every application of a marking rule.

Let $\sqsubseteq$ be a partial order introduced on marks of S by

$$\mu_1 \sqsubseteq \mu_2 \stackrel{dfn}{\equiv} \forall a \; \mu_1 a \sqsubseteq \mu_2 a.$$

Observe that the stationary marking is a solution of the equation system

1. $\mu a.i = \mu a.i \sqcup Sem^{\downarrow}(f,i)(\mu a)$
   for all addresses a of S such that
   $S \downarrow a = f(t_1,\dots,t_n)$, where $f \in \mathcal{F}_b, i \in \{1,\dots,n\}$,
2. $\mu[F] = \mu[F] \sqcup (\bigsqcup_{a\in C} \mu a)$
   for all defined symbols F of S,
   where C is the set of the addresses where F is called in S,
3. $\mu c.i = \mu c.i \sqcup (\bigsqcup_{a\in A} \mu a)$
   for all such addresses c of calls to F in S that
   $\mu c \neq \perp$, where A is the set of addresses of occurrences of the ith formal parameter of F in F's body, $1 \leq i \leq r(F)$,

that satisfies $\mu_0 \sqsubseteq \mu_s$. Let us show that any reachable h-marking of S does not exceed any solution μ of the system such that $\mu_0 \sqsubseteq \mu$. For the initial h-marking it holds by the condition for μ. Suppose $\mu' \sqsubseteq \mu$ for a reachable h-marking μ', and the h-marking μ'' is obtained from μ' by applying a marking rule to an address a. If $S \downarrow a = f(t_1,\dots,t_n)$, where $f \in \mathcal{F}_b$, then $\forall i = 1,\dots,n$ $\mu''a.i = \mu'a.i \sqcup Sem^{\downarrow}(f,i)(\mu'a) \sqsubseteq \mu a.i \sqcup Sem^{\downarrow}(f,i)(\mu a) = \mu a.i$. If $S \downarrow a = F(t_1,\dots,t_n)$, where $F \in \mathcal{F}_d$, then $\mu''[F] = \mu'[F] \sqcup \mu'a \sqsubseteq \mu[F] \sqcup \mu a = \mu[F]$. If a is an addresses of the ith formal parameter of F in F's body, $1 \leq i \leq r(F)$, and c is the address of a call to F for which $\mu'c \neq \perp$, then $\mu''c.i = \mu'c.i \sqcup \mu'a \sqsubseteq \mu c.i \sqcup \mu a = \mu c.i$. Other addresses a do not change their marks when building μ'' out of μ', therefore $\mu''a = \mu'a \sqsubseteq \mu a$. So, the stationary h-marking is the least solution of the system out of those solutions μ that satisfy $\mu_0 \sqsubseteq \mu$. If μ_1, μ_2 are two stationary h-markings of S, it implies $\mu_1 \sqsubseteq \mu_2$ and $\mu_2 \sqsubseteq \mu_1$, i.e. $\mu_1 = \mu_2$.

In order to prove $\forall a\ H^{\downarrow}(a) \sqsubseteq \mu_s a$ it suffices to show

$$\forall a\ \forall b \in Image(a)\ h^{\downarrow}(b) \sqsubseteq \mu_s a$$

for stationary h-marking μ_s. Let us show it by induction on the address length b.

Let b have length 0. Then $b = \Lambda$, and $h^{\downarrow}(b) = h_0$. On the other hand, since $b \in Image(a)$, we have either $a = \Lambda$ and $\mu_s(\Lambda) \sqsupseteq \mu_0(\Lambda) = h_0$, or a is an address of either the body of a defined symbol or a formal/actual parameter occurrence. In the latter case $\mu_s a \sqsupseteq \mu_s(\Lambda) \sqsupseteq \mu_0(\Lambda) = h_0$.

Let $\forall a\ \forall b \in Image(a)\ h^{\downarrow}(b) \sqsubseteq \mu_s a$ hold for all addresses b shorter than n. Prove the statement for an arbitrary $b \in Image(a)$ of length n. It follows that $b = c.i$ for some c of length $n-1$. If there is an address d such that $a = d.i, S \downarrow d = f(t_1, \ldots, t_k)$ for $f \in \mathcal{F}_b$, then the induction hypothesis $h^{\downarrow}(c) \sqsubseteq \mu_s d$ implies $\mu_s a = \mu_s a \sqcup Sem^{\downarrow}(f,i)(\mu_s d) \sqsupseteq Sem^{\downarrow}(f,i)(h^{\downarrow}(c)) = h^{\downarrow}(b)$. Otherwise, there is a sequence of addresses $a = a_1, a_2, \ldots, a_k$ such that all of them but last one are addresses of the bodies of defined symbols or parameter occurrences. Besides, $b \in Image(a_i)\ \ \forall\ i = 1, \ldots, k-1\ \mu_s a_i = \mu_s a_i \sqcup \mu_s a_{i+1}$ holds for $i = 1, \ldots, k$, i.e. $\forall\ i = 1, \ldots, k-1\ \mu_s a_i \sqsupseteq \mu_s a_{i+1}$, and for a_k there exists an address d such that $a_k = d.i, S \downarrow d = f(t_1, \ldots, t_k)$ with some $f \in \mathcal{F}_b$, and therefore $\mu_s a \sqsupseteq \mu_s a_k = \mu_s a_k \sqcup Sem^{\downarrow}(f,i)(\mu_s d) \sqsupseteq Sem^{\downarrow}(f,i)(h^{\downarrow}(c)) = h^{\downarrow}(b)$. □

