The Interprocedural Coincidence Theorem

Jens Knoop[1]
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Preußerstr. 1 - 9
D–2300 Kiel 1

Bernhard Steffen
Lehrstuhl für Informatik II
Rheinisch-Westfälische Technische Hochschule Aachen
Ahornstr. 55
D–5100 Aachen

Aachener Informatikberichte
Nr. 91 – 27

Abstract

We present an interprocedural generalization of the well-known (intraprocedural) Coincidence Theorem of Kam and Ullman, which provides a sufficient condition for the equivalence of the *meet over all paths* (*MOP*) solution and the *maximal fixed point* (*MFP*) solution to a data flow analysis problem. This generalization covers arbitrary imperative programs with recursive procedures, formal parameters and local variables, and, in the absence of procedures, reduces to the classical intraprocedural version. In particular, our stack-based approach generalizes the coincidence theorems of Barth and Sharir/Pnueli for the same setup, which do not properly deal with local variables of recursive procedures.

Keywords: abstract interpretation; data flow analysis; (interprocedural) coincidence theorem; (interprocedural) meet over all paths solution; (interprocedural) maximal fixed point solution; program analysis and optimization.

[1]The author is supported by the DFG grant La 426/9-2

Contents

1 Motivation

Data flow analysis is a classical tool for the static analysis of programs that supports the generation of efficient object code by "optimizing" compilers (cf. [He,MJ]). For imperative languages, it provides information about the program states that may occur at some given program points during execution.

Theoretically well-founded are data flow analyses that are based on ***abstract interpretation*** (cf. [CC1]). The point of this approach is to replace the "full" semantics by a simpler more abstract version, which is tailored to deal with a specific problem. Usually, the abstract semantics is specified by a local semantic functional, which gives abstract meaning to every program statement in terms of a transformation function from a lattice $\mathcal{C}$ into itself. The elements of $\mathcal{C}$ express the data flow information of interest. The (global) abstract semantics then results from one of the following two globalization strategies: the "operational" ***meet over all paths*** (*MOP*) strategy, and the "denotational" ***maximal fixed point*** (*MFP*) strategy[2] in the sense of Kam and Ullman [KU][3].

The *MOP*-strategy directly mimics possible program executions: it "meets" (intersects) all information corresponding to program paths reaching the program point under consideration. This specifies the optimal result of a globalization but is in general not effective.

The *MFP*-strategy iteratively approximates the greatest solution of a set of simultaneous equations that express consistency between pre-conditions and post-conditions that are given in terms of data flow information: the pre-condition of a statement must be implied by each of the post-conditions of the predecessors, and the post-condition must be implied by the result of transforming the pre-condition according to the (abstract) meaning of the statement. In general, this leads to a suboptimal but algorithmic description.

The (intraprocedural) Coincidence Theorem of [KU] states the coincidence of the *MOP*-solution and the *MFP*-solution in the case of ***distributive*** (local) semantic functionals (see Section 2). In this paper, we present an ***interprocedural*** generalization of this theorem (cf. Interprocedural Coincidence Theorem 7.2), which covers arbitrary imperative programs with recursive procedures, formal parameters and local variables, and, in the absence of procedures, reduces to the classical intraprocedural version of Kam and Ullman. The point of our generalization is the introduction of stacks of lattice elements as data flow information, which is necessary to properly deal with local variables of recursive procedures. These stacks directly mimic the run-time stack used by run-time systems for maintaining the activation records of different procedure incarnations. Whereas the operational (*MOP*) strategy exhaustively makes use of the stack structure, it turns out that the denotational (*MFP*) strategy only needs stacks of length at most two (Remark 4.7).

[2] These are the strategies that lead to the *MOP*-solution and *MFP*-solution, respectively.

[3] The operational and denotational flavour of these two strategies becomes particularly apparent in the interprocedural setting.

Related Work

Semantically based reasoning about interprocedural data flow analysis was first considered by Cousot/Cousot [CC2], Rosen [Ro], and Jones and Muchnick [JM].

Rosen [Ro] proved the correctness of an interprocedural data flow analysis algorithm computing a maximal fixed point solution that provides information whether a variable is used, modified or preserved. Thus his algorithm is tailored to specific problems, and it is not clear how to generalize his approach to arbitrary abstract interpretations. Such a general situation was investigated in [CC2] and [JM], leading to highly technical definitions that are difficult to apply. Moreover, as in [Ro], these two papers only address the correctness aspect but no optimality.

Both correctness and optimality were considered by Barth [Ba1,Ba2] and Sharir/-Pnueli [SP], who independently proposed an interprocedural version of the Coincidence Theorem of Kam and Ullman. However, their approaches do not properly deal with local variables of recursive procedures.

Structure of the Paper

After sketching the intraprocedural setting in Section 2, we recall the formal framework for interprocedural data flow analysis in Section 3, and define two versions of abstract semantics for programs with procedures in Section 4: an "operational" one caused by the ***interprocedural meet over all paths*** strategy and a "denotational" one caused by the ***interprocedural maximal fixed point*** strategy. Subsequently, Section 5 presents the data flow analysis algorithms that compute the interprocedural maximal fixed point solution, Section 6 the Main Lemma 6.4, and Section 7 our main results, the Interprocedural Safety Theorem 7.1 and the Interprocedural Coincidence Theorem 7.2. Finally, Section 8 sketches three applications of the Interprocedural Coincidence Theorem 7.2, and Section 9 contains our conclusions.

2 The Intraprocedural Setting

In this section, we summarize the intraprocedural setting for data flow analysis, which is characterized by a separate and independent investigation of the procedures of a program. Here it is common to represent procedures as ***directed flow graphs***[4] $G = (N, E, \mathbf{s}, \mathbf{e})$ with node set N and edge set E. Nodes $n \in N$ represent the statements and edges $(n, m) \in E$ the nondeterministic branching structure of the corresponding procedure. $pred_G(n) =_{df} \{ m \mid (m, n) \in E \}$ denotes the set of all predecessors and $succ_G(n) =_{df} \{ m \mid (n, m) \in E \}$ the set of all successors of a node n, and $\mathbf{s}$ and $\mathbf{e}$ the unique ***start node*** and ***end node*** of G, which are assumed to possess no predecessors and successors, respectively. A ***finite path*** in G is a sequence $(m \equiv n_1, \ldots, n_q \equiv n)$ of nodes such that $(n_j, n_{j+1}) \in E$ for $j \in \{1, ., q-1\}$. $\mathbf{P}[m, n]$ denotes the set of all finite paths from m to n, and $\mathbf{P}[m, n)$ the set of all finite paths from m to a predecessor of n. Moreover, ";" denotes the concatenation of two paths, $lgth(p)$ the number of node occurrences in p, and ε the unique path of length 0. Finally, we assume that every node $n \in N$ lies on a path from $\mathbf{s}$ to $\mathbf{e}$.

[4] The construction of flow graphs is described in [All].

Given a complete semi-lattice $(\mathcal{C}, \sqcap, \sqsubseteq, \perp, \top)$, whose elements are intended to express the relevant data flow information, the ***local*** abstract semantics of a flow graph G is given by a semantic functional

$$[\![\ ]\!] : N \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$$

which gives meaning to every node $n \in N$ in terms of a transformation on $\mathcal{C}$. For simplicity, it is assumed that $\mathbf{s}$ and $\mathbf{e}$ are associated with the identity on $\mathcal{C}$, $id_{\mathcal{C}}$.

The local abstract semantics $[\![\ ]\!]$ can easily be extended to cover finite paths as well: for every path $p = (n_1, ..., n_q) \in \mathbf{P}[m, n]$, we define:

$$[\![p]\!] =_{df} \begin{cases} id_{\mathcal{C}} & \text{if } p \equiv \varepsilon \\ [\![(n_2, ..., n_q)]\!] \circ [\![n_1]\!] & \text{otherwise} \end{cases}$$

The (global) abstract semantics then results from one of the following two globalization strategies: the "operational" ***meet over all paths*** (*MOP*) strategy, and the "denotational" ***maximal fixed point*** (*MFP*) strategy in the sense of Kam and Ullman [KU].

The *MOP*-strategy directly mimics possible program executions: it "meets" (intersects) all informations, which belong to a program path reaching the program point under consideration:

The *MOP*-Solution: $\forall\, n \in N\ \forall\, c_0 \in \mathcal{C}.\ MOP_{c_0}(n) = \bigsqcap \{\, [\![p]\!](c_0) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\}$

This directly reflects our desires but is in general not effective.

The *MFP*-strategy iteratively approximates the greatest solution of a set of simultaneous equations that express consistency between pre-conditions and post-conditions that are expressed in terms of $\mathcal{C}$:

Equation System 2.1

$$\mathbf{pre}(n) \quad = \quad \begin{cases} c_0 & \text{if } n = \mathbf{s} \\ \bigsqcap \{\, \mathbf{post}(m) \mid m \in pred_G(n) \,\} & \text{otherwise} \end{cases}$$

$$\mathbf{post}(n) \quad = \quad [\![n]\!](\mathbf{pre}(n))$$

Denoting the greatest solution of Equation System 2.1 with respect to the start information c_0 by $\mathbf{pre}_{c_0}$ and $\mathbf{post}_{c_0}$, the solution of the *MFP*-strategy is defined by:

The *MFP*-Solution: $\forall\, n \in N\ \forall\, c_0 \in \mathcal{C}.\ MFP_{c_0}(n) = \mathbf{pre}_{c_0}(n)$

In general, this leads to a suboptimal but algorithmic description.

Thus we have two global notions of semantics here, an operational one, which precisely mimics our intention, and a denotational one, which has an algorithmic character. In fact, we consider the *MOP*-strategy as a mean for the direct specification of data flow analysis problems, and the *MFP*-strategy as an algorithmic realization of such problems[5]. This view rises the question of ***correctness*** (safety) or even ***completeness*** (optimality) of such algorithms. For the elegant answer to these questions we need two further notions: Given a complete semi-lattice $(\mathcal{C}, \sqcap, \sqsubseteq, \perp, \top)$, a function $f : \mathcal{C} \rightarrow \mathcal{C}$ is called

- ***monotonic*** iff $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq c'$ implies $f(c) \sqsubseteq f(c')$
- ***distributive*** iff $\forall\, C' \subseteq \mathcal{C}.\ f(\bigsqcap C') = \bigsqcap \{ f(c) \mid c \in C' \}$.

It is well-known that distributivity is a stronger requirement than monotonicity in the following sense:

Lemma 2.2
A function $f : \mathcal{C} \rightarrow \mathcal{C}$ is monotonic iff $\forall\, C' \subseteq \mathcal{C}.\ f(\bigsqcap C') \sqsubseteq \bigsqcap \{ f(c) \mid c \in C' \}$.

As in this lemma, $\mathcal{C}$ will always denote a complete semi-lattice. We have (cf. [KU]):

Theorem 2.3 (Safety Theorem)
Given a flow graph $G = (N, E, \mathbf{s}, \mathbf{e})$, the MFP-solution is a correct *(or* safe*) approximation of the MOP-solution, i.e. $\forall\, n \in N\ \forall\, c_0 \in \mathcal{C}.\ MFP_{c_0}(n) \sqsubseteq MOP_{c_0}(n)$, if all the semantic functions $[\![n]\!]$, $n \in N$, are monotonic.*

Distributivity of the semantic functions yields completeness (optimality). This follows from the well-known intraprocedural Coincidence Theorem 2.4 of [KU]:

Theorem 2.4 (Coincidence Theorem)
Given a flow graph $G = (N, E, \mathbf{s}, \mathbf{e})$, the MFP-solution is complete *(or* optimal*) for the MOP-solution, i.e. $\forall\, n \in N\ \forall\, c_0 \in \mathcal{C}.\ MOP_{c_0}(n) = MFP_{c_0}(n)$, if all the semantic functions $[\![n]\!]$, $n \in N$, are distributive.*

3 Interprocedural Notions

In the interprocedural setting, we represent programs as systems $\{\pi_0, \pi_1, ., \pi_k\}$ of (recursive) procedure definitions. π_0 is assumed to denote the ***main program*** and therefore cannot be called. The denotational (*IMFP*) approach and the operational (*IMOP*) approach require two different representations of programs Π: ***flow graph systems*** and ***interprocedural flow graphs***.

Flow Graph Systems

The denotational approach works on systems $S = \{G_0, G_1, ., G_k\}$ of ***flow graphs*** with disjoint sets of nodes N_i and edges E_i, in which every procedure π of Π (including the main program π_0) is represented as a ***directed flow graph*** $G = (N, E, \mathbf{s}, \mathbf{e})$ in the sense of Section 2. $\mathbf{N}^S =_{df} \bigcup \{ N_i \mid i \in \{0, ., k\} \}$ denotes the set of all nodes of S, $\mathbf{E}^S =_{df} \bigcup \{ E_i \mid i \in \{0, ., k\} \}$ the set of all edges of S, and $\mathbf{N}^S_C \subseteq \mathbf{N}^S$ the set of nodes representing procedure calls. Finally, we need the following functions, where $\mathcal{P}$ denotes the power set operator:

- $fg : \mathbf{N}^S \rightarrow S$ with $fg(n) =_{df} G_i$ iff $n \in N_i$,

[5] Explicit algorithms are presented in Section 5.

- *decl* : $\mathbf{N}_C^S \rightarrow S$ with *decl*$(n) =_{df} G_i$ iff n repesents a procedure call of G_i,
- *calls* : $S \rightarrow \mathcal{P}(\mathbf{N}_C^S)$ with *calls*$(G_i) =_{df} \{n \mid decl(n) = G_i\}$,
- *start* : $S \rightarrow \{\mathbf{s}_0, ., \mathbf{s}_k\}$ with *start*$(G_i) =_{df} \mathbf{s}_i$ for all $i \in \{0, ., k\}$ and
- *end* : $S \rightarrow \{\mathbf{e}_0, ., \mathbf{e}_k\}$ with *end*$(G_i) =_{df} \mathbf{e}_i$ for all $i \in \{0, ., k\}$.

Intuitively, *fg* maps every node of a flow graph system to its corresponding flow graph, *decl* every call node to the called procedure, *calls* every procedure to the set of nodes calling it, and *start* and *end* every procedure to its start node and its end node, respectively.

An illustrative flow graph system is given below:

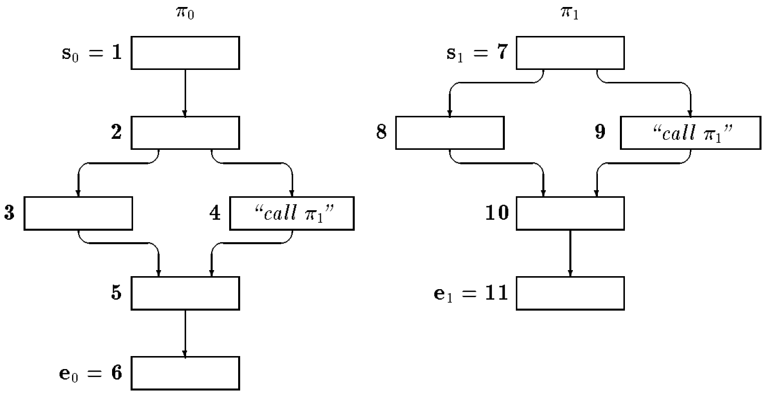

Figure 3.1

Interprocedural Flow Graphs

The operational approach requires an explicit representation of the interprocedural control flow caused by procedure calls. This is achieved by combining the flow graphs of S to an *interprocedural flow graph* $G^* = (N^*, E^*, \mathbf{s}^*, \mathbf{e}^*)$, where $\mathbf{s}^*$ is given by $\mathbf{s}_0$ and $\mathbf{e}^*$ by $\mathbf{e}_0$ (cf. [My,SP]). In detail, G^* results from S by applying the following two step procedure to every node n of $\mathbf{N}_C^S$:

Algorithm 3.2 *Let S be a flow graph system, and $n \in \mathbf{N}_C^S$. Then*

1. *Replace n by two new nodes, the* call node n_C *and the* return node n_R *such that n_C has the same set of predecessors as n but no successors and n_R has the same set of successors as n but no predecessors*[6].
2. *Draw an edge from n_C to start(decl(n)) and from end(decl(n)) to n_R.*

[6] n_C is called to *match* n_R and vice versa.

N_C^* and N_R^* denote the set of all ***call nodes*** and ***return nodes*** in N^*, respectively, and *pred*$^*(n) =_{df} \{ m \mid (m, n) \in E^* \}$ the set of all interprocedural predecessors of n. In the following, we will identify the set $\mathbf{N}^S$ of nodes of S with the set $N^* \backslash N_R^*$ of nodes of G^* to get an interpretation independent notion of program point.

Figure 3.3 shows the interprocedural flow graph that corresponds to the flow graph system of Figure 3.1:

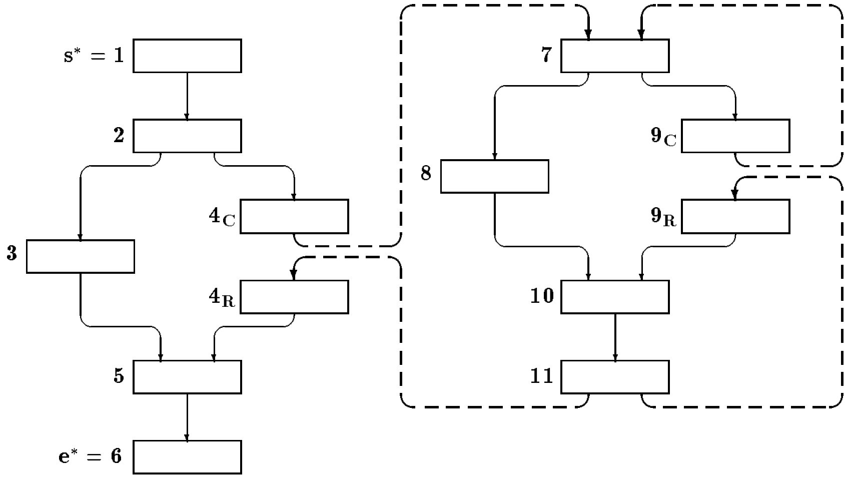

Figure 3.3

Interprocedural Paths

The notion of finite path as introduced in Section 2 naturally applies to interprocedural flow graphs as well. However, due to the special nature of procedure calls not every finite path $p \in \mathbf{P}[m, n]$ represents a valid execution: for example in Figure 3.3 the path $(2, 4_C, 7, 8, 10, 11, 4_R)$ is possible, while the path $(2, 4_C, 7, 8, 10, 11, 9_R)$ is not. This led to the following definition of interprocedural (valid) paths [SP]:

Definition 3.4 (Interprocedural Path)

1. *Let $p \in \mathbf{P}[m, n]$. Then p is an* interprocedural path *iff the tuple $(m_1, ., m_r)$, which results from p by deleting all nodes in $N^* \backslash (N_C^* \cup N_R^*)$, is well-formed in the following sense:*
   - *if there is no return node in $(m_1, ., m_r)$, then $(m_1, ., m_r)$ is well-formed,*
   - *otherwise, let j be the smallest index in $\{1,.,r\}$ such that m_j is a return node. Then $(m_1, ., m_r)$ is well-formed iff $j > 1$ and m_{j-1} is a call node matching m_j, and the remaining sequence after deleting m_{j-1} and m_j is well-formed too.*

2. *Occurrences of a call node and a return node of p are said to* correspond *to each other, if they are eliminated simultaneously in the procedure above.*

3. $\mathbf{IP}[m,n]$ *denotes the set of all* interprocedural paths *from m to n and $\mathbf{IP}[m,n)$ the set of all interprocedural paths from m to a predecessor of n.*

Complete Interprocedural Paths

In order to determine the semantics of procedure calls, we need to deal with complete interprocedural paths p from $start(fg(n))$ to n, which are characterized by the fact that all procedure calls in p have been completed by a subsequent return. This guarantees that the occurrences of $start(fg(n))$ and n belong to the same procedure incarnation.

Definition 3.5 (Complete Interprocedural Path)

1. *An interprocedural path $p = (n_1, ..., n_k) \in \mathbf{IP}[start(fg(n)), n]$ is called* complete *if it possesses equally many occurrences of procedure call and return nodes:*

$$|\ \{ i \mid n_i \in N_C^* \}\ | = |\ \{ i \mid n_i \in N_R^* \}\ |$$

2. $\mathbf{CIP}[start(fg(n)), n]$ *and* $\mathbf{CIP}[start(fg(n)), n)$ *denote the set of all* complete interprocedural paths *from $start(fg(n))$ to n, and from $start(fg(n))$ to a predecessor of n, respectively.*

That this actually realizes our intention is a consequence of the following property of interprocedural paths:

Lemma 3.6 *Let $p = (n_1, ..., n_k) \in \mathbf{IP}[m,n]$ be an interprocedural path and (n_i, n_j) and $(n_{i'}, n_{j'})$ two of its pairs of corresponding call and return nodes. Then the integer intervals $[i:j]$ and $[i':j']$ are either disjoint or one is included in the other.*

Proof. Assume that there exist two overlapping integer intervals $[i:j]$ and $[i':j']$ of a path $p \in \mathbf{IP}[m,n]$. Without loss of generality, we can assume (1) $i < i' < j < j'$ and (2) that all intervalls $[i'':j'']$ with $i' < i'' < j$ satisfy $j'' < j$. Due to $j < j'$, the elimination procedure of Definition 3.4 processes n_j before $n_{j'}$. Together with (2), this delivers that $(n_{i'}, n_j)$ is a pair of corresponding call and return nodes. This, however, contradicts the precondition that (n_i, n_j) is a corresponding pair, since every occurrence of a node of p occurs in at most one pair of corresponding call and return nodes. $\Box$

The pattern stated in Lemma 3.6 is illustrated in Figure 3.7, where $\{ (n_{C_i}, n_{R_i}) \mid i \in \{1, .., 4\} \}$ are assumed to be pairs of corresponding call and return nodes of p.

$$p = (m, ..., n_{C_1}, ..., n_{C_2}, ..., n_{R_2}, ..., n_{C_3}, ..., n_{C_4}, ..., n_{R_4}, ..., n_{R_3}, ..., n_{R_1}, ..., n) \in \mathbf{IP}[m,n]$$

Figure 3.7

The following easy to prove lemma will be important:

Lemma 3.8 *Let $\mathbf{s} \in \{\mathbf{s}_0, ..., \mathbf{s}_k\}$, $p = (n_1, ..., n_k) \in \mathbf{IP}[\mathbf{s}, n]$ and (n_i, n_j) a pair of corresponding call and return nodes. Then we have:*

$$(n_{i+1}, ..., n_{j-1}) \in \mathbf{CIP}[start(decl(n_i)), end(decl(n_i))]$$

Remark 3.9 If the underlying program Π has no procedures, the flow graph system S and the interprocedural flow graph G^* collapse to the flow graph G_0 of π_0. In this special case our framework coincides with the standard intraprocedural framework.

Conventions: Throughout the rest of this paper we assume an arbitrary but fixed program $\Pi = \{\pi_0, \pi_1, ., \pi_k\}$ with flow graph system $S = \{G_0, G_1, ., G_k\}$ and interprocedural flow graph $G^* = (N^*, E^*, \mathbf{s}^*, \mathbf{e}^*)$. Moreover, m and n, possibly indexed, are nodes of S or G^*, and for every node $n \in \mathbf{N}_C^S$, n_C and n_R denote its corresponding call node and return node in N^*, respectively.

4 Abstract Semantics

In this section, we present new *interprocedural* versions of the meet over all paths strategy and the maximal fixed point strategy. They define the (global) semantics for interprocedural flow graphs and flow graph systems, respectively. The point of this presentation is the extension of the data flow information in a way that mimics run-time stacks as used in usual run-time systems.

4.1 The Local Semantic Functional

As its intraprocedural counterpart, the interprocedural meet over all paths (*IMOP*) solution directly records all possible program executions that lead to a particular program point. However, in the presence of recursive procedures it is necessary to work on stacks of lattice elements instead of the lattice itself, in order to record the part of the history, which will become relevant after returning from (nested) procedure calls. Thus the local semantic functional has type

$$[\![\ ]\!] : N^* \to (STACK \to STACK)$$

where $STACK$ denotes the set of all non empty stacks with components of $\mathcal{C}$, which can be manipulated by means of the following operations:

- $newstack : \mathcal{C} \to STACK$
- $push : STACK \times \mathcal{C} \to STACK$
- $pop : STACK \to STACK$
- $top : STACK \to \mathcal{C}$

Intuitively, $\boldsymbol{newstack}(c)$ creates a new stack with single component c, $\boldsymbol{push}$ puts a new component on the top of the argument stack, $\boldsymbol{pop}$ removes the top component, and $\boldsymbol{top}$ delivers the content of the top component, while maintaining the argument stack[7].

$STACK$ is an abstract version of the run-time stack used by run-time systems for maintaining the activation records of different procedure incarnations. Intuitively, the top component of a stack contains the data flow information corresponding to the currently valid activation record[8], while the data flow informations of the remaining stack components correspond to activation records of preceding but not yet finished procedure calls. Thus the stacks directly reflect the nesting of procedure incarnations according to the current call sequence.

Formally, the local semantic functional $[\![\ ]\!]^*$ for this setting is defined by

$\forall\, n \in N^*\ \forall\, stk \in STACK.$

$$[\![n]\!]^*(stk) =_{df} \begin{cases} \boldsymbol{push}(\,\boldsymbol{pop}(stk),\ [\![n]\!]'(\boldsymbol{top}(stk))\,) & \text{if } n \in N^* \backslash (N_C^* \cup N_R^*) \\ \boldsymbol{push}(\,stk,\ [\![n]\!]'(\boldsymbol{top}(stk))\,) & \text{if } n \in N_C^* \\ \boldsymbol{push}(\,\boldsymbol{pop}(\boldsymbol{pop}(stk)),\ \mathcal{R}_n(\,\boldsymbol{top}(\boldsymbol{pop}(stk)),\ [\![n]\!]'(\boldsymbol{top}(stk))\,)\,) & \\ & \text{if } n \in N_R^* \end{cases}$$

where $[\![\ ]\!]' : N^* \to (\mathcal{C} \to \mathcal{C})$ denotes the straightforward extension of the semantic functional of Section 2 to interprocedural flow graphs[9], and $\mathcal{R}_n : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is a function as described below.

The intuition behind this definition is as follows:

The execution of an ordinary statement (i.e. $n \in N^* \backslash (N_C^* \cup N_R^*)$) only affects the currently valid activation record. Thus it can be modelled by simply modifying the top component of the stack representing the current data flow information.

A procedure call (i.e. $n \in N_C^*$) requires the generation of a new activation record. This is reflected by pushing a new element on the top of the stack, which usually results from modifying the top component of the stack according to the parameter transfer (cf. [KS2]).

The treatment of return statements (i.e. $n \in N_R^*$) demonstrates the necessity of introducing stacks into the framework: Returning from a procedure call (i.e. $n \in N_R^*$) essentially requires removal of the activation record belonging to the called procedure and reactivation of its predecessor. However, one observation is important here. The effect of a (directly) recursive procedure to a global variable needs to be maintained, whereas the local variables must be reset to their values at call time. Thus we need to consider the data flow information valid immediately before entering the procedure (available in $\boldsymbol{top}(\boldsymbol{pop}(stk))$), as well as the information valid after executing its body (available in $[\![n]\!]'(\boldsymbol{top}(stk))$), in order to compute the data flow information being valid after returning from the called procedure. The function $\mathcal{R}_n : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ models this computation (cf. [KS2]). Thus popping the top component of the stack and replacing the subsequent component by

$$\mathcal{R}_n(\,\boldsymbol{top}(\boldsymbol{pop}(stk)),\ [\![n]\!]'(\boldsymbol{top}(stk))\,)$$

reflects the whole process of completing a procedure call.

[7] We consider the operation $newstack$ instead of the usual $emptystack : \to STACK$ here, in order to exclude empty stacks, which are irrelevant in our framework.

[8] Therefore, we are never dealing with empty stacks.

[9] $[\![n]\!]^*(stk)$, $n \in N_R^*$, is only defined for stacks with at least two components, a fact, which is automatically be taken care of in any reasonable analysis context.

4.2 The Structure of the Semantic Functions

Let $\mathcal{F} =_{df} [STACK \to STACK]$ denote the set of all functions from $STACK$ to $STACK$ and

$$\mathcal{F}_O =_{df} \{\, f \in \mathcal{F} \mid \forall\, stk \in STACK.\ \boldsymbol{pop}(f(stk)) = \boldsymbol{pop}(stk) \,\}$$

$$\mathcal{F}_C =_{df} \{\, f \in \mathcal{F} \mid \forall\, stk \in STACK.\ \boldsymbol{pop}(f(stk)) = stk \,\}$$

$$\mathcal{F}_R =_{df} \{\, f \in \mathcal{F} \mid \forall\, stk \in STACK.\ \boldsymbol{pop}(f(stk)) = \boldsymbol{pop}(\boldsymbol{pop}(stk)) \,\}$$

Then we have:

Lemma 4.1

1. $\forall\, n \in N^* \backslash (N_C^* \cup N_R^*).\ [\![n]\!]^* \in \mathcal{F}_O$
2. $\forall\, n \in N_C^*.\ [\![n]\!]^* \in \mathcal{F}_C$
3. $\forall\, n \in N_R^*.\ [\![n]\!]^* \in \mathcal{F}_R$

Intuitively this means that the semantic function of an ordinary statement only affects the top component of the argument stack, that the semantic function of a call statement simply adds a new top component to the argument stack, and that a return statement replaces the upper two components of the argument stack by a new component. The following lemma is an easy consequence of these properties of $\mathcal{F}_O$, $\mathcal{F}_C$ and $\mathcal{F}_R$.

Lemma 4.2 $\quad \forall\, f_r \in \mathcal{F}_R\ \ \forall\, f_o, f_o' \in \mathcal{F}_O\ \ \forall\, f_c \in \mathcal{F}_C.\quad f_o \circ f_o',\ \ f_r \circ f_o \circ f_c \in \mathcal{F}_O$

The formal development of the paper requires the following derived notions of monotonicity and distributivity:

Definition 4.3 (S-Monotonicity (S-Distributivity))
A function $f \in \mathcal{F}_O \cup \mathcal{F}_C \cup \mathcal{F}_R$ is called

- s-monotonic *iff f_s is monotonic*
- s-distributive *iff f_s is distributive*

where f_s, the significant part *of f, is defined according to the following two cases:*

- *$f \in \mathcal{F}_O \cup \mathcal{F}_C$: Here $f_s : \mathcal{C} \to \mathcal{C}$ is defined by: $f_s(c) =_{df} \boldsymbol{top}(f(\boldsymbol{newstack}(c)))$*
- *$f \in \mathcal{F}_R$: Here $f_s : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is defined by*[10]*:*
  $$f_s(c_1, c_2) =_{df} \boldsymbol{top}(f(\boldsymbol{push}(\boldsymbol{newstack}(c_1), c_2)))$$

[10] Note that $\mathcal{C} \times \mathcal{C}$ is a lattice, whenever $\mathcal{C}$ is.

The following lemma is important, because it restricts the effort for checking the preconditions of the Interprocedural Safety Theorem 7.1 and the Interprocedural Coincidence Theorem 7.2 in a way that is comparable to the effort necessary for their intraprocedural counterparts (cf. Section 2 and 7).

Lemma 4.4 *For all* $n \in N^*$ *we have that* $[\![n]\!]^*$ *is s-monotonic (s-distributive) if*

$n \in N^*_R$: $[\![n]\!]'$ *and* $\mathcal{R}_n$ *are monotonic (distributive)*

$n \notin N^*_R$: $[\![n]\!]'$ *is monotonic (distributive)*

Conventions: In the following, we consider s-monotonicity (s-distributivity) as generalizations of the usual monotonicity (distributivity) by identifying lattice elements with their unique representations as one-component stacks. Moreover, we extend the meet operation $\sqcap$ to work on stacks in the following way:

$$\forall\, STK \subseteq STACK.\ \sqcap STK =_{df} newstack(\sqcap\{top(stk) \mid stk \in STK\})$$

Thus, the meet over a set of stacks is just the one-component stack containing the meet of all the top components in its single component.

4.3 The Interprocedural Meet Over all Paths Solution

Analogously to Section 2 the local abstract semantics $[\![\ ]\!]^*$ can be extended to cover finite interprocedural paths: for every path $p = (n_1, ..., n_q) \in \mathbf{IP}[m, n]$, we define $[\![p]\!]^* : STACK \rightarrow STACK$ by

$$[\![p]\!]^* =_{df} \begin{cases} id_{STACK} & \text{if } p \equiv \varepsilon \\ [\![(n_2, ..., n_q)]\!]^* \circ [\![n_1]\!]^* & \text{otherwise} \end{cases}$$

Now, as its intraprocedural counterpart (cf. [KU]), the interprocedural meet over all paths (*IMOP*) solution directly records all possible program executions leading to a particular program point. Here it is important that for any interprocedural path $p \in \mathbf{IP}[\mathbf{s}^*, n)$ and any stack $stk \in STACK$, $top([\![p]\!]^*(stk))$ is the only data flow information relevant for node n after executing p, since all other components of $[\![p]\!]^*(stk)$ correspond to activation records that are not valid after p. Identifying one-component stacks with the content of their unique component, the formal definition of the interprocedural meet over all paths solution is given by:

The ***IMOP*-Solution:**

$$\forall\, n \in N^*\ \ \forall\, c_0 \in \mathcal{C}.\ \ IMOP_{c_0}(n) \ = \ \sqcap \{\, [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, n)\,\}$$

4.4 The Interprocedural Maximal Fixed Point Solution

In addition to the equational characterization of the intraprocedural case (Equation System 2.1), flow graph systems require a preprocess, which determines the meaning of call nodes in terms of the meaning of the called procedures. This requires the introduction of an auxiliary semantic functional $[\![\ ]\!]$, which gives meaning to whole flow graphs. Essentially, $[\![n]\!]$ transforms data flow information that is assumed to be valid at the entry of the procedure that contains n into the corresponding data flow information being valid before an execution of n. In particular, $[\![\mathbf{e}_i]\!]$ is the meaning function of the i-th procedure[11]. Formally, the full preprocess for determining the meaning $[\![n]\!]$ of call nodes $n \in \mathbf{N}^S_C$ is characterized by:

Definition 4.5 $[\![\ ]\!] : \mathbf{N}^S \rightarrow (STACK \rightarrow STACK)$ *and* $[\![\ ]\!] : \mathbf{N}^S \rightarrow (STACK \rightarrow STACK)$ *are defined as the greatest solution of the equation system given by:*

$$[\![n]\!] =_{df} \begin{cases} id_{STACK} & \text{if } n \in \{\, \mathbf{s}_0, .., \mathbf{s}_k \,\} \\ \sqcap\{[\![m]\!] \circ [\![m]\!] \mid m \in pred_{fg(n)}(n)\} & \text{otherwise} \end{cases}$$

and

$$[\![n]\!] =_{df} \begin{cases} [\![n]\!]^* & \text{if } n \in \mathbf{N}^S \backslash \mathbf{N}^S_C \\ [\![n_R]\!]^* \circ [\![end(decl(n))]\!] \circ [\![n_C]\!]^* & \text{otherwise} \end{cases}$$

where id_{STACK} *denotes the identity on* $STACK$*, and* $\sqcap$ *the "componentwise" meet operation on* $\mathcal{F}_O$[12]*.*

The effect of a procedure call $n \in \mathbf{N}^S_C$ is determined in three steps reflecting the three phases of its execution:

- *Entering* the called procedure: $[\![n_C]\!]^*$ creates a new activation record by transforming the content of the top component of the stack according to the semantics of the call node and pushing it onto the stack. – Usually, the semantics of call nodes will reflect the parameter transfer (cf. [KS2]).
- *Evaluating* the call: $[\![end(decl(n))]\!]$ computes the effect of the procedure body. Note that this affects the top component of the argument stack only.
- *Leaving* the called procedure: $[\![n_R]\!]^*$ removes the activation record related with the current procedure call by popping the top component from the stack, and replacing its subsequent component by the data flow information representing the effect of the procedure call relative to its call site.

Applying Lemma 4.2, we obtain

Lemma 4.6 $\forall\, n \in \mathbf{N}^S.\ [\![n]\!],\ \ [\![n]\!] \in \mathcal{F}_O$

Remark 4.7 Lemma 4.6 is important, since it shows that all the stacks occurring during the iterative computation of the *IMFP*-solution will have at most two components[13]. This is in contrast to the *IMOP*-strategy, where the size of stacks contributing to the *IMOP*-solution is in general unbounded. Moreover, it allows us to prove termination in the usual way.

[11]Remember, $[\![\mathbf{e}_i]\!]' =_{df} id_C$. Thus, $\mathbf{e}_i$ is related to the identity on $STACK$.

[12]$\forall\, f, f' \in \mathcal{F}_O.\ f \sqcap f' =_{df} f'' \in \mathcal{F}_O$ with $\forall\, stk \in STACK.\ top(f''(stk)) = top(f(stk)) \sqcap top(f'(stk))$. As usual, "$\sqcap$" induces an inclusion relation "$\sqsubseteq$" on $\mathcal{F}_O$ by: $f \sqsubseteq f'$ iff $f \sqcap f' = f$.

[13]This is because the computation starts from a one-component stack $newstack(c_0)$ (cf. Equation System 4.8 and Algorithm 5.4).

After fixing the meaning of call nodes, $[\![\ ]\!]$ plays essentially the same role as the local (abstract) semantic functional of Section 2. Formally, the interprocedural maximal fixed point strategy is characterized by Equation System 4.8. As its intraprocedural counterpart, this strategy labels every node n of $\mathbf{N}^S$ with a pre-information $\mathbf{pre}_{c_0}(n)$ and a post-information $\mathbf{post}_{c_0}(n)$, whose top components are the greatest solution of this equation system with respect to $c_0 \in \mathcal{C}$.

Equation System 4.8

$$\mathbf{pre}(n) = \begin{cases} \mathit{newstack}(c_0) & \text{if } n = \mathbf{s}_0 \\ \bigsqcap \{ [\![m_C]\!]^*(\mathbf{pre}(m)) \,|\, m \in \mathit{calls}(\mathit{fg}(n)) \} & \text{if } n \in \{\mathbf{s}_1, .., \mathbf{s}_k\} \\ \bigsqcap \{ \mathbf{post}(m) \,|\, m \in \mathit{pred}_{\mathit{fg}(n)}(n) \} & \text{otherwise} \end{cases}$$

$$\mathbf{post}(n) = [\![n]\!](\mathbf{pre}(n))$$

As before, identifying a stack having a single component only with the content of this component, we obtain as in the intraprocedural case:

The *IMFP*-**Solution:** $\forall\, n \in \mathbf{N}^S\ \forall\, c_0 \in \mathcal{C}.\ \mathit{IMFP}_{c_0}(n) = \mathbf{pre}_{c_0}(n)$

5 Algorithms

This section provides algorithms that compute the *IMFP*-solution. First, we have an algorithm for the preprocess that determines the semantics of call nodes:

Algorithm 5.1 (Computation of the Semantic Functionals $[\![\ ]\!]$ and $[\![\![\ ]\!]\!]$)

Input:
A flow graph system $S = \{G_0, G_1, ., G_k\}$, a complete semi-lattice $\mathcal{C}$, for every node $n \in \mathbf{N}^S \backslash \mathbf{N}^S_C$ a s-monotonic function $[\![n]\!]^* : STACK \to STACK \in \mathcal{F}_O$, which is the identity for all nodes in $\{\mathbf{s}_0, ..., \mathbf{s}_k, \mathbf{e}_0, ..., \mathbf{e}_k\}$. Moreover, for every node $n \in \mathbf{N}^S_C$ two s-monotonic functions $[\![n_C]\!]^* : STACK \to STACK \in \mathcal{F}_C$ and $[\![n_R]\!]^* : STACK \to STACK \in \mathcal{F}_R$. All semantic functions $[\![n]\!]^*$, $n \in N^* \backslash N^*_R$, are assumed to map stacks with $top(stk) = \top$ to stacks with top component $\top$. Analogously, all semantic functions $[\![n]\!]^*$, $n \in N^*_R$, are assumed to map stacks with upper two components $\top$ to stacks with top component $\top$ ("$\top$-assumption").

Output:
An annotation of S with functions $[\![\![n]\!]\!] : STACK \to STACK$ (stored in gtr) and $[\![n]\!] : STACK \to STACK$ (stored in ltr) that satisfy Definition 4.5.

Remark: *$\top_{\mathcal{F}_O} : STACK \to STACK \in \mathcal{F}_O$ denotes the "universal" function which is assumed to "contain" every function $f \in \mathcal{F}_O$, and id_{STACK} is the identity on STACK. The variable workset controls the iterative process. Its elements are tuples, whose first components are nodes $m \in \mathbf{N}^S$ of the flow graph system S, and whose second components are functions $f : STACK \to STACK \in \mathcal{F}_O$ that specify a new approximation for the function $[\![\![m]\!]\!]$ of the node of the first component. Note that due to the mutual interdependence of the definitions of $[\![\![\ ]\!]\!]$ and $[\![\ ]\!]$ the iterative approximation of $[\![\![\ ]\!]\!]$ is superposed by an interprocedural iteration step which updates the semantics $[\![\ ]\!]$ of call nodes.*

(Initialization of the annotation arrays gtr and ltr and the variable workset)
FOR $m \in \mathbf{N}^S$ **DO**
  $gtr[m] := \top_{\mathcal{F}_O}$;
  IF $m \in \mathbf{N}^S_C$ **THEN** $ltr[m] := \top_{\mathcal{F}_O}$ **ELSE** $ltr[m] := [\![m]\!]^*$ **FI**
OD;
$workset := \{ (\mathbf{s}, id_{STACK}) \,|\, \mathbf{s} \in \{\mathbf{s}_0, ..., \mathbf{s}_k\} \}$;

(Iterative fixed point computation)
WHILE $workset \neq \emptyset$ **DO**
  LET $(m, f) \in workset$
    BEGIN
      $workset := workset \backslash \{ (m, f) \}$;
      IF $gtr[m] \sqsupset gtr[m] \sqcap f$
        THEN
          $gtr[m] := gtr[m] \sqcap f$;
          IF $m \in \{\mathbf{e}_i \,|\, i \in \{0, ., k\}\}$
            THEN
              FOR $l \in calls(fg(m))$ **DO**
                $ltr[l] := [\![l_R]\!]^* \circ gtr[m] \circ [\![l_C]\!]^*$;
                $workset := workset \cup \{(n, ltr[l] \circ gtr[l]) \,|\, n \in succ_{fg(l)}(l)\}$
              OD
            ELSE
              $workset := workset \cup \{ (n, ltr[m] \circ gtr[m]) \,|\, n \in succ_{fg(m)}(m) \}$
          FI
      FI
    END
OD.

In order to simplify the formulation of the central property of this algorithm, we abbreviate the values of *workset*, $ltr[n]$ and $gtr[n]$ after the k-th execution of the while-loop by $workset^k$, $ltr^k[n]$ and $gtr^k[n]$, respectively. We have:

Proposition 5.2

1. $\forall\, n \in \mathbf{N}^S\ \forall\, k \in \omega.\ ltr^k[n] \in \mathcal{F}_O \ \wedge\ gtr^k[n] \in \mathcal{F}_O$
2. $\forall\, n \in \mathbf{N}^S\ \forall\, k \in \omega.\ (ltr^{k+1}[n], gtr^{k+1}[n]) \sqsubseteq (ltr^k[n], gtr^k[n])$
3. $\forall\, n \in \mathbf{N}^S\ \forall\, k \in \omega.\ ltr^k[n] = \begin{cases} [\![n_R]\!] \circ gtr^k[end(decl(n))] \circ [\![n_C]\!]^* & \text{if } n \in \mathbf{N}^S_C \\ [\![n]\!]^* & \text{otherwise} \end{cases}$

The first part of Proposition 5.2 follows immediately from Lemma 4.2, the second part from the s-monotonicity of the semantic functions $[\![\ ]\!]^*$, and the third part from a straightforward induction on the number k of iterations of the while-loop of Algorithm 5.1 (cf. proof of Theorem 5.3). The central result concerning Algorithm 5.1, however, is:

Theorem 5.3 $\forall\, n \in \mathbf{N}^S.\ [\![\, n \,]\!] \;=\; \bigsqcap\{ltr^k[n] \mid k \geq 0\} \;\;\wedge\;\; [\![\, n \,]\!] \;=\; \bigsqcap\{gtr^k[n] \mid k \geq 0\}$
In particular, we have $\forall\, n \in \mathbf{N}^S.\ [\![\, n \,]\!] = gtr[n] \;\wedge\; [\![\, n \,]\!] = ltr[n]$ *after termination of Algorithm 5.1.*

Proof. Let *fix-gtr* and *fix-ltr* denote an arbitrary solution of the equation system of Definition 4.5 for $[\![\ ]\!]$ and $[\![\ ]\!]$, respectively. The central step in proving Theorem 5.3 is to check the following four invariants of Algorithm 5.1:

1. $\forall\, k \in \omega\ \forall\, n \in \mathbf{N}^S.\ (\textit{fix-gtr}[n], \textit{fix-ltr}[n]) \sqsubseteq (gtr^k[n], ltr^k[n])$
2. $\forall\, k \in \omega\ \forall\, n \in \mathbf{N}^S.\ gtr^k[n] \sqcap \bigsqcap\{\, f \mid (n, f) \in \textit{workset}^k \,\} =$

$$\begin{cases} id_{STACK} & \text{if } n \in \{\mathbf{s}_0, ..., \mathbf{s}_k\} \\ \bigsqcap\{\, ltr^k[m] \circ gtr^k[m] \mid m \in pred_{fg(n)}(n) \,\} & \text{otherwise} \end{cases}$$

3. $\forall\, k \in \omega\ \forall\, n \in \mathbf{N}^S.\ ltr^k[n] =$

$$\begin{cases} [\![\, n_R \,]\!]^* \circ gtr^k[end(decl(n))] \circ [\![\, n_C \,]\!]^* & \text{if } n \in \mathbf{N}^S_C \\ [\![\, n \,]\!]^* & \text{otherwise} \end{cases}$$

4. $\forall\, (m, f) \in \textit{workset}^k.\ \textit{fix-gtr}[m] \sqsubseteq f$

The invariants (1) to (4) are proved simultaneously by induction on the number k of iterations of the while-loop. For $k = 0$, we obtain by investigating the initialization part of Algorithm 5.1:

$$\begin{array}{ll} (a) & \forall\, n \in \mathbf{N}^S.\ gtr^0[n] = \top_{\mathcal{F}_O} \\ (b) & \forall\, n \in \mathbf{N}^S.\ ltr^0[n] = \begin{cases} \top_{\mathcal{F}_O} & \text{if } n \in \mathbf{N}^S_C \\ [\![\, n \,]\!]^* & \text{otherwise} \end{cases} \\ (c) & \textit{workset}^0 = \{\, (\mathbf{s}, id_{STACK}) \mid \mathbf{s} \in \{\mathbf{s}_0, ..., \mathbf{s}_k\} \,\} \end{array}$$

Thus, (1) is an immediate consequence of (a) and (b), and (2) is trivial for all nodes $n \in \mathbf{N}^S \backslash \{\mathbf{s}_0, ..., \mathbf{s}_k\}$. For the remaining nodes $\mathbf{s} \in \{\mathbf{s}_0, ..., \mathbf{s}_k\}$, (2) holds because of

$$gtr^0[\mathbf{s}] \sqcap id_{STACK} = \top_{\mathcal{F}_O} \sqcap id_{STACK} = id_{STACK}$$

(3) is obviously satisfied for nodes $n \in \mathbf{N}^S \backslash \mathbf{N}^S_C$, and for nodes $n \in \mathbf{N}^S_C$ it is a consequence of the $\top$-assumption[14]. Finally, (4) is satisfied, since *workset* initially contains only nodes $\mathbf{s} \in \{\mathbf{s}_0, ..., \mathbf{s}_k\}$, which satisfy $\textit{fix-gtr}[\mathbf{s}] = id_{STACK}$ according to the choice of *fix-gtr* and Definition 4.5.

[14] See Algorithm 5.1 (**INPUT**).

For the induction step let $k>0$ and (m, f) be the element currently chosen from the workset. Obviously, we have

$$(d)\quad gtr^k[n] = \begin{cases} gtr^{k-1}[n] \sqcap f & \text{if } n = m \;\wedge\; gtr^{k-1}[n] \sqsupset gtr^{k-1}[n] \sqcap f \\ gtr^{k-1}[n] & \text{otherwise} \end{cases}$$

$$(e)\quad ltr^k[n] = \begin{cases} [\![\, n_R \,]\!]^* \circ gtr^k[m] \circ [\![\, n_C \,]\!]^* & \text{if } n \in calls(fg(m)) \;\wedge \\ & \quad m \in \{\, \mathbf{e}_0, ..., \mathbf{e}_k \,\} \;\wedge\; gtr^k[m] \sqsubset gtr^{k-1}[m] \\ ltr^{k-1}[n] & \text{otherwise} \end{cases}$$

Moreover the induction hypothesis yields:

$$\begin{array}{ll} (i) & \forall\, n \in \mathbf{N}^S.\ (\textit{fix-gtr}[n], \textit{fix-ltr}[n]) \sqsubseteq (gtr^{k-1}[n], ltr^{k-1}[n]) \\ (ii) & \textit{fix-gtr}[m] \sqsubseteq f \quad \text{and} \\ (iii) & \forall\, n \in \mathbf{N}^S.\ ltr^{k-1}[n] = \begin{cases} [\![\, n_R \,]\!]^* \circ gtr^{k-1}[end(decl(n))] \circ [\![\, n_C \,]\!]^* & \text{if } n \in \mathbf{N}^S_C \\ [\![\, n \,]\!]^* & \text{otherwise} \end{cases} \end{array}$$

Together this directly proves (1) and (3).

During the induction step for the remaining two invariants (2) and (4), we can assume

$$gtr^k[m] \;=\; gtr^{k-1}[m] \;\sqcap\; f \sqsubset gtr^{k-1}[m]$$

because they would otherwise be trivial by the induction hypothesis.

Considering invariant (2) first, we obtain the induction step in the case of $n \equiv m$ as an immediate consequence of (d) and the induction hypothesis. Otherwise, if $m \in \{\mathbf{e}_0, .., \mathbf{e}_k\}$ and $n \in \bigcup\{\, succ_{fg(l)}(l) \mid l \in calls(fg(m)) \,\}$, or if $n \in succ_{fg(m)}(m)$, (2) follows from the induction hypothesis and the updating of the workset during the k^{th} iteration of the while-loop. For the remaining nodes (2) is a trivial consequence of the induction hypothesis.

Thus, we are left with showing the induction step for invariant (4). Here we have to consider two cases. If $m \notin \{\, \mathbf{e}_0, ..., \mathbf{e}_k \,\}$, we have to show

$$(*)\qquad \forall\, n \in succ_{fg(m)}(m).\ \textit{fix-gtr}[n] \sqsubseteq ltr^k[m] \circ gtr^k[m]$$

and if $m \in \{\, \mathbf{e}_0, ..., \mathbf{e}_k \,\}$, we have to show

$$(**)\qquad \forall\, l \in calls(fg(m))\ \forall\, n \in succ_{fg(l)}(l).\ \textit{fix-gtr}[n] \sqsubseteq ltr^k[l] \circ gtr^k[l]$$

According to the choice of *fix-gtr* and *fix-ltr* and Definition 4.5, we have in the first case

$$\forall\, n \in succ_{fg(m)}(m).\ \textit{fix-gtr}[n] \sqsubseteq \textit{fix-ltr}[m] \circ \textit{fix-gtr}[m]$$

and in the second one

$$\forall\, l \in calls(fg(m))\ \forall\, n \in succ_{fg(l)}(l).\ \textit{fix-gtr}[n] \sqsubseteq \textit{fix-ltr}[l] \circ \textit{fix-gtr}[l]$$

Thus, $(*)$ and $(**)$ are immediate consequences of (1), and the s-monotonicity of all semantic functions, which completes the proof of the remaining invariant (4).

Now, the first part of Theorem 5.3 is a consequence of Proposition 5.2(2), the invariants (1), (2), and (3), and the equations

$$\bigsqcap\{gtr^k[n] \mid k \geq 0\} \;=\; \bigsqcap\{\, f \mid k>0 \;\wedge\; (n, f) \in \textit{workset}^k \,\}$$

and

$$\bigsqcap \{ ltr^k[n] \mid k \geq 0 \} \;=\; \begin{cases} [\![n_R]\!]^* \circ gtr^k[end(decl(n))] \circ [\![n_C]\!]^* & \text{if } n \in \mathbf{N}^S_C \\ [\![n]\!]^* & \text{otherwise} \end{cases}$$

which hold due to the commutativity and associativity of $\sqcap$. The second, more specific part then follows from the invariants (2) and (3), and the emptiness of the workset after the termination of Algorithm 5.1. $\Box$

The second algorithm computes the *IMFP*-Solution:

Algorithm 5.4 (Computation of the *IMFP*-Solution)

Input:
A flow graph system $S = \{G_0, G_1, ., G_k\}$*, the semantic functional* $[\![\ ]\!]$*, for every node* $n \in \mathbf{N}^S_C$ *the function* $[\![n_C]\!]^*$*, and a start information* $c_0 \in \mathcal{C}$*. All semantic functions* $[\![n]\!]$ *and* $[\![n_C]\!]^*$ *are assumed to be s-monotonic and to map stacks with* $top(stk) = \top$ *to stacks with top component* $\top$ *("*$\top$*-assumption").*

Output:
An annotation of S *with data flow informations, i.e. an annotation with pre-informations (stored in pre) and post-informations (stored in post) of one-component stacks that characterize valid data flow information at the entry and at the exit of every node.*

Remark:
$newstack(\top)$ *denotes the "universal" data flow information, which is assumed to "contain" every data flow information. The variable workset controls the iterative process. Its elements are tuples whose first components are nodes* $m \in \mathbf{N}^S$ *of the flow graph system* S *and whose second components are elements of STACK specifying a new approximation for the pre-information of the node of the first component.*

(Initialization of the annotation arrays pre and post and the variable workset)
FOR $m \in \mathbf{N}^S$ **DO** $(pre[m], post[m]) := (\, newstack(\top), newstack(\top)\,)$ **OD**;
$workset := \{\, (\mathbf{s}_0, newstack(c_0))\, \}$;

(Iterative fixed point computation)
WHILE $workset \neq \emptyset$ **DO**
  LET $(m, stk) \in workset$
    BEGIN
      $workset := workset \setminus \{\, (m, stk)\, \}$;
      IF $pre[m] \sqsupset pre[m] \sqcap stk$
        THEN
          $pre[m] := pre[m] \sqcap stk$;
          $post[m] := [\![m]\!](pre[m])$;
          $workset := workset \cup \{\, (n, post[m]) \mid n \in succ_{fg(m)}(m)\, \}$;
          IF $m \in \mathbf{N}^S_C$
            THEN $workset := workset \cup \{\, (start(decl(m)), [\![m_C]\!]^*(pre[m]))\, \}$
          FI
      FI
    END
OD.

As before, given a start information c_0, we abbreviate the values of $workset$, $pre[n]$, and $post[n]$ after the k-th execution of the while-loop by $workset^k$, $pre^k[n]$, and $post^k[n]$. In analogy to Proposition 5.2 and Theorem 5.3 we have:

Proposition 5.5

1. $\forall\, n \in \mathbf{N}^S\ \forall\, k \in \omega.\ pop(pre^k[n]) = emptystack \quad \wedge \quad pop(post^k[n]) = emptystack$
2. $\forall\, n \in \mathbf{N}^S\ \forall\, k \in \omega.\ (pre^{k+1}[n], post^{k+1}[n]) \sqsubseteq (pre^k[n], post^k[n])$

Theorem 5.6 $\forall\, n \in \mathbf{N}^S.\ IMFP_{c_0}(n) \;=\; \bigsqcap \{ pre^k[n] \mid k \geq 0 \}$
In particular, we have $\forall\, n \in \mathbf{N}^S.\ IMFP_{c_0}(n) = pre[n]$ *after termination of Algorithm 5.4.*

Proof. Let *fix-pre* and *fix-post* denote an arbitrary solution of Equation System 4.8 with respect to $newstack(c_0) \in STACK$. In analogy to the proof of Theorem 5.3, we first prove the following four invariants of Algorithm 5.4 by simultaneous induction on the number k of iterations of the while-loop.

1. $\forall\, k \in \omega\ \forall\, n \in \mathbf{N}^S.\ (\textit{fix-pre}[n], \textit{fix-post}[n]) \sqsubseteq (pre^k[n], post^k[n])$
2. $\forall\, k \in \omega\ \forall\, n \in \mathbf{N}^S.\ pre^k[n] \sqcap \bigsqcap \{\, stk \mid (n, stk) \in workset^k\, \} =$

$$\begin{cases} newstack(c_0) & \text{if } n = \mathbf{s}_0 \\ \bigsqcap \{\, [\![m_C]\!]^*(pre^k[m]) \mid m \in calls(fg(n))\, \} & \text{if } n \in \{\mathbf{s}_1, \ldots, \mathbf{s}_k\} \\ \bigsqcap \{\, post^k[m] \mid m \in pred_{fg(n)}(n)\, \} & \text{otherwise} \end{cases}$$

3. $\forall\, k \in \omega\ \forall\, n \in \mathbf{N}^S.\ post^k[n] = [\![n]\!](pre^k[n])$

4. $\forall\,(m, stk) \in workset^k.\ \textit{fix-pre}[m] \sqsubseteq stk$

For $k=0$, the investigation of the initialization part of Algorithm 5.4 yields:

$$\begin{array}{ll} (a) & \forall\, n \in \mathbf{N}^S.\ pre^0[n] = post^0[n] = newstack(\top) \quad \text{and} \\ (b) & workset^0 = \{\,(\mathbf{s}_0, newstack(c_0))\,\} \end{array}$$

Thus (1) is an immediate consequence of (a), and (2) is trivial for all nodes $n \neq \mathbf{s}_0$. Otherwise, i.e. if $n = \mathbf{s}_0$, (2) holds because of

$$pre^0[\mathbf{s}_0] \sqcap newstack(c_0) = newstack(\top) \sqcap newstack(c_0) = newstack(c_0)$$

(3) is a consequence of the $\top$-assumption[15], and (4) of $\textit{fix-pre}[\mathbf{s}_0] = newstack(c_0)$, which holds according to the choice of *fix-pre* and Equation System 4.8.

For the induction step let $k>0$ and (m, stk) be the element currently chosen from the workset. Obviously, we have

$$(c) \quad pre^k[n] = \begin{cases} pre^{k-1}[n] \sqcap stk & \text{if } n = m \ \wedge\ pre^{k-1}[n] \sqsupset pre^{k-1}[n] \sqcap stk \\ pre^{k-1}[n] & \text{otherwise} \end{cases}$$

$$(d) \quad post^k[n] = \begin{cases} [\![\, n \,]\!](pre^k[n]) & \text{if } n = m \ \wedge\ pre^{k-1}[n] \sqsupset pre^{k-1}[n] \sqcap stk \\ post^{k-1}[n] & \text{otherwise} \end{cases}$$

Moreover the induction hypothesis yields:

$$\begin{array}{ll} (i) & \forall\, n \in \mathbf{N}^S.\ (\textit{fix-pre}[n], \textit{fix-post}[n]) \sqsubseteq (pre^{k-1}[n], post^{k-1}[n]) \\ (ii) & \textit{fix-pre}[m] \sqsubseteq stk \quad \text{and} \\ (iii) & \forall\, n \in \mathbf{N}^S.\ post^{k-1}[n] = [\![\, n \,]\!](pre^{k-1}[n]) \end{array}$$

Together this directly proves (1) and (3).

During the induction step for the remaining two invariants (2) and (4), we can assume

$$(pre^k[m], post^k[m]) \ = \ (pre^{k-1}[m] \sqcap stk, [\![\, m \,]\!](pre^{k-1}[m] \sqcap stk)) \sqsubset (pre^{k-1}[m], post^{k-1}[m])$$

because they would otherwise be trivial by the induction hypothesis. Considering invariant (2) first, we obtain the induction step in the case of $n = m$ as an immediate consequence of (c) and the induction hypothesis. Otherwise, if $n \in succ_{fg(m)}(m)$, or if $m \in \mathbf{N}^S_C$ and $n = start(decl(m))$, (2) follows from the induction hypothesis and the updating of the workset during the k^{th} iteration of the while-loop. For the remaining nodes (2) is a trivial consequence of the induction hypothesis.

Thus, we are left with showing the induction step for invariant (4). Here we have to show

$$(*) \quad \forall\, n \in succ_{fg(m)}(m).\ \textit{fix-pre}[n] \sqsubseteq post^k[m]$$

and, if $m \in \mathbf{N}^S_C$, additionally

[15] See Algorithm 5.4 (**INPUT**).

$$(**) \quad \textit{fix-pre}[start(decl(m))] \sqsubseteq [\![\, m_C \,]\!]^*(pre^k[m])$$

According to the choice of *fix-pre* and Equation System 4.8, we have

$$\forall\, n \in succ_{fg(m)}(m).\ \textit{fix-pre}[n] \sqsubseteq \textit{fix-post}[m]$$

and, if $m \in \mathbf{N}^S_C$, additionally

$$\textit{fix-pre}[start(decl(m))] \sqsubseteq [\![\, m_C \,]\!]^*(\textit{fix-pre}[m])$$

Thus, $(*)$ and $(**)$ are immediate consequences of (1), and the s-monotonicity of all semantic functions, which completes the proof of the remaining invariant (4).

Now, the first part of Theorem 5.6 is a consequence of Proposition 5.5(2), the invariants (1) and (2), and equation

$$\bigsqcap \{pre^k[n] \mid k \geq 0\} = \bigsqcap \{\, c \mid k>0 \ \wedge\ (n, c) \in workset^k \,\}$$

which holds due to the commutativity and associativity of $\sqcap$. The second, more specific part then follows immediately from invariant (2), and the emptiness of the workset after the termination of Algorithm 5.4. □

6 The Main Lemma

The composition and meet of s-monotonic (s-distributive) functions is again s-monotonic (s-distributive). Thus we have:

Lemma 6.1 *The semantic functions* $[\![\, n \,]\!]$ *and* $[\![n]\!]$, $n \in \mathbf{N}^S$, *are s-monotonic (s-distributive) iff the semantic functions* $[\![\, m \,]\!]^*$, $m \in N^*$, *are s-monotonic (s-distributive).*

In the remainder of this section, we assume that the semantic functions are s-monotonic or s-distributive, and we define $imop_n : STACK \rightarrow STACK$ by

$$imop_n =_{df} \begin{cases} id_{STACK} & \text{if } n \in \{\, \mathbf{s}^*_0, ..., \mathbf{s}^*_k \,\} \\ \bigsqcap \{\, [\![\, p \,]\!]^* \mid p \in \mathbf{CIP}[start(fg(n)), n) \,\} & \text{otherwise} \end{cases}$$

We have:

Lemma 6.2 *For all* $n \in \mathbf{N}^S$, *we have, if the semantic functions* $[\![\, m \,]\!]^*$, $m \in N^*$, *are*

1. *s-monotonic:* $[\![n]\!] \sqsubseteq imop_n$
2. *s-distributive:* $[\![n]\!] = imop_n$

Proof. The first part, $[\![n]\!] \sqsubseteq imop_n$, is an immediate consequence of

$$(*) \quad \forall\, n \in \mathbf{N}^S\ \forall\, p \in \mathbf{CIP}[start(fg(n)), n).\ [\![n]\!] \sqsubseteq [\![\, p \,]\!]^*$$

which we prove by induction on the length k of path p. Paths of length 0 only reach nodes in $\{\mathbf{s}_0, \mathbf{s}_1, .., \mathbf{s}_k\}$ for which $(*)$ is trivial. Moreover, the only complete interprocedural paths that end in a start node $\mathbf{s} \in \{\mathbf{s}_0, \mathbf{s}_1, .., \mathbf{s}_k\}$ are of length 0. Thus let $n \in \mathbf{N}^S \backslash \{\mathbf{s}_0, \mathbf{s}_1, .., \mathbf{s}_k\}$ such that there is a path $p \in \mathbf{CIP}[start(fg(n)), n)$ with $0 < lgth(p) \leq k$. Then we must show

$$[\![\, p \,]\!]^* \sqsupseteq [\![\, n \,]\!]$$

under the induction hypothesis

(IH) $\quad \forall\, m \in \mathbf{N}^S \; (\forall\, p \in \mathbf{CIP}[start(fg(m)), m).\ 0 \leq lgth(p) < k).\ [\![\, p \,]\!]^* \sqsupseteq [\![\, m \,]\!]$

Let $m \in pred^*(n)$ and $p' \in \mathbf{IP}[start(fg(n)), m)$ such that $p = p';(m)$. Now supposing $m \in N^* \backslash (N_C^* \cup N_R^*)$, we have $m \notin \mathbf{N}_C^S$, and $p' \in \mathbf{CIP}[start(fg(n)), m)$. Therefore, we obtain as desired:

$$\begin{array}{lcl}
[\![\, p \,]\!]^* & = & [\![\, m \,]\!]^* \circ [\![\, p' \,]\!]^* \\
(m \notin \mathbf{N}_C^S) & = & [\![\, m \,]\!] \circ [\![\, p' \,]\!]^* \\
(\text{IH, s-monotonicity}) & \sqsupseteq & [\![\, m \,]\!] \circ [\![\, m \,]\!] \\
(\text{Definition 4.5 and } n \notin \{\mathbf{s}_0, \mathbf{s}_1, .., \mathbf{s}_k\}) & \sqsupseteq & [\![\, n \,]\!]
\end{array}$$

On the other hand, $m \in N_C^*$ cannot occur because of $n \notin \{\mathbf{s}_0, \mathbf{s}_1, .., \mathbf{s}_k\}$. Thus we are left with the case $m \in N_R^*$, which is problematic, because here m is not the predecessor of n in $\mathbf{N}^S$. This situation, which does not allow the direct application of the induction hypothesis, can be pictured as follows:

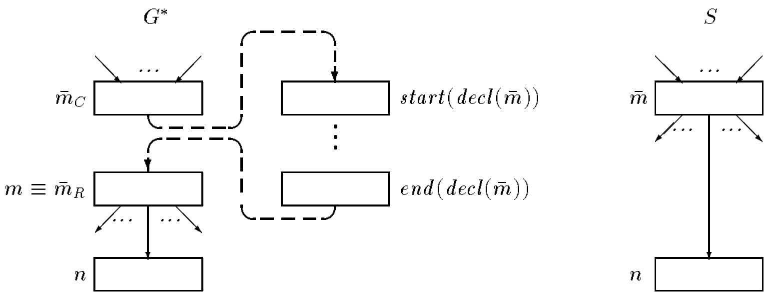

Figure 6.3

Now, identifying $\bar{m}$ and $\bar{m}_C$ and considering the decomposition

$$p \;=\; \bar{p}; (\bar{m}_C); (start(decl(\bar{m})), ..., end(decl(\bar{m}))); (\bar{m}_R)$$

we have by means of Lemma 3.8

$$\bar{p} \in \mathbf{CIP}[start(fg(n)), \bar{m})$$

and

$$(start(decl(\bar{m})), ..., end(decl(\bar{m}))) \in \mathbf{CIP}[start(decl(\bar{m})), end(decl(\bar{m}))]$$

Abbreviating $decl(\bar{m})$ by G', the first inclusion can therefore be completed by[16]:

$$\begin{array}{lcl}
[\![\, p \,]\!]^* & = & [\![\, \bar{m}_R \,]\!]^* \circ [\![\, (start(G'), ., end(G')) \,]\!]^* \circ [\![\, \bar{m}_C \,]\!]^* \circ [\![\, \bar{p} \,]\!]^* \\
(\text{IH, s-monotonicity}) & \sqsupseteq & [\![\, \bar{m}_R \,]\!]^* \circ [\![\, end(G') \,]\!] \circ [\![\, \bar{m}_C \,]\!]^* \circ [\![\, \bar{p} \,]\!]^* \\
(\text{Definition 4.5}) & = & [\![\, \bar{m} \,]\!] \circ [\![\, \bar{p} \,]\!]^* \\
(\text{IH, s-monotonicity}) & \sqsupseteq & [\![\, \bar{m} \,]\!] \circ [\![\, \bar{m} \,]\!] \\
(\text{Definition 4.5}) & \sqsupseteq & [\![\, n \,]\!]
\end{array}$$

Using Theorem 5.3, the second part of Lemma 6.2, $[\![\, n \,]\!] \sqsupseteq imop_n$, is a consequence of:

$$(*) \quad \forall\, k \geq 0\; \forall\, n \in \mathbf{N}^S.\ (1)\ imop_n \sqsubseteq gtr^k[n] \;\text{ and }\; (2)\ [\![\, n \,]\!] \sqsubseteq ltr^k[n]$$

which is proved by (simultaneous) induction on the number k of times the while-loop in Algorithm 5.1 is executed.

The induction base is an immediate consequence of the initialization part of the algorithm. Thus let $k>0$, and let us consider the k-th iteration of the while-loop under the induction hypothesis (IH) that $(*)$ holds before the k-th execution of the while-loop. To show (1), we only need to consider the node $n \in \mathbf{N}^S$, which belongs to the workset entry (n, f) being processed in the k-th iteration, because the gtr-information associated with all the other nodes remains unchanged in this iteration and therefore satisfies $(*)$ according to (IH).

If $n \equiv \mathbf{s} \in \{\mathbf{s}_0, ..., \mathbf{s}_k\}$, we have $f = id_{STACK}$, since the workset is initialized with $\{(\mathbf{s}, id_{STACK}) \mid \mathbf{s} \in \{\mathbf{s}_0, .., \mathbf{s}_k\}\}$ and no node in $\{\mathbf{s}_0, .., \mathbf{s}_k\}$ occurs more than once in the workset. Thus, we have as desired

$$imop_{\mathbf{s}} = id_{STACK} = gtr^k[\mathbf{s}]$$

This leaves us with the case $n \in \mathbf{N}^S \backslash \{\mathbf{s}_0, .., \mathbf{s}_k\}$. Thus there exists $m \in \mathbf{N}^S$ and $k > l \geq 0$ such that

$$(n, f) \;=\; (n, ltr^l[m] \circ gtr^l[m])$$

was added in the l-th iteration to the workset. This allows us to deduce $imop_n \sqsubseteq f$ by investigating the following two cases:

Case 1. $m \notin \mathbf{N}_C^S$

Here we have:

$$\begin{array}{lcl}
imop_n & = & \sqcap \{\, [\![\, p \,]\!]^* \mid p \in \mathbf{CIP}[start(fg(n)), n) \,\} \\
(m \notin \mathbf{N}_C^S) & \sqsubseteq & \sqcap \{\, [\![\, m \,]\!]^* \circ [\![\, p \,]\!]^* \mid p \in \mathbf{CIP}[start(fg(n)), m) \,\} \\
(\text{s-distributivity}) & = & [\![\, m \,]\!]^* \circ \sqcap \{\, [\![\, p \,]\!]^* \mid p \in \mathbf{CIP}[start(fg(n)), m) \,\} \\
(\text{Def. } imop_m) & = & [\![\, m \,]\!]^* \circ imop_m \\
(\text{IH, s-monotonicity}) & \sqsubseteq & [\![\, m \,]\!]^* \circ gtr^l[m] \\
(m \notin \mathbf{N}_C^S) & = & ltr^l[m] \circ gtr^l[m] \;=\; f
\end{array}$$

Case 2. $m \in \mathbf{N}_C^S$

Abbreviating $start(decl(m))$ and $end(decl(m))$ by $\mathbf{s}'$ and $\mathbf{e}'$, respectively, we obtain

[16] Remember that $[\![\, end(G') \,]\!] = id_{STACK}$.

by means of the s-distributivity and the s-monotonicity[17]:

$$\begin{array}{lll}
imop_n & = & \sqcap\{\,[\![p]\!]^* \,|\, p \in \mathbf{CIP}[start(fg(n)), n)\,\} \\
(m \in \mathbf{N}^S_C) & \sqsubseteq & \sqcap\{\,[\![m_R]\!]^* \circ [\![p']\!]^* \circ [\![m_C]\!]^* \circ [\![p]\!]^* \,|\, p' \in \mathbf{CIP}[\mathbf{s}', \mathbf{e}'], \\
& & \qquad p \in \mathbf{CIP}[start(fg(m)), m)\,\} \\
(\mathrm{IH}) & \sqsubseteq & [\![m_R]\!]^* \circ gtr^l[\mathbf{e}'] \circ [\![m_C]\!]^* \circ gtr^l[m] \\
(\mathrm{Prop.\ 5.2(3)}) & = & ltr^l[m] \circ gtr^l[m] \;=\; f
\end{array}$$

Thus together with the induction hypothesis this yields:

$$imop_n \sqsubseteq gtr^{k-1}[n] \sqcap f \;=\; gtr^k[n].$$

Also the remaining inclusion

(2) $[\![n']\!] \sqsubseteq ltr^k[n'],\; n' \in \mathbf{N}^S$

requires the investigation of two cases, depending on the nature of the node $n \in \mathbf{N}^S$ being processed in the k-th iteration:

Case 1. $n \notin \{\mathbf{e}_0, .., \mathbf{e}_k\}$. This guarantees that ltr remains unchanged in the k-th iteration of the while-loop. Thus $[\![n']\!] \sqsubseteq ltr^k[n'],\; n' \in \mathbf{N}^S$, is a direct consequence of the induction hypothesis.

Case 2. $n \in \{\mathbf{e}_0, .., \mathbf{e}_k\}$. In this case we have

$$ltr^k[l] = \begin{cases} [\![l_R]\!]^* \circ gtr^k[n] \circ [\![l_C]\!]^* & \text{if } l \in calls(fg(n)) \\ ltr^{k-1}[l] & \text{otherwise} \end{cases}$$

Thus $[\![n']\!] \sqsubseteq ltr^k[n']$ is a direct consequence of the already proved $imop_n \sqsubseteq gtr^k[n]$, Definition 4.5 and the induction hypothesis. □

As a consequence of this lemma we have for arbitrary $n \in \mathbf{N}^S_C$, where $G' =_{df} decl(n)$:

$$\begin{array}{lll}
\sqcap\{\,[\![p]\!]^* | p \in \mathbf{CIP}[n_C, n_R]\,\} & = & [\![n_R]\!]^* \circ \sqcap\{\,[\![p]\!]^* \,|\, p \in \mathbf{CIP}[start(G'), end(G')]\,\} \circ [\![n_C]\!]^* \\
(\mathrm{Lemma\ 6.2(2)}) & = & [\![n_R]\!]^* \circ [\![end(G')]\!] \circ [\![n_C]\!]^* \\
(\;(\mathrm{Lemma\ 6.2(1)}) & \sqsupseteq & [\![n_R]\!]^* \circ [\![end(G')]\!] \circ [\![n_C]\!]^* \;) \\
(\mathrm{Definition\ 4.5}) & = & [\![n]\!]
\end{array}$$

This directly implies the Main Lemma 6.4[18], which allows us to prove our main results (cf. Section 7) in a straightforward fashion almost as in the intraprocedural case (cf. [Ki]).

Lemma 6.4 (The Main Lemma)
For all $n \in \mathbf{N}^S_C$, we have, if the semantic functions $[\![m]\!]^*$, $m \in N^*$, are

1. *s-monotonic:* $[\![n]\!] \sqsubseteq \sqcap\{\,[\![p]\!]^* \,|\, p \in \mathbf{CIP}[n_C, n_R]\,\}$
2. *s-distributive:* $[\![n]\!] = \sqcap\{\,[\![p]\!]^* \,|\, p \in \mathbf{CIP}[n_C, n_R]\,\}$

[17] Remember that $[\![\mathbf{e}']\!] = id_{STACK}$.
[18] Remember that $[\![end(G')]\!] = id_{STACK}$.

7 Main Results

As a first result, we have as in the intraprocedural case that the *IMFP*-solution is a correct approximation of the *IMOP*-solution, whenever all the local abstract semantic functions are s-monotonic:

Theorem 7.1 (Interprocedural Safety Theorem)
Given a flow graph system $S = \{G_0, G_1, ., G_k\}$ and its corresponding interprocedural flow graph $G^* = (N^*, E^*, \mathbf{s}^*, \mathbf{e}^*)$, the IMFP-solution is a correct approximation of the IMOP-solution, i.e. $\forall\, n \in \mathbf{N}^S\; \forall\, c_0 \in \mathcal{C}.\; IMFP_{c_0}(n) \sqsubseteq IMOP_{c_0}(n)$, if the abstract semantics $[\![n]\!]^*$ of all nodes $n \in N^*$ is given by a s-monotonic function.

Proof. We (equivalently) show for all $c_0 \in \mathcal{C}$:

$(*)\quad \forall\, n \in \mathbf{N}^S\; \forall\, p \in \mathbf{IP}[\mathbf{s}^*, n).\; IMFP_{c_0}(n) \sqsubseteq [\![p]\!]^*(newstack(c_0))$

by induction on the length k of path p. The case $k = 0$ is an immediate consequence of

$$[\![p]\!]^*(newstack(c_0)) \;=\; [\![\varepsilon]\!]^*(newstack(c_0)) \;=\; newstack(c_0) \;=\; \mathbf{pre}_{c_0}(\mathbf{s}_0)$$

Thus let $k>0$ and assume that $(*)$ holds for all paths p with $lgth(p)<k$, i.e.

(IH) $\quad \forall\, n \in \mathbf{N}^S\; (\forall\, p \in \mathbf{IP}[\mathbf{s}^*, n).\, 0 \le lgth(p)<k).\; IMFP_{c_0}(n) \sqsubseteq [\![p]\!]^*(newstack(c_0))$

Now, given $n \in \mathbf{N}^S$, it is sufficient to show for any path $p \in \mathbf{IP}[\mathbf{s}^*, n)$ with $lgth(p) = k$:

$$IMFP_{c_0}(n) \sqsubseteq [\![p]\!]^*(newstack(c_0))$$

Thus we can assume that there is such a path p, which then obviously can be written as $p = p';(m)$ for some $m \in pred^*(n)$ and $p' \in \mathbf{IP}[\mathbf{s}^*, m)$. Moreover, we have by induction hypothesis:

$$IMFP_{c_0}(m) \sqsubseteq [\![p']\!]^*(newstack(c_0))$$

$(*)$ now follows by investigating three cases:

Case 1. $m \in N^* \backslash (N^*_C \cup N^*_R)$
Here we straightforwardly succeed by considering the following inclusions that result from Equation System 4.8:

$$\begin{array}{lll}
IMFP_{c_0}(n) & = & \sqcap\{\,\mathbf{post}_{c_0}(l) \,|\, l \in pred_{fg(n)}(n)\,\} \\
& \sqsubseteq & \mathbf{post}_{c_0}(m) \\
(m \notin \mathbf{N}^S_C) & = & [\![m]\!]^*(IMFP_{c_0}(m)) \\
(\mathrm{IH,\ s\text{-}monotonicity}) & \sqsubseteq & [\![m]\!]^*([\![p']\!]^*(newstack(c_0))) \\
& = & [\![p]\!]^*(newstack(c_0))
\end{array}$$

Case 2. $m \in N^*_C$
In this case, we have $n \in \{\mathbf{s}_1, .., \mathbf{s}_k\}$ and the existence of a node $\bar{m} \in \mathbf{N}^S_C$ corresponding to $m \equiv \bar{m}_C$ in S. Thus we obtain as above:

$$\begin{array}{lll}
IMFP_{c_0}(n) & = & \sqcap\{\,[\![l_C]\!]^*(\mathbf{pre}_{c_0}(l)) \,|\, l \in calls(fg(n))\} \\
& \sqsubseteq & [\![\bar{m}_C]\!]^*(\mathbf{pre}_{c_0}(\bar{m})) \\
& = & [\![\bar{m}_C]\!]^*(IMFP_{c_0}(\bar{m})) \\
(\mathrm{IH,\ s\text{-}monotonicity}) & \sqsubseteq & [\![\bar{m}_C]\!]^*([\![p']\!]^*(newstack(c_0))) \\
& = & [\![p]\!]^*(newstack(c_0))
\end{array}$$

Case 3. $m \in N_R^*$
Here, there exists a predecessor $\bar{m} \in \mathbf{N}_C^S$ of n in S which corresponds to m, i.e. $\bar{m}_R \equiv m$. Now considering the following decomposition of path p:

$$p = \bar{p}; (\bar{m}_C); (start(decl(\bar{m})), ..., end(decl(\bar{m}))); (\bar{m}_R)$$

and identifying $\bar{m}$ and $\bar{m}_C$, the induction hypothesis yields[19]:

$$IMFP_{c_0}(\bar{m}_C) \sqsubseteq [\![\bar{p}]\!]^*(newstack(c_0))$$

and therefore by means of Equation System 4.8 as desired:

$$\begin{array}{lcl}
IMFP_{c_0}(n) & = & \bigsqcap \{ \mathbf{post}_{c_0}(l) \mid l \in pred_{fg(n)}(n) \} \\
 & \sqsubseteq & \mathbf{post}_{c_0}(\bar{m}) \\
 & = & [\![\bar{m}]\!](IMFP_{c_0}(\bar{m}_C)) \\
\text{(IH, s-monot.)} & \sqsubseteq & [\![\bar{m}]\!]([\![\bar{p}]\!]^*(newstack(c_0))) \\
\text{(M.L. 6.4(1))} & \sqsubseteq & \bigsqcap \{ [\![p']\!]^*([\![\bar{p}]\!]^*(newstack(c_0))) \mid p' \in \mathbf{CIP}[\bar{m}_C, \bar{m}_R] \} \\
\text{(Lemma 3.8)} & \sqsubseteq & [\![\bar{m}_R]\!]^*([\![(start(G'), ., end(G'))]\!]^*([\![\bar{m}_C]\!]^*([\![\bar{p}]\!]^*(newstack(c_0)))))
\end{array}$$

where $G' =_{df} decl(\bar{m})$. This completes the proof of the theorem. □

Again, as in the intraprocedural case, s-distributivity of the local semantic functions yields optimality (or completeness):

Theorem 7.2 (Interprocedural Coincidence Theorem)
Given a flow graph system $S = \{G_0, G_1, ., G_k\}$ *and its corresponding interprocedural flow graph* $G^* = (N^*, E^*, \mathbf{s}^*, \mathbf{e}^*)$*, the IMFP-solution and the IMOP-solution coincide, i.e.* $\forall\, n \in \mathbf{N}^S\ \forall\, c_0 \in \mathcal{C}.\ IMOP_{c_0}(n) = IMFP_{c_0}(n)$*, if the abstract semantics* $[\![n]\!]^*$ *of all nodes* $n \in N^*$ *is given by a s-distributive function.*

Proof. The first inclusion, "$\sqsupseteq$", is a consequence of the Interprocedural Safety Theorem 7.1. In order to prove the second inclusion, "$\sqsubseteq$", we (equivalently) show (cf. Theorem 5.6):

$$(*) \quad \forall\, k \geq 0\ \forall\, n \in \mathbf{N}^S.\ IMOP_{c_0}(n) \sqsubseteq pre^k[n]$$

by induction on k. The case $k = 0$ is trivial according to the initialization of $pre[n]$, $n \in \mathbf{N}_C^S$, with $newstack(\top)$, and $k = 1$ is dealt with by means of:

$$IMOP_{c_0}(\mathbf{s}^*) \sqsubseteq [\![\varepsilon]\!]^*(newstack(c_0)) = newstack(c_0) = pre[\mathbf{s}_0]$$

Thus let $k>1$, and let us consider the k-th iteration of the while-loop under the induction hypothesis (IH) that $(*)$ holds before the k-th execution of the while-loop. Then we only need to consider the node $n \in \mathbf{N}^S$, which belongs to the workset entry (n, stk) being processed in the k-th iteration, because the information associated with all the other nodes remains unchanged in this iteration and therefore satisfies $(*)$ according to

[19] The situation is illustrated in Figure 6.3.

(IH). Now $k>1$ yields the existence of a node $m \in \mathbf{N}^S$, whose execution in the l-th iteration caused (n, stk) to be added to the workset, and we have:

$$stk = \begin{cases} [\![m_C]\!]^*(pre^l[m]) & \text{if } n \in \{\mathbf{s}_0, ..., \mathbf{s}_k\} \\ [\![m]\!](pre^l[m]) & \text{otherwise} \end{cases}$$

Now we prove $IMOP_{c_0}(n) \sqsubseteq stk$ by investigating three cases:

Case 1. $m \in pred_{fg(n)}(n) \backslash \mathbf{N}_C^S$
Here we succeed straightforwardly:

$$\begin{array}{lcl}
IMOP_{c_0}(n) & = & \bigsqcap \{ [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, n) \} \\
(m \in pred_{fg(n)}(n) \backslash \mathbf{N}_C^S) & \sqsubseteq & \bigsqcap \{ [\![m]\!]^*([\![p]\!]^*(newstack(c_0))) \mid p \in \mathbf{IP}[\mathbf{s}^*, m) \} \\
\text{(s-distributivity)} & = & [\![m]\!]^*(\bigsqcap \{ [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, m) \}) \\
\text{(Definition of } IMOP\text{)} & = & [\![m]\!]^*(IMOP_{c_0}(m)) \\
(m \notin \mathbf{N}_C^S) & = & [\![m]\!](IMOP_{c_0}(m)) \\
\text{(IH, monotonicity)} & \sqsubseteq & [\![m]\!](pre^l[m]) = stk
\end{array}$$

Case 2. $m \in pred_{fg(n)}(n) \cap \mathbf{N}_C^S$
Identifying the nodes m and m_C, we obtain as promised:

$$\begin{array}{lcl}
IMOP_{c_0}(n) & = & \bigsqcap \{ [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, n) \} \\
 & \sqsubseteq & \bigsqcap \{ [\![\bar{p}]\!]^*([\![p]\!]^*(newstack(c_0))) \mid p \in \mathbf{IP}[\mathbf{s}^*, m) \ \wedge \\
 & & \qquad \bar{p} \in \mathbf{CIP}[m_C, m_R] \} \\
\text{(s-distributivity)} & = & \bigsqcap \{ [\![\bar{p}]\!]^*(\bigsqcap \{ [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, m) \}) \mid \\
 & & \qquad \bar{p} \in \mathbf{CIP}[m_C, m_R] \} \\
\text{(Definition of } IMOP\text{)} & = & \bigsqcap \{ [\![\bar{p}]\!]^*(IMOP_{c_0}(m)) \mid \bar{p} \in \mathbf{CIP}[m_C, m_R] \} \\
\text{(Main Lemma 6.4(2))} & = & [\![m]\!](IMOP_{c_0}(m)) \\
\text{(IH, monotonicity)} & \sqsubseteq & [\![m]\!](pre^l[m]) = stk
\end{array}$$

Case 3. $m \notin pred_{fg(n)}(n)$
In this case we have $m \in calls(fg(n))$ and $n \in \{\mathbf{s}_0, ..., \mathbf{s}_k\}$, and therefore by identifying the nodes m and m_C as desired:

$$\begin{array}{lcl}
IMOP_{c_0}(n) & = & \bigsqcap \{ [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, n) \} \\
 & \sqsubseteq & \bigsqcap \{ [\![m_C]\!]^*([\![p]\!]^*(newstack(c_0))) \mid p \in \mathbf{IP}[\mathbf{s}^*, m_C) \} \\
\text{(s-distributivity)} & = & [\![m_C]\!]^*(\bigsqcap \{ [\![p]\!]^*(newstack(c_0)) \mid p \in \mathbf{IP}[\mathbf{s}^*, m_C) \}) \\
\text{(Definition of } IMOP\text{)} & = & [\![m_C]\!]^*(IMOP_{c_0}(m)) \\
\text{(IH, s-monotonicity)} & \sqsubseteq & [\![m_C]\!]^*(pre^l[m]) = stk
\end{array}$$

Thus together with the induction hypothesis, we uniformly have in all three cases as desired: $IMOP_{c_0}(n) \sqsubseteq pre^{k-1}[n] \sqcap stk = pre^k[n]$. □

Note that Lemma 4.4 allows to check the s-monotonicity or s-distributivity of the semantic functions $[\![n]\!]^*$ simply by checking these properties for the semantic functions $[\![n]\!]'$ and the reduction functions $\mathcal{R}_n$. Thus the only additional effort in comparison to the intraprocedural case arises from checking the reduction functions.

8 Applications

In this section we sketch three applications of the Interprocedural Coincidence Theorem 7.2. We omit details here, since all examples require their own setup.

In [SK2] we propose an algorithm for ***interprocedural constant propagation*** and ***constant folding***, which generalizes and improves all previous techniques for interprocedural constant propagation (cf. [CC2,CCKT,JM]). This algorithm determines all ***finite interprocedural constants***, which are the interprocedural analogue to the set of finite constants introduced in [SK1]. As in the intraprocedural case, finite interprocedural constants have a purely ***operational*** characterization in the sense of the *IMOP*-strategy, and a purely ***denotational*** characterization in the sense of the *IMFP*-strategy. The Interprocedural Coincidence Theorem 7.2 yields the equivalence of these characterizations.

The second example concerns the interprocedural versions of the classical bit-vector data flow analyses, e.g. determining available expressions, reaching definitions, live variables, very busy (anticipatable) expressions (cf. [He])[20], and, more sophisticatedly, the elimination of interprocedural partial redundancies[21]. In all these cases, the Interprocedural Coincidence Theorem 7.2 allows us to prove the optimality of our algorithms for programs with recursive procedures, local variables and formal parameters [KS2].

As a third example we consider the interprocedural extension of the semantic code motion algorithms proposed in [SKR1,SKR2]. Here the Interprocedural Coincidence Theorem 7.2 is important to show the completeness of the underlying data flow analysis algorithm, i.e. that the algorithm determines ***all*** equivalences between program terms with respect to the Herbrand interpretation. This can be done along the lines proposed in [SK2].

9 Conclusions

We have presented an interprocedural generalization of the well-known intraprocedural Coincidence Theorem of Kam and Ullman [KU], which covers arbitrary programs with recursive procedures, formal parameters and local variables. Our theorem, which reduces to the classical intraprocedural version in the absence of procedures, delivers a sufficient condition for the coincidence of the ***interprocedural meet over all paths*** strategy and the ***interprocedural maximal fixed point*** strategy, and it generalizes previous results (cf. [Ba1,Ba2,SP]), which do not deal properly with local variables of recursive procedures.

Our results are formulated within the framework of abstract interpretation, thus covering a wide range of data flow analyses. This has been illustrated by means of some concrete examples of practical relevance.

[20] This application has also been suggested (without giving any details) by Sharir and Pnueli [SP].
[21] This problem is heuristically dealt with in [Mo,MR].

Acknowledgements

We would like to thank Olaf Burkart, Marion Dreimüller, and Oliver Rüthing for their careful proof reading.

References

[All] F. E. Allen. *"Control Flow Analysis"*. ACM Sigplan Notices, July 1970

[Ba1] G. Barth. *"Interprozedurale Datenflußsysteme"*. Habilitationsschrift, University of Kaiserslautern, Germany, 1981

[Ba2] G. Barth. *"Interprocedural Data Flow Systems"*. 6^{th} GI-Conference, Dortmund, Germany, LNCS 145, 49 - 59, 1983

[CC1] P. Cousot and R. Cousot. *"Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints"*. 4^{th} POPL, Los Angeles, California, 238 - 252, 1977

[CC2] P. Cousot and R. Cousot. *"Static Determination of Dynamic Properties of Recursive Procedures"*. In: E. Neuhold (Ed.), 2^{nd} Proceedings of the IFIP Working Conference on Formal Description of Programming Concepts, St. Andrews, N.B., Canada, 1977

[CCKT] D. Callahan, K. D. Cooper, K. W. Kennedy and L. M. Torczon. *"Interprocedural Constant Propagation"*. Proc. SIGPLAN'86 Symp. on Compiler Construction, SIGPLAN Notices, Vol. 21, No. 7, 152 - 161, 1986

[Gu] J. Guttag. *"Abstract Data Types and the Development of Data Structures"*. CACM 20(6):396-404, 1977

[He] M. S. Hecht. *"Flow Analysis of Computer Programs"*. Elsevier, North-Holland, 1977

[JM] N. D. Jones and S. S. Muchnick. *"A Flexible Approach to Interprocedural Data Flow Analysis and Programs with Recursive Data Structures"*. 9^{th} POPL, Albuquerque, New Mexico, 1982

[Ki] G. A. Kildall. *"A Unified Approach to Global Program Optimization"*. 1^{st} POPL, Boston, Massachusetts, 194 - 206, 1973

[KS2] J. Knoop and B. Steffen. *"Efficient and Optimal Interprocedural Bit-Vector Data Flow Analyses: A Uniform Interprocedural Framework"*. Submitted to ICCL'92

[KU] J. B. Kam and J. D. Ullman. *"Monotone Data Flow Analysis Frameworks"*. Acta Informatica 7, 309 - 317, 1977

[Mo] E. Morel. *"Data Flow Analysis and Global Optimization"*. In: B. Lorho (Ed.). "Methods and Tools for Compiler Construction", Cambridge University Press, 1984

[My] E. W. Myers. *"A Precise Inter-Procedural Data Flow Algorithm"*. 8^{th} POPL, Williamsburg, Virginia, 219 - 230, 1981

[MR] E. Morel and C. Renvoise. *"Interprocedural Elimination of Partial Redundancies"*. In: St. S. Muchnick and N. D. Jones (Eds.). "Program Flow Analysis: Theory and Applications", Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981

[MJ] St. S. Muchnick and N. D. Jones (Eds.). "Program Flow Analysis: Theory and Applications", Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981

[Ro] B. K. Rosen. "Data Flow Analysis for Procedural Languages". Journal of the ACM, Vol. 26, No. 2, 322 - 344, 1979

[SK1] B. Steffen and J. Knoop. *"Finite Constants: Characterizations of a New Decidable Set of Constants"*. 14^{th} MFCS, Porąbka-Kozubnik, Poland, LNCS 379, 481 - 491, 1989 – An extended version appeared in: TCS 80(1), 303 - 318, 1991

[SK2] B. Steffen and J. Knoop. *"Finite Interprocedural Constants"*. Technical Report, to appear

[SKR1] B. Steffen, J. Knoop and O. Rüthing. *"The Value Flow Graph: A Program Representation for Optimal Program Transformations"*. 3^{rd} ESOP, Copenhagen, Denmark, LNCS 432, 389 - 405, 1990

[SKR2] B. Steffen, J. Knoop and O. Rüthing. *"Efficient Code Motion and an Adaption to Strength Reduction"*. 4^{th} TAPSOFT, Brighton, United Kingdom, LNCS 494, 394 - 415, 1991

[SP] M. Sharir and A. Pnueli. *"Two Approaches to Interprocedural Data Flow Analysis"*. In: St. S. Muchnick and N. D. Jones (Eds.). "Program Flow Analysis: Theory and Applications", Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981

Aachener Informatik-Berichte

Please write to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, W-5100 Aachen

87-1 Jahresbericht 1986
87-2 D. de Frutos Escrig / K. Indermark: Equivalence relations of non-deterministic Ianov-schemes
87-3 M. Nagl: A Software Development Environment Based on Graph Technology
87-4 C. Lewerentz / M. Nagl / B. Westfechtel: On Integration Mechanisms within a Graph-based Software Development Environment
87-5 R. Rinn: Über Eingabeanomalien bei verschiedenen Inferenzmodellen
87-6 W. Damm / G. Döhmen: Specifying Distributed Computer Architectures in AADL*
87-7 G. Engels / C. Lewerentz / W. Schäfer: Graph Grammar Engineering: A Software Specification Method
87-8 M. Nagl: Set Theoretic Approaches to Graph Grammars
87-9 C. Lewerentz / A. Schürr: Experiences with a Database System for Software Documents
87-10 H. Klaeren / K. Indermark: A New Implementation Technique for Recursive Function Definitions
87-11 R. Loogen: Design of a parallel programmable graph reduction machine with distributed memory
87-12 J. Börstler / U. Möncke / R. Wilhelm: Table Compression for Tree Automata
87-13 C. Lewerentz: Extended Programming in the Large in a Software Development Environment
87-14 G. Engels / Th. Janning / W. Schäfer: A Highly Integrated Tool Set for Program Development Support
87-15 R. Loogen / U. Goltz: A non-interleaving semantic model for nondeterministic concurrent processes
87-16 Th. Hafer: On the Boolean Closure of Büchi Tree Automaton Definable Sets of w-Trees.
87-17 U. Heuter: First-Order Properties of Trees, Star-Free Expressions and Aperiodicity
88-1 G. Esser / J. Rückert / F. Wagner (Hrsg.): Gesellschaftliche Aspekte der Informatik
88-2 P. Martini / O. Spaniol: Token-Passing in High-Speed Backbone Networks for Campus-Wide Environments
88-3 Th. Welzel: Simulation of a Multiple Token Ring Backbone
88-4 P. Martini: Performance Comparison for HSLAN Media Access Protocols
88-5 P. Martini: Performance Analysis of Multiple Token Rings
88-6 A. Mann / J. Rückert / O. Spaniol: Datenfunknetze
88-7 A. Mann / J. Rückert: Packet Radio Networks for Data Exchange

88-8 A. Mann / J. Rückert: Concurrent Slot Assignment Protocol for Packet Radio Networks
88-9 W. Kremer / F. Reichert / J. Rückert / A. Mann: Entwurf einer Netzwerktopologie für ein Mobilfunknetz zur Unterstützung des öffentlichen Straßenverkehrs
88-10 K. Jakobs: Towards User-Friendly Networking
88-11 K. Jakobs: The Directory - Evolution of a Standard
88-12 K. Jakobs: Directory Services in Distributed Systems - A Survey
88-13 M. Schümmer: RS-511, a Protocol for the Plant Floor
88-14 U. Quernheim: Satellite Communication Protocols - A Performance Comparison Considering On-Board Processing
88-15 P. Martini / O. Spaniol / Th. Welzel: File Transfer in High Speed Token Ring Networks: Performance Evaluation by Approximate Analysis and Simulation
88-16 Jahresbericht 1987
88-17 W. Thomas: Automata on Infinite Objects
88-18 M. Sonnenschein: On Petri Nets and Data Flow Graphs
88-19 H. Vogler: Functional Description of the Contextual Analysis in Block-Structured Programming Languages: A Case Study of Tree Transducers
88-20 Th. Welzel: Einsatz des Simulationswerkzeuges QNAP2 zur Leistungsbewertung von Kommunikationsprotokollen
88-21 Th. Janning / C. Lewerentz: Integrated Project Team Management in a Software Development Environment
88-22 J. Engelfriet / H. Vogler: Modular Tree Transducers
88-23 W. Thomas: Automata and Quantifier Hierarchies
88-24 U. Heuter: Generalized Definite Tree Languages
89-1 Jahresbericht 1988
89-2 G. Esser / J. Rückert / F. Wagner (Hrsg.): Gesellschaftliche Aspekte der Informatik
89-3 H. Vogler: Bottom-Up Computation of Primitive Recursive Tree Functions
89-4 A. Schürr: Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language
89-5 J. Börstler: Wiederverwendbarkeit und Softwareentwicklung - Probleme, Lösungsansätze und Bibliographie
89-6 K. Jakobs: OSI - An Appropriate Basis for Group Communication?
89-7 K. Jakobs: ISO's Directory Proposal Evolution, Current Status and Future Problems
89-8 B. Westfechtel: Extension of a Graph Storage for Software Documents with Primitives for Undo / Redo and Revision Control
89-9 P. Martini: High Speed Local Area Networks - A Tutorial
89-00 Jozef Filser: Bolidische Gedangen (Gabidel 5): "Ieber die Kombjuderkunzt"
89-10 P. Davids / Th. Welzel: Performance Analysis of DQDB Based on Simulation
89-11 M. Nagl (Ed.): Abstracts of Talks presented at the WG '89 15th International Workshop on Graphtheoretic Concepts in Computer Science, Rolduc (NL) near Aachen (D), June 14-16, 1989.

89-12 P. Martini: The DQDB Protocol - Is it Playing the Game?
89-13 M. Schümmer: CNC / DNC Communication with MAP
89-14 M. Schümmer: Local Area Networks for Manufactoring Environments with hard Real-Time Requirements
89-15 M. Schümmer / Th. Welzel / P. Martini: Integration of Field Bus and MAP Networks - Hierarchical Communication Systems in Production Environments
89-16 G. Vossen / K.-U. Witt: SUXESS: Towards a Sound Unification of Extensions of the Relational Data Model
89-17 J. Derissen / P. Hruschka / M.v.d.Beeck / Th. Janning / M. Nagl: Integrating Structured Analysis and Information Modelling
89-18 A. Maaßen: Programming with Higher Order Functions
89-19 M. Rodriguez-Artalejo / H. Vogler: Note on A Narrowing Machine for Syntax Directed BABEL
89-20 H. Kuchen / R. Loogen / J. J. Moreno-Navarro / M. Rodriguez-Artalejo Graph-based Implementation of a Functional Logic Language
89-i4 O. Spaniol: Natürliche Intelligenz oder: Der Parallel-Igel
89-21 G. Vossen / S. Wang: Correct and Efficient Deadlock Detection and Resolution in Distributed Database Systems
89-22 G. Vossen / R. Schwarz: Dynamic Scheduling of Transactions in Design Databases based on Version Consistency
90-1 Jahresbericht 1989
90-2 V. Jansen / A. Potthoff / W. Thomas / U. Wermuth: A Short Guide to the AMORE System (computing Automata, MOnoids, and Regular Expressions)
90-3 J. Skurczynski: On Three Hierarchies of Weak SkS Formulas
90-4 R. Loogen: Stack-based Implementation of Narrowing
90-5 H. Kuchen / A. Wagener: Comparison of Dynamic Load Balancing Strategies
90-6 K. Jakobs / F. Reichert: Directory Services for Mobile Communication
90-7 K. Jakobs: What's Beyond the Interface-OSI Networks to Support CooperativeWork
90-8 K. Jakobs: Directory Names and Schema - An Evaluation
90-9 U. Quernheim / D. Kreuer: Das CCITT - Signalisierungssystem Nr. 7 auf Satellitenstrecken; Simulation der Zeichengabestrecke
90-10 U. Quernheim / St. Jacobs / D. Kreuer: Adaptive ARQ-Fehlersicherungsverfahren für die Point-to-Multipoint-Satellitenkommunikation
90-11 J.J.Moreno-Navarro / H. Kuchen / R. Loogen / M. Rodriguez-Artalejo: Lazy Nar-rowing in a Graph Machine
90-12 K. Jakobs / J. Kaltwasser / F. Reichert / O. Spaniol: Der Computer fährt mit
90-13 R. Mathar / A. Mann: Analyzing a Distributed Slot Assignment Protocol by Markov Chains
90-14 A. Maaßen: Compilerentwicklung in Miranda - ein Praktikum in funktionaler Programmierung
90-15 M. Nagl / A. Schürr: A Specification Environment for Graph Grammars
90-16 A. Schürr: PROGRESS: a VHL-Language Based on Graph Grammars

90-17 M. Möller: Ein Ebenenmodell wissensbasierter Konsultationen - Unterstützung für Wissensakquisition und Erklärungsfähigkeit

90-18 E. Kowalewski: Entwurf und Interpretation einer Sprache zur Beschreibung von Konsultationsphasen in Expertensystemen

90-19 F. Strauß: Erklärungsfähigkeit eines ebenenorientierten Expertensystemmodells

90-20 Y. Ortega-Mallen / D. de Frutos-Escrig: A Complete Proof System for Timed Observations

90-21 M. Nagl: Modelling of Software Architectures: Importance, Notions, Experiences

90-22 H. Faßbender / H. Vogler: A Call-by-need Implementation of Syntax Directed Functional Programming

90-23 Th. Janning / M. Lefering: A Transformation from Requirements Engineering into Design - The Method and the Tool

90-24 J. Noack: Natürlichsprachliche Interaktion mit Datenbanken

90-25 R. Mathar: Concurrence Probabilities for a Locally Slotted Packet Radio Network by Combinatorial Methods

90-26 F. Herrmann / Heiko Vogler: Primitive Recursive Programs for Defining Tree Functions

90-27 Z. Fülöp / F. Herrmann / S. Vagvölgyi / H. Vogler: Tree Transducers with External Functions

90-28 P. Bachmann / W. Oberschelp / B. Thalheim / G. Vossen: The Design of RAD: Towards an Interactive Toolbox for Database Design

91-1 Jahresbericht 1990

91-2 F. Beuting / F. Piron / K. Rausch: Ein neuronales Modell zur Feststellung von für die visuelle Aufmerksamkeit relevanten Figureneigenschaften

91-3 B. Steffen / A. Ingolfsdottir: Characteristic Formulae for Processes with Divergence

91-4 M. Portz: A new class of cryptosystems based on interconnection networks

91-5 H. Kuchen / G. Geiler: Distributed Applicative Arrays

91-6 L. Staiger: Kolmogorov Complexity and Hausdorff Dimension

91-7 L. Staiger: Syntactic congruences for ω-languages

91-8 K. Jakobs / K. D. Kleefeld: Using Public Communication Services for Archaeological Applications

91-9 E. Kuikka: A Proposal for a Syntax-Directed Text Processing System

91-10 K. Gladitz / H. Faßbender / H. