Optimal Data Flow Analysis via Observational Equivalence

by

Bernhard Steffen

Optimal Data Flow Analysis
via Observational Equivalence

ECS-LFCS-89-82
(also published as CSR-300-89)

LFCS Report Series

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

May 1989

Optimal Data Flow Analysis via Observational Equivalence

Bernhard Steffen *

Abstract

In [18] a three level model was presented to establish a concept of *completeness* or *optimality* for data flow analysis algorithms in the framework of *abstract interpretation* [2]. The notion of *observational equivalence* which we introduce here generalizes the idea of the three level model, which can only deal with hierarchies of abstract interpretations. Investigating this more general notion, it actually turns out that the three level model is general in a *theoretical* sense: it determines the *most abstract* computation level which delivers complete results. However, consideration of other aspects of data flow analysis profit from the extra generality of our observation directed approach. For example the completeness or optimality proof for a "real life" optimizer could be shortened significantly this way [1].

1 Motivation

Large software projects require sophisticated environments to support well structured and reliable programming. Unfortunately all too often developers are unwilling to use high level programming languages (which provide these facilities) because of efficiency reasons. Consequently, optimizing compilers are constructed to increase their practicability. The quality of an optimizing compiler depends on data flow analysis. An important step in the systematization of data flow analysis has been taken by Cousot/Cousot [2] by introducing the notion of abstract interpretation. Abstract interpretation is a framework that allows observation of program behaviour at different levels of abstraction. It can be used to define *correctness* and *optimality* of a data flow analysis algorithm.

Correctness has been studied in [2] using a two level model of abstract interpretations. However, to analyse *optimality* of a data flow analysis algorithm the second level of this model has been split into two parts [18]. The resulting

*Laboratory for Foundations of Computer Science, University of Edinburgh – The author is supported by the Science and Engineering Research Council grant GC/D69464

[1]This paper also appeared in [19].

model, therefore, has three levels: a basic level (related to the static semantics of [2]), a computation level and an observation level. The advantage of this model is to separate the environment of a data flow analysis algorithm, given by a full semantics and questions about the program behaviour, from the specification of an algorithm answering these questions.

In this paper we discuss the generality and limitations of the three level model. To do so we first adapt the notion of *observational equivalence* [5, 16] for models of abstract interpretations. Second, we generalize the notion of a fully abstract model (cf. [8]) in the sense of [18], which deals only with the hierarchical situation of the three level model, to a situation without this restriction. Third, we introduce a product construction between models of abstract interpretations. This construction yields an abstract semantics that embodies both of its argument models in a "synchronized" way. It is appropriate for discovering similarly behaving parts of arbitrary models of abstract interpretations. And fourth, we use the "synchronization" mechanism delivered by the product construction to prove our main result: *fully abstract models* provide the *most abstract* computation levels that are observationally equivalent to a given basic level (**Equivalence Theorem 3.13**).

As a consequence of this result we conclude the generality of the *three level model*, since, given an environment for a data flow analysis algorithm, the (in a theoretical sense) *ideal* computation level (which is given by the fully abstract model), can be determined within the three level model.

On the other hand the practical *implementation* of a data flow analysis algorithm requires a specific representation of the *abstract domain*, the algorithm operates on. Such a representation usually leads to an abstract interpretation (i.e. computation level) which cannot be seen as an abstraction of the full semantics. Thus, the concrete implementation of a data flow analysis algorithm does not fit into the three level model, but it can be handled in our observation directed approach. This has been successfully used in proving the completeness or optimality of a "real life" optimizer.

The theoretical development in this paper is accompanied by the very concrete development of a special data flow analysis which we call *order analysis.* On the one hand this analysis is restricted enough to allow an efficient and optimal implementation. On the other hand it is general enough to illustrate the main aspects of the theory.

2 Preliminary Definitions

We directly introduce the syntax of our programming language as the class of all (nondeterministic) flow graphs over a set $\mathbf{N}$, which represents the occurrences of elementary statements. A *flow graph* over $\mathbf{N}$ is a quadruple $G=(N, E, \mathrm{s}, \mathrm{e})$, where (N,E) is a connected and directed graph with node set $N \subseteq \mathbf{N}$ and edge set $E \subseteq N \times N$. $\mathbf{s}$ and $\mathbf{e}$ (*start* and *end* node) denote a special element of N without predecessor and successor nodes respectively. We write $\mathbf{P}(G)$ for the set of all finite paths from $\mathbf{s}$ to $\mathbf{e}$ of a given flow graph G, $\mathbf{FG}$ for the set of all flow graphs over $\mathbf{N}$ and $\mathbf{LFG}$ for the set of all *linear* flow graphs over $\mathbf{N}$, i.e. $\mathbf{LFG} = \{G \in \mathbf{FG} \mid |\mathbf{P}(G)| = 1\}$.

Remark 2.1 Paths will be identified with linear flow graphs, i.e. $\mathbf{P}(G) \subseteq \mathbf{LFG}$.

The *semantics* of flow graphs will now be introduced as the *MOP-solution* in the sense of Kam and Ullman [7]. This is a form of operational semantics which simply collects all "final state descriptions" that are derivable by executing finite paths (a denotational approach, such as the *MFP-solution* in the sense of Kam and Ullman, would deliver exactly the same results). For this we need a *complete semi-lattice* $(C, \sqcup, \sqsubseteq, \perp, \top)$ with bottom element $\perp$ and top element $\top$. The elements of C serve as descriptions for the situations which may occur at an arbitrary program point.

Definition 2.2

1. *Let* $[\![\ ]\!]_l : \mathbf{N} \to (C \to C)$ *be a function, which relates to every node* $n \in \mathbf{N}$ *an additive function* $[\![n]\!]_l$ *from* C *to* C*, i.e.:*

   $$\forall n \in \mathbf{N}\ \forall T \subseteq C.\ [\![n]\!]_l(\sqcup T) = \sqcup \{ [\![n]\!]_l(c) \mid c \in T \}$$

   In this case, we call $([\![\ ]\!]_l, C)$ *an* abstract interpretation *or a* local abstract semantics.

2. *Let* $([\![\ ]\!]_l, C)$ *be a local abstract semantics and* $[\![\ ]\!] : \mathbf{FG} \to (C \to C)$ *be the* globalization *of* $[\![\ ]\!]_l$*, i.e.:* $\forall G \in \mathbf{FG}\ \forall c \in C.$

   $$[\![G]\!](c) =_{df} \begin{cases} [\![n_1]\!]_l; \dots; [\![n_k]\!]_l(c) & \textit{if } G = (n_1, \dots, n_k) \in \mathbf{LFG} \\ \sqcup \{ [\![P]\!](c) \mid P \in \mathbf{P}(G) \} & \textit{otherwise} \end{cases}$$

   (here "$X;Y(..)$" means "$Y(X(..))$"). Then $([\![\ ]\!], C)$ *is called the* (global) abstract semantics *induced by* $([\![\ ]\!]_l, C)$.

This delivers a uniform framework for describing the state transformation semantics of an imperative programming language (in the sense of the collecting semantics [12]) as well as the abstract semantics which is induced by a data flow analysis algorithm, see [17] or [18].

We illustrate the concepts and results presented in this paper by an example which we call *order analysis.* This analysis might be especially useful for physicists to optimize computations by suppressing higher order terms. Think of approximations using the Taylor-development, perhaps in particular of applications in perturbation theory, where the behaviour of very small perturbations is studied. There, physicists often reduce their attention to the first two terms, the constant term and the first order term, because the higher order terms can be considered as insignificant in those contexts. Using such application dependent information leads to powerful program transformations and therefore to fast code. The order analysis delivers a simple kind of finite interpretation which allows an efficient and optimal data flow analysis. But it is also complex enough to illustrate the main ideas developed in this paper.

Example: Let V be a set of variables, Π be the set of polynomials p over unknowns $x, y, .. \in V$, i.e. p is of the form:

$$a_1 * x^{j_1} * y^{k_1} * ... + ... + a_i * x^{j_i} * y^{k_i} * ... + ... + a_n * x^{j_n} * y^{k_n} * ...$$

(the kind of the coefficients a_i is unimportant for our considerations) and $\mathbf{N}$ be the set of all assignments of the form $v := ex$ $(v \in V, ex \in \Pi)$. Then our programming language L consists of the set of all nondeterministic flow graphs whose nodes are occurrences of elements $n \in \mathbf{N}$. To keep things simple we furthermore assume that there is only one variable x and that this variable is initialized by a value x_0 at run time.

Note: The order analysis developed further on is important whenever x_0 is known to be very small (i.e.: $x_0 \ll 1$), a very reasonable assumption in many physical applications.

Finally, we need the set Π_0 of all polynomials over x_0 and the function $o : \Pi_0 \rightarrow \omega$ that assigns to all its arguments the exponent of the highest x_0–power that can be factored out. This setup allows to define abstract semantics on several levels of abstraction. We start with two of them:

First, $\mathcal{S}_B =_{df} ([\![\ ]\!]_B, C_B)$, where $C_B =_{df} \mathcal{P}(\Pi_0)$ with $\sqsubseteq \; = \; \subseteq$ ($\mathcal{P}(\Pi_0)$ denotes the powerset of Π_0) and $[\![x := t]\!]_B(\pi) = \{t[t_0/x] \mid t_0 \in \pi\}$, for all $\pi \in \mathcal{P}(\Pi_0)$ and $(x := t) \in \mathbf{N}$ ($t[x/y]$ is the polynomial that results from the substitution of all occurrences of y by x in t). $\mathcal{S}_B$ describes the symbolic evaluation of a program. Here the set of all possible program states is characterized by means of the collection of those program terms over the initial value x_0 that can be represented by x at a certain program point. We choose this semantics as our basic level (indicated by the index "B"), because it is powerful enough to represent the *complete behaviour* of any program in L .

Second, $\mathcal{S}_e =_{df} ([\![\ ]\!]_e, C_e)$ reduces attention to the order of the variable x wrt the initial value x_0. It is defined by $C_e =_{df} \omega \cup \{\omega\}$ with $\sqsubseteq \, = \, \geq$ and $[\![x := t]\!]_e(n) = o(t[{x_0}^n/x])$, for all $(x := t) \in \mathbf{N}$ and $n \in \omega$. We choose the index "e" here because this semantics only considers the (smallest) exponent of x occurring in a given term. Later we will see that $\mathcal{S}_e$ optimally reflects the order analysis and that it is possible to further abstract the underlying semantics without loosing optimality, whenever the minimal order of insignificance is known. In this stage of the development of our example one observation is important:

$\mathcal{S}_e$ can be regarded as an *abstracted* version of $\mathcal{S}_B$.

This can be formalized using the notion of an abstraction function:

Definition 2.3 *Let* $\mathcal{S}_1 = ([\![\ ]\!]_1, C_1)$ *and* $\mathcal{S}_2 = ([\![\ ]\!]_2, C_2)$ *be two abstract semantics. We call a function* $A : C_1 \rightarrow C_2$ *an* abstraction function, *if* A *is additive and surjective and fulfils the correctness condition* $\forall n \in \mathbf{N}.\ [\![n]\!]_1; A \sqsubseteq A; [\![n]\!]_2$. *In this case we write* $\mathcal{S}_2 \leq_A \mathcal{S}_1$.

In order to describe the mutual relationship between these levels we additionally use the notion of an adjoint (*or* concretization) *function* A^a *defined by:*

$\forall c \in C_2.\ A^a(c) =_{df} \bigsqcup\{ c' \mid A(c') = c\}$

Remark 2.4

1. The additivity requirement of the semantic functionals and the abstraction functions is an important restriction that is responsible for most of the properties we obtain. The surjectivity requirement is essentially technical, present only to simplify the development of the theory.
2. Pairs of the form (A, A^a) are *pairs of adjoint functions* in the sense of Cousot and Cousot [4]. Notice that A^a is monotonic, but in general not additive.

The abstraction function A_{Be} between $\mathcal{S}_B$ and $\mathcal{S}_e$ is given by

$A_{Be}(\pi) = min\{o(t) \mid t \in \pi\}$

in our example. Thus ${A_{Be}}^a(n)$ consists of all terms t with $o(t) \geq n$ (We will proceed using this notation, which specifies source and target of the abstraction function by its index expression.) It is easy to see that A_{Be} is both additive and surjective, and that it fulfils the correctness requirement. Finally let us introduce the notion of isomorphism:

Definition 2.5 *Two abstract semantics* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are called* isomorphic, *if there exists an additive and bijective abstraction function* $A : C_1 \rightarrow C_2$ *such that* $[\![G]\!]_1; A = A; [\![G]\!]_2$ *for all* $G \in \mathbf{FG}$. *In this case we write* $\mathcal{S}_1 \cong_A \mathcal{S}_2$ *or just* $\mathcal{S}_1 \cong \mathcal{S}_2$.

This notion of isomorphism exactly meets the usual intuition. In particular $\mathcal{S}_1 \cong_A \mathcal{S}_2$ implies that A is bijective and that $\mathcal{S}_2 \cong_{A^{-1}} \mathcal{S}_1$ holds.

3 Observational Equivalence

This section is based on the fact that, given an abstract semantics $\mathcal{S} = ([\![\ ]\!], C)$, a complete semi-lattice Ω ("Ω" stands for "observation") and an additive and surjective function $A : C \to \Omega$, $\mathcal{S}$ induces a semantic functional (or a *behaviour*) $[\![\ ]\!]_A : \mathbf{FG} \to (\Omega \to \Omega)$ on Ω by:

$$\forall G \in \mathbf{FG}.\ [\![G]\!]_A =_{df} A^a; [\![G]\!]; A.$$

In the following we abbreviate this situation by $\mathcal{S} \rightarrow_A \Omega$.

Example: Let us now assume that it is enough to consider only the constant term and the first order term to obtain reasonable results and that we are only interested in whether the variable x contains a significant result at the end of the program or not, i.e. whether $o(x) < 2$ or not. (Think perhaps of the situation where x contains a value of disturbance after a special amount of time. In this situation this observation level expresses whether the disturbance will be definitely damped after a while or not.) This gives the observation level $C_O =_{df} \{\bot, \top\}$ with $\bot \sqsubseteq \top$ and the abstraction functions A_{BO} and A_{eO} that indicate the information "definitely insignificant" (i.e. each possible value has order ≥ 2) by $\bot$ and the complement ("possibly significant") by $\top$.

Data flow analysis is concerned with the problem of *efficiently* answering questions about program behaviour as *precisely* as possible. The complete program semantics and the behaviour of a data flow analysis algorithm can be formulated as *abstract semantics*. And it is also possible to characterize the aspects of interest of (or questions about) the program behaviour using a complete (semi-)lattice (the question level). Thus it turns out that we have to search for "minimal" abstract semantics which *simulate* the complete program semantics wrt the observation level, i.e. for "minimal" representatives of the following equivalence relation $\approx_\Omega$:

Definition 3.1 *Let Ω be an observation level. Then we define:*

1. *Given an abstract semantics $\mathcal{S}$ satisfying $\mathcal{S} \rightarrow_A \Omega$, we call the pair $(\mathcal{S}, A)$ a* model *of Ω.*

2. *Two models $(\mathcal{S}, A)$ and $(\mathcal{S}', A')$ of Ω are called* $\Omega-$equivalent *or* observationally equivalent *wrt Ω, $(\mathcal{S}, A) \approx_\Omega (\mathcal{S}', A')$, iff they induce the same behaviour on Ω, i.e. iff $[\![\ ]\!]_A = [\![\ ]\!]_{A'}$.*

Remark 3.2 This approach is "observation directed". It does not consider a standard semantics, it only deals with the behaviour induced on a certain observation level. Thus the following development differs in its "orientation" from the considerations in [2, 3, 4] because the approach of Cousot / Cousot, the *data flow analysis framework*, is "semantics directed" (i.e. directed towards a standard semantics). In particular, Cousot / Cousot do not consider a seperate observation level.

Obviously we have:

Lemma 3.3 $\approx_\Omega$ *is an equivalence relation on the set of all models of* Ω.

Let us now illustrate the nature of the generalization that we obtain by replacing *optimality* (or *completeness*) wrt the three level model by *observational equivalence.* The three level model deals only with hierarchical situations like:

Diagram 3.4

$$\begin{tikzcd}
\mathcal{S}_1 : & C_1 \arrow[r, "{[\![\ ]\!]_1}"] \arrow[d, "A_1"'] & C_1 \arrow[d] & (\textit{basic level}) \\
\mathcal{S}_2 : & C_2 \arrow[r, "{[\![\ ]\!]_2}"] \arrow[d, "A_2"'] & C_2 \arrow[d] & (\textit{computation level}) \\
 & \Omega & \Omega & (\textit{observation or question level})
\end{tikzcd}$$

Consequently, only *abstractions* of a given *basic level* are considered as *computation levels.* In contrast, the more general notion of *observational equivalence* covers the following situation as well:

Diagram 3.5

$$\begin{tikzcd}
\mathcal{S} : & C \arrow[r, "{[\![\ ]\!]}"] \arrow[dr, "A"'] & C \arrow[dr] & C' \arrow[r, "{[\![\ ]\!]'}"] \arrow[dl] & C' \arrow[dl, "A'"] & : \mathcal{S}' & (\textit{inducing level}) \\
 & & \Omega & \Omega & & & (\textit{observation level})
\end{tikzcd}$$

Example: Let us consider $\mathcal{S}=_{df}\mathcal{S}_e$, $\mathcal{S}'=_{df}\mathcal{S}_b$, $\Omega=_{df}\mathcal{C}_O$, $A=_{df}A_{eO}$, $A'=_{df}A_{bO}$, where the definition of A_{bO} (and also the definitions of A_{Bb}, A_{Bfa}, A_{efa}, A_{bfa} and A_{faO}, which we will use later on in our example) is straightforward, and where $\mathcal{S}_b=_{df}([\![\;]\!]_b, \mathcal{C}_b)$ is defined by: $\mathcal{C}_b=_{df}\mathcal{P}(\{t \mid t\in\Pi_0 \wedge o(t)\leq 2\})$ with $\sqsubseteq\,=\,\subseteq$ (the index "b" stands for "bound") and $[\![\, x := t\,]\!]_b(\pi)=\{t[t_0/x] \mid t_0\in\pi \wedge o(t[t_0/x])\leq 2\}$, for all $(x := t)\in \mathbf{N}$ and $\pi\in\mathcal{C}_b$. Then $(\mathcal{S},A)$ and $(\mathcal{S}',A')$ are two observationally equivalent models of Ω (see **Definition 3.1(2)**) which cannot be "ordered" by means of an abstraction function. Thus, they are incomparable within the three level model, but equivalent in the general model illustrated by **Diagram 3.5**.

Nevertheless, the main result of this paper shows that the *three level model* is general in a theoretical sense: the ideal computation level (which is given by the fully abstract model) can be determined within the three level hierarchical model. This is because there exists a most abstract semantics, the *fully abstract model*, that is observationally equivalent to a certain basic level. (Indeed, the notion of a fully abstract model in [18], which is defined wrt the three level hierarchical model, characterizes the same family of abstract semantics as the more general notion introduced here. The difference consists only of the necessity to specify the abstraction context here. This was unnecessary in [18] with its fixed standard situation.) We will now proceed by developing this general result.

3.1 Fully Abstract Models

In order to avoid nonstandard models, we have to reduce the domain of an abstract semantics to its *reachable part* which is introduced as "$\mathcal{RL}$(..)" in **Definition 3.6(1)**. This technical reduction is essential to obtain the uniqueness of fully abstract models.

Definition 3.6 *Let $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ and $\mathcal{S}_2 \rightarrow_{A_2} \Omega$. Then we introduce:*

1. $\mathcal{RI}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega)=_{df}\{\, A_2{}^a; A_1{}^a; [\![\, G\,]\!]_1; A_1(c) \mid G\in\mathbf{FG} \wedge c\in\Omega\,\}$

2. $\mathcal{RL}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega)$ *denotes the smallest complete sub-(semi-)lattice of* $\mathcal{C}_2$ *that contains* $\mathcal{RI}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega)$.

3. $\mathcal{RS}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega)=_{df}([\![\;]\!],\ \mathcal{RL}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega))$, *where* $[\![\;]\!]$ *is defined as follows:*

   $\forall G\in\mathbf{FG}$.

   $$[\![\, G\,]\!] =_{df} \begin{cases} [\![\, G\,]\!]_2 \mid \mathcal{RL}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega) & \text{if } \mathcal{RL}(\mathcal{S}_1,A_1,\mathcal{S}_2,A_2,\Omega) \text{ is closed under } [\![\;]\!]_2 \\ \perp & \text{otherwise} \end{cases}$$

4. $\mathcal{S}_2$ *is called* locally optimal *wrt* $\mathcal{S}_1$ *and* A_1, *iff* $A_1{}^a; [\![\, n\,]\!]_1; A_1 = [\![\, n\,]\!]_2$ *for all* $n\in\mathbf{N}$.

The following notion of a fully abstract model generalizes the notion introduced in [18] by replacing "abstract semantics" by "model of Ω". Thus fully abstract models are no longer abstract semantics but models of Ω. — Originally, the notion of *full abstractness* was introduced by Milner [8] in the context of the λ-calculus.

Definition 3.7 *Let $\mathcal{S}_1$, Ω and A fulfil $\mathcal{S}_1 \rightarrow_A \Omega$. A pair $(\mathcal{S}_2, A_2)$ with $\mathcal{S}_2 \rightarrow_{A_2} \Omega$ is called a* fully abstract model *for $\mathcal{S}_1$ wrt Ω and A, iff an abstraction function A_1 with $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ exists, satisfying the following four properties:*

1. $A = A_1; A_2$
2. $\mathcal{S}_2 = RS(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$
3. *$\mathcal{S}_2$ is locally optimal wrt $\mathcal{S}_1$ and A_1*
4. $\forall\, c, c' \in RI(\mathcal{S}_1, ID, \mathcal{S}_1, A, \Omega)$.

   $$A_1(c) = A_1(c') \iff \forall\, G \in \mathbf{LFG}.\ [\![G]\!]_1; A(c) = [\![G]\!]_1; A(c')$$

   where ID is the identity on the semantic domain of $\mathcal{S}_1$.

We denote the set of all those fully abstract models $(\mathcal{S}_2, A_2)$ *by* $\Phi(\mathcal{S}_1, A, \Omega)$.

Example:
Concerning the order analysis, $\mathcal{S}_{fa} =_{df} ([\![\ ]\!]_{fa}, C_{fa})$, defined by $C_{fa} =_{df} \{0,1,2\}$ with $\sqsubseteq = \geq$ and $[\![x := t]\!]_{fa}(n) = min\,\{\,2,\ o(t[x_0{}^n/x])\,\}$ for all $(x := t) \in \mathrm{N}$ and $n \in \{0,1,2\}$, is a fully abstract model of $\mathcal{S}_B$ wrt C_O and A_{BO} (see section **3.3**). (Here, "fa" stands for "fully abstract".)

The following two results are central:

Theorem 3.8 (Existence and Uniqueness Theorem)

Let $\mathcal{S}_1$, Ω and A satisfy $\mathcal{S}_1 \rightarrow_A \Omega$. Then there exists a fully abstract model $(\mathcal{S}_2, A_2)$ for $\mathcal{S}_1$ wrt Ω and A, and we have the following uniqueness result:

$$\Phi(\mathcal{S}_1, A, \Omega) = \{\, (\mathcal{S}'_2, A'_2) \mid \exists A'.\ \mathcal{S}_2 \cong_{A'} \mathcal{S}'_2 \wedge A_2 = A'; A'_2 \,\}.$$

The second result, Theorem 3.9, gives us the opportunity to cut off the abstraction process at the level of a fully abstract model without any loss wrt the equivalence properties we are interested in. This is important, because it allows easy proofs by induction.

Theorem 3.9 *Let* $(\mathcal{S}_2, A_2) \in \Phi(\mathcal{S}_1, A_1; A_2, \Omega)$ *with* $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$. *Then we have:*

$$\forall G \in \mathbf{FG}.\ A_1{}^a; [\![G]\!]_1; A_1 = [\![G]\!]_2$$

In particular: $(\mathcal{S}_1, A_1; A_2) \approx_\Omega (\mathcal{S}_2, A_2)$

3.2 The Product

Our main result, the **Equivalence Theorem 3.13**, requires the construction of a connecting isomorphism between two arbitrarily given observationally equivalent fully abstract models. This isomorphism can be elegantly constructed by means of the following product construction:

Definition 3.10 *The* product $(\mathcal{S},A)\times(\mathcal{S}',A')$ *of two models* $(\mathcal{S},A)$ *and* $(\mathcal{S}',A')$ *of* Ω *is defined by* $(\mathcal{S},A)\times(\mathcal{S}',A')=_{df}([\![\ ]\!]_p, C_p)$, *where* C_p *is the smallest complete semi-lattice contained in the cartesian product* $C\times C'$ *of the semantic domains of* $\mathcal{S}$ *and* $\mathcal{S}'$ *that includes*

$$(*)\quad \{\,(A^a;[\![G]\!](c),\ A'^a;[\![G]\!]'(c))\mid G\in\mathbf{FG}\wedge c\in\Omega\,\}$$

and $[\![\ ]\!]_p$ *is the semantic functional characterized by componentwise application, i.e.:*

$$\forall n\in\mathbf{N}\ \forall(c,c')\in C.\ [\![n]\!](c,c')=_{df}([\![n]\!](c),\ [\![n]\!]'(c')).$$

The point of this definition is the choice of the semantic domain C_p. Let us illustrate the effect of this choice by means of the order analysis.

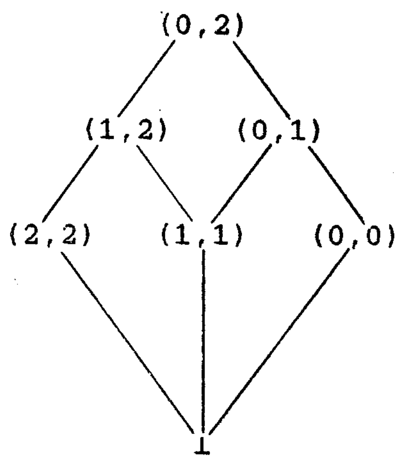
Example: $(*)$ restricts the domain of a product to consistent pairs, i.e. pairs, where the two components can be generated by means of a common "history". In particular, constructing the product of one model with itself essentially delivers the relevant part of this model. For example:

$$(\mathcal{S}_{fa},A_{faO})\times(\mathcal{S}_{fa},A_{faO})\ \cong\ \mathit{RS}(\mathcal{S}_{fa},\mathit{ID},\mathcal{S}_{fa},A_{faO},C_O)\ =\ (\mathcal{S}_{fa},A_{faO})$$

On the other hand, if we consider a programming language with two variables x and y, and if we construct the product of the abstract semantics $(\mathcal{S},A)$ concerning x and the abstract semantics $(\mathcal{S}',A')$ concerning y, then these two argument semantics are completely independent. Consequently, the "product domain" is the full cartesian product $C\times C'$.

Finally let us be more sophisticated. Suppose that we are not only interested in the question of whether an expression *must* be insignificant (perhaps to decide, whether an expression will be definitely damped after a while), but also in the question of whether it *might* be insignificant (perhaps to discover the perturbations that are definitely stable in time). To deal with both of these questions in parallel, we must determine the whole range of the order of an expression restricted to the set $\{0,1,2\}$. We therefore define the *dual* model $(\mathcal{S}^{fa},A^{faO})=_{df}((C^{fa},[\![\ ]\!]^{faO}),A^{faO})$ of $(\mathcal{S}_{fa},A_{faO})$ by: $C^{fa}=_{df}\{0,1,2\}$ with $\sqsubseteq=\leq$ and $[\![\ ]\!]^{fa}=_{df}[\![\ ]\!]_{fa}$ (i.e. just by "dualizing" the semantic domain). In this case, the semantic domain of $(\mathcal{S}_{fa},A_{faO})\times(\mathcal{S}^{fa},A^{faO})$ is given by:

Diagram 3.11



"every value is possible: no information"

"inconsistency: marks unreachable code"

(Here, the first and second component represent the values wrt $\mathcal{S}_{fa}$ and $\mathcal{S}^{fa}$ respectively.) The consistency requirement $(*)$ is responsible for the loss of $(2,1)$, $(1,0)$ and $(2,0)$ and the completeness requirement for the new artificial element $\perp$. Intuitively you can regard $\perp$ as the collapse of the three inconsistent informations mentioned. In practice, $\perp$ serves as the initial information of the iterative data flow analysis process for computing the optimal range information.

The following (immediate) properties of our product construction are essential for the proof of the **Equivalence Theorem 3.13:**

Lemma 3.12

Let $(\mathcal{S},A)$ and $(\mathcal{S}',A')$ be models of Ω and $\mathcal{S}''=_{df}(\mathcal{S},A)\times(\mathcal{S}',A')$. Then we have:

(1) *$\mathcal{S}''$ is an abstract semantics.*

Also, let $S=RS(\mathcal{S},ID,\mathcal{S},A,\Omega)$, $S'=RS(\mathcal{S}',ID,\mathcal{S}',A',\Omega)$ and pr resp. pr′ be the projection functions of C to its first resp. second component. Then we have:

(2) $S \leq_{pr} (\mathcal{S},A)\times(\mathcal{S}',A')$ *and* $S' \leq_{pr'} (\mathcal{S},A)\times(\mathcal{S}',A')$

(3) $(\mathcal{S},A)\times(\mathcal{S}',A')\rightarrow_{pr;A}\Omega$ *and* $(\mathcal{S},A)\times(\mathcal{S}',A')\rightarrow_{pr';A'}\Omega$

(4) *S resp. S' are locally optimal wrt $(\mathcal{S},A)\times(\mathcal{S}',A')$ and pr resp. pr′*

3.3 The Equivalence Theorem

The main result of this paper (**Equivalence Theorem 3.13**) generalizes the *Equivalence Theorem* of [18], which only deals with the three level hierarchical model (see **Diagram 3.4**).

Theorem 3.13 (Equivalence Theorem)

Let $(\mathcal{S}, A)$ and $(\mathcal{S}', A')$ be two models of Ω. Then we have:

$$(\mathcal{S}, A) \approx_{\Omega} (\mathcal{S}', A') \iff \Phi(\mathcal{S}, A, \Omega) = \Phi(\mathcal{S}', A', \Omega).$$

According to the **Existence and Uniqueness Theorem 3.8** *and* **Theorem 3.9** *we obtain that fully abstract models are (up to isomorphism) the most abstract models of their observational equivalence class.*

Sketch of the Proof:

Whereas the implication "$\Leftarrow$" is rather straightforward, the implication "$\Rightarrow$" is complicated. The problem here is to "synchronize" elements $(\hat{\mathcal{S}}, \hat{A}) \in \Phi(\mathcal{S}, A, \Omega)$ and $(\hat{\mathcal{S}}', \hat{A}') \in \Phi(\mathcal{S}', A', \Omega)$ in such a way that we are able to construct the connecting isomorphism. Fortunately the product defined in the last section exactly delivers this kind of "synchronization". In fact, using this product construction, the whole proof can be sketched in four steps by means of the following diagram:

Diagram 3.14

$$\begin{tikzcd}
 & \mathcal{S} \arrow[ddr] & & & & \mathcal{S}' \arrow[ddl] & \\
 & & & \hat{\mathcal{S}} \times \hat{\mathcal{S}}' \arrow[dl, "pr"'] \arrow[dr, "pr'"] & & & \\
A & & \hat{\mathcal{S}} \arrow[dr, "\hat{A}"'] & & \hat{\mathcal{S}}' \arrow[dl, "\hat{A}'"] & & A' \\
 & & & \Omega & & &
\end{tikzcd}$$

First, we reduce the statement of the theorem to the case where the models concerned are fully abstract themselves. That allows us to use the parts (2), (3), and (4) of **Lemma 3.12**. Second, we consider the product of the two fully abstract models mentioned in the first part and show that the projection functions pr and

pr' are bijective. That can be done using **Theorem 3.9** twice. Third, we consider the function $pr^{-1}; pr'$ (which is bijective due to the second step) and show that it is indeed an isomorphism between the fully abstract models $(\hat{\mathcal{S}}, \hat{A})$ and $(\hat{\mathcal{S}}', \hat{A}')$. Finally, we show $\hat{A} = (pr^{-1}; pr'); \hat{A}'$ and apply the **Existence and Uniqueness Theorem 3.8** to finish the proof. Details can be found in [17].

Example: The complete setup of our example can be summarized as follows:

$$\begin{tikzcd}
& \mathcal{S}_B \arrow[dl, "A_{Bb}"'] \arrow[dr, "A_{Be}"] & \\
\mathcal{S}_b \arrow[dr, "A_{bfa}"'] & & \mathcal{S}_e \arrow[dl, "A_{efa}"] \\
& \mathcal{S}_{fa} \arrow[d, "A_{faO}"] & \\
& \mathcal{C}_O &
\end{tikzcd}$$

Diagram 3.15

We conclude by arguing that:

1. $(\mathcal{S}_B, A_{BO})$, $(\mathcal{S}_b, A_{bO})$, $(\mathcal{S}_e, A_{eO})$ and $(\mathcal{S}_{fa}, A_{faO})$ are indeed all members of the same $\mathcal{C}_O$–equivalence class $\mathcal{K}$, and

2. $(\mathcal{S}_{fa}, A_{faO})$ is the unique fully abstract model of $\mathcal{K}$.

Here the following theorem is essential:

Theorem 3.16 *Let* $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ *and* $\mathcal{S}_2 \not\rightarrow_{A_2} \Omega$. *Then we have:*

$$\forall n \in \omega.\ A_1; [\![n]\!]_2 = [\![n]\!]_1; A_1 \Rightarrow (\mathcal{S}_2, A_2) \approx_\Omega (\mathcal{S}_1, A_1; A_2)$$

This theorem is a consequence of the Simulation Theorem [18]. However, a direct proof by induction on the size of a certain argument program is straightforward as well.

(1) directly follows by means of **Theorem 3.16**. Thus, according to the **Equivalence Theorem 3.13**, it remains to show that $(\mathcal{S}_{fa}, A_{faO})$ is the most abstract member of $\mathcal{K}$. To prove this, let us assume that there exists a more abstract model $(\mathcal{S}_{ce}, A_{ce}) \in \mathcal{K}$ ("ce" stands for "counter example"). Furthermore realize that a more abstract computation level than C_{fa} must have domain C_O (up to isomorphism), and that $A_{face}(2)$ must be $\perp$ for correctness reasons. This is already enough to yield an "observable difference" between $(\mathcal{S}_{ce}, A_{ceO})$ and $(\mathcal{S}_{fa}, A_{faO})$ and therefore a contradiction: the correctness condition (see **Definition 2.3**) delivers:

$$\begin{array}{rcl} [\![\, x := x^2 \,]\!]_{ce}(\top) & = & [\![\, x := x^2 \,]\!]_{ce}(A_{faO}(1)) \\ & \sqsubseteq & A_{faO}([\![\, x := x^2 \,]\!]_{faO}(1)) \\ & = & A_{faO}(2) \\ & = & \perp \end{array}$$

Thus $[\![\, x := x^2 \,]\!]_{ce}([\![\, x := 0 \,]\!]_{ce}(\top)) = \perp$. On the other hand:

$$\begin{array}{rcl} A_{faO}([\![\, x := x^2 \,]\!]_{fa}([\![\, x := 0 \,]\!]_{fa}(A_{faO}{}^{a}(\top)))) & = & A_{faO}([\![\, x := x^2 \,]\!]_{fa}(0)) \\ & = & A_{faO}(0) \\ & = & \top \end{array}$$

4 Applications and Future Work

The three level model presented in [18] was the basis for the construction of the analysis algorithm of a *run time optimizer* which has been implemented in a multitarget, multilanguage compiler system. Indeed, using the three level model it could be shown that the analysis algorithm computes *complete* information about arbitrary flow graphs: it detects whenever two terms are *transparently equivalent* [15], i.e. it detects all equivalences between program terms which do not depend on specific properties of certain term operators [17, 18].

In contrast to the three level model itself, the generalization presented in this paper is powerful enough to deal even with the specific *representation* of the *abstract domain* data flow analysis algorithms operate on, e.g. it allows a uniform completeness (or optimality) proof of the concrete implementation of our analysis algorithm, a problem which was tackled separately in the original proof. This property stresses the power of our *observation directed* approach. It covers not only *abstractions*, but also *concretizations* or *data refinements* [6] in a very general way.

Although our framework of observational equivalence for abstract interpretations is conceived for imperative languages, its basic idea applies to functional, logical and parallel languages as well. In fact, it is possible to develop a similar, language independent setup in the categorical framework (cf. [20]).

Abstract interpretation is already introduced and successfully used for functional languages, for example, for strictness analysis [1, 9, 10, 13]. More general is the two level semantics approach of Nielson/Nielson [11, 13, 14] which widens the range of data flow analysis applications by introducing an explicit distinction between run time types and compile time types. It has to be checked to what extent our ideas apply to these topics.

5 Acknowledgements

I owe very much to Flemming Nielson, Michael Mendler, Andrzej Tarlecki, Don Sannella, Colin Stirling, Jens Knoop, Jordi Farrés, Karla Börnig and Bob Harper for helpful comments about and discussions of the ideas which led to this paper.

References

[1] G. L. Burn, C. L. Hankin, and S. Abramsky. The theory of strictness analysis for higher order functions. *Science of Computer Programming*, 7:249–278, 1986.

[2] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, 1977.

[3] P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: Mathematical foundations. *ACM Sigplan Notices*, 12:1–12, 1977.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, 1979.

[5] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[6] C. A. R. Hoare and H. Jifeng. Data refinement in a categorical setting. Technical report, Oxford University, Computing Laboratory, Programming Research Group, February 1988.

[7] J. B. Kam and J. D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:309–317, 1975.

[8] R. Milner. Fully abstract models of typed lambda calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[9] A. Mycroft. *Abstract Interpretation and Optimizing Transformations for Applicative Programs.* PhD thesis, Edinburgh Univ., Dept. of Comp. Sci., 1981.

[10] A. Mycroft and F. Nielson. Strong abstract interpretation using power domains. In *ICALP '83*, pages 536–547. LNCS 154, 1983.

[11] F. Nielson. Abstract interpretation of denotational definitions. In *STACS '86*, pages 1–20. LNCS 210, 1986.

[12] F. Nielson. A bibliography on abstract interpretations. *ACM Sigplan Notices*, 21:31–38, 1986.

[13] F. Nielson. Strictness analysis and denotational abstract interpretation. In *14th POPL*, pages 120–131, Munich, West-Germany, 1987.

[14] H. R. Nielson and F. Nielson. Pragmatic aspects of two-level denotational meta-languages. In *ESOP '86*, pages 133–143. LNCS 213, 1986.

[15] B. K. Rosen, M. N. Wegmann, and F. K. Zadeck. Global value numbers and redundant computations. In *15th POPL*, pages 12–27, San Diego, California, 1988.

[16] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specifications. *Journal of Computer and System Sciences*, pages 150–178, 1987.

[17] B. Steffen. *Abstrakte Interpretationen beim Optimieren von Programmlaufzeiten. Ein Optimalitätskonzept und seine Anwendung.* PhD thesis, Christian-Albrechts-Universität Kiel, 1987.

[18] B. Steffen. Optimal run time optimization - proved by a new look at abstract interpretations. In *TAPSOFT '87*, pages 52–68. LNCS 249, 1987.

[19] B. Steffen. Optimal data flow analysis via observational equivalence. In *MFCS '89*, Lecture Notes in Computer Science. Springer Verlag, 1989.

[20] B. Steffen and M. Mendler. Compositional characterization of observable program properties. LFCS report series, LFCS, Edinburgh Univ., Dept. of Comp. Sci., 1989.