We will establish now the approximation in the other direction. Recall that a function $\varphi : L \to L$ is *distributive*, iff

$$\forall\ x, y \in L\ \varphi(x \sqcup y) = \varphi(x) \sqcup \varphi(y).$$

Theorem 6. *If the semi-lattice L is finite and for all $f \in \mathcal{F}_b$ and for all $i = 1, \ldots, r(f)$ the property transformer $Sem^{\downarrow}(f,i) : L \to L$ is distributive, then for the stationary h-marking μ_s and for all addresses a of a program S the inequality $\mu_s a \sqsubseteq H^{\downarrow}(a)$ holds.*

Proof. For a proof it suffices to show that $\forall a\ \mu a \sqsubseteq H^{\downarrow}(a)$ for all reachable h-markings μ. For the initial h-marking μ_0 we have $\mu_0(\Lambda) = h_0 \sqsubseteq H^{\downarrow}(\Lambda)$, since $\Lambda \in Image(\Lambda)$ and $h^{\downarrow}(\Lambda) = h_0$. For other addresses a, $\mu_0 a = \bot \sqsubseteq H^{\downarrow}(a)$.

Let $\forall a\ \mu a \sqsubseteq H^{\downarrow}(a)$ hold for a reachable h-marking μ, and the h-marking μ' is obtained out of μ by applying the marking rule to a. Then in the case $S \downarrow a = f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}_b$, we have

$$\begin{gathered}
\forall\ i \in \{1, \ldots, n\}\ \mu' a.i = \mu a.i \sqcup Sem^{\downarrow}(f,i)(\mu a) \sqsubseteq \\
H^{\downarrow}(a.i) \sqcup Sem^{\downarrow}(f,i)(H^{\downarrow}(a)) = \\
H^{\downarrow}(a.i) \sqcup Sem^{\downarrow}(f,i)(\bigsqcup_{b \in Image(a)} h^{\downarrow}(b)) = \\
H^{\downarrow}(a.i) \sqcup (\bigsqcup_{b \in Image(a)} Sem^{\downarrow}(f,i)(h^{\downarrow}(b))) \\
= H^{\downarrow}(a.i) \sqcup (\bigsqcup_{b \in Image(a.i)} h^{\downarrow}(b)) = \\
H^{\downarrow}(a.i) \sqcup H^{\downarrow}(a.i) = H^{\downarrow}(a.i).
\end{gathered}$$

Let now $S \downarrow a = F(t_1, \ldots, t_n)$, where $F \in \mathcal{F}_d$. Note that

$$H^{\downarrow}(a) = \bigsqcup_{b \in Image(a)} h^{\downarrow}(b) \sqsubseteq \bigsqcup_{b \in Image([F])} h^{\downarrow}(b) = H^{\downarrow}([F]),$$

because any call image is a body image as well. Thus,

$$\mu'[F] = \mu[F] \sqcup \mu a \sqsubseteq H^{\downarrow}([F]) \sqcup H^{\downarrow}(a) \sqsubseteq$$
$$H^{\downarrow}([F]) \sqcup H^{\downarrow}([F]) = H^{\downarrow}([F]).$$

Let a is the address of the ith formal parameter of F, and c is any of F-call addresses for which $\mu c \neq \perp$. Observe that $H^{\downarrow}(a) = \bigsqcup_{b \in Image(a)} h^{\downarrow}(b) \sqsubseteq \bigsqcup_{b \in Image(c.i)} h^{\downarrow}(b) = H^{\downarrow}(c.i)$, since any formal parameter image is a corresponding actual parameter image in a call with a non-empty image set. Hence, $\mu' c.i = \mu c.i \sqcup \mu a \sqsubseteq H^{\downarrow}(c.i) \sqcup H^{\downarrow}(a) \sqsubseteq H^{\downarrow}(c.i) \sqcup H^{\downarrow}(c.i) = H^{\downarrow}(c.i)$. For other addresses d of S we have $\mu' d = \mu d \sqsubseteq H^{\downarrow}(d)$. $\square$

A.2 Proof of Theorem 2

If the semi-lattice L is finite and all property transformers are distributive in each argument, the forwards analysis problem can be solved by the d-marking algorithm.

Proof. The proof technique for this theorem is similar to the proofs in Appendix A.1 for the backwards analysis. To ensure the termination we need no infinitely ascending chains in the semi-lattice of functions on L, which is guaranteed by the finiteness of L. The part $D(a) \sqsubseteq \mu a$ is proved by showing $\forall a\ \forall\ n \geq 0 \quad d(a, n) \sqsubseteq \mu_s a$ (induction on n). The proof of $\mu a \sqsubseteq D(a)$ is done by induction on the d-marking step. Start off with the least marking, and at every step add nothing that "exceeds" $D(a)$. It is slightly more complicated since the semantic properties are functions, and the deductive semantic function $Sem^{\uparrow}$ is second-order. On the other hand, this is a possibility for a speed-up: the property transformer used in the marking can itself be changed during the marking.

Let us prove that $\mu a \sqsubseteq D(a)$ for any address a and any reachable marking μ in a program S. For the initial marking μ_0 it holds since $\mu_0 a = \perp\!\!\!\perp \sqsubseteq D(a)$ for any term but a variable at address a, and $\mu_0 a = Pr_i = D(a)$ for an occurrence of a variable x_i at a. Let $\forall a.\ \mu a \sqsubseteq D(a)$ hold for a reachable marking μ, and the marking μ' is obtained from μ by applying a marking rule to an address a. If $S \downarrow a = f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}_b$, then $\mu' a = \mu a \sqcup Sem^{\uparrow}(f)(\mu a.1, \ldots, \mu a.n) \sqsubseteq D(a) \sqcup Sem^{\uparrow}(f)(D(a.1), \ldots, D(a.n)) = D(a) \sqcup D(a) = D(a)$. In case $S \downarrow a = F(t_1, \ldots, t_n)$, where $F \in \mathcal{F}_d$, we have $\mu' a = \mu a \sqcup (\mu[F])(\mu a.1, \ldots, \mu a.n) \sqsubseteq D(a) \sqcup D([F])(D(a.1), \ldots, D(a.n)) = D(a) \sqcup D(a) = D(a)$. $\square$

References

[CC77] P. Cousot, R. Cousot. *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.* In Conference Records of the 4^{th} Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238–252. Los Angeles, California, 1977.

[CC92] P. Cousot, R. Cousot. *Abstract interpretation frameworks.* Journal of Logic and Computation, 2(4):511–547, August 1992.

[CC93] P. Cousot, R. Cousot. *Galois connection based abstract interpretations for strictness analysis.* In D. Bjorner, M. Broy, and I.V. Pottosin, editors, Proceedings of the International Conference on Formal Methods in Programming and their Applications, Academgorodok, Novosibirsk, Russia, 28 June – 2 July 1993. Lecture Notes in Computer Science 735, pages 98–127. Springer-Verlag, 1993.

[CC95] P. Cousot, R. Cousot. *Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation.* In Conference Record of FPCA '95 SIGPLAN/SIGARCH/WG2.8 Conference on Functional Programming and Computer Architecture, pages 170–181, La Jolla, California, U.S.A., 25–28 June 1995. ACM Press, New York, U.S.A, 1995.

[DW90] K. Davis, P. Wadler. *Strictness analysis in 4d.* In Glasgow Functional Programming Workshop, 1990.

[GL71] S.J. Garland, D.C. Luckham. *Program schemes, recursion schemes and formal languages.* Report UCLA-ENG-7154, June 1971, School of Engineering and Applied Science, University of California, Los Angeles, 1971.

[Gue81] I. Guessarian. *Algebraic Semantics.* Volume 99 of LNCS, 1981. Springer-Verlag.

[Hug87] J. Hughes. *Backwards Analysis of Functional Programs.* In Bjørner and Ershov, editors, IFIP Workshop on Partial Evaluation and Mivxed Computation, 1987.

[HL92] J. Hughes, J. Launchbury. *Reversing Abstract Interpretations.* In European Symposium on Programming, volume 582 of LNCS, Rennes, 1992. Springer-Verlag. Also to appear in Science of Computer Programming.

[Hun90] S. Hunt. *Projection analysis and stable functions.* An unpublished note.

[JM79] N.D. Jones, S.S. Muthnik. *Flow analysis and optimization of LISP-like programs.* In the 6^{th} ACM POPL, pages 244–256, 1979.

[JM86] N. D. Jones, A. Mycroft. Data flow analysis of applicative programs using minimal function graphs. In *Proceedings of the Thirteenth ACM Symposium on Principles of Programming Languages*, pages 296–306, St. Petersburg, Florida, 1986.

[Klo91] J.W. Klop. Term Rewriting Systems. In S. Abramsky, Dov M. Gabbay, T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6, Oxford University, 1991. Oxford University Press.

[KU77] J.B. Kam, J.D. Ullman. *Monotone data flow analysis frameworks.* Acta Informatica *7 : 3*, pages 305–318, 1977.

[Myc80] A. Mycroft. *The theory and practice of transforming call-by-need into call-by-name.* In: Robinet B. ed., Proceedings of the Fourth International Symposium on Programming, Paris, 22–24 April 1980, LNCS *83*, pages 270–281, 1980.

[Myc81] A. Mycroft. *Abstract Interpretation and Optimizing Transformations for Applicative Programs.* Ph.D. Dissertation, CST-15-81, Department of Computer Science, University of Edinburgh, December 1981.

[Pos46] E.L. Post. *A variant of a recursive unsolvable problem.* Bull. of Am. Math. Soc., Vol. 52, 4, pages 264–268, 1946.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages. An Introduction.* The MIT Press, 1993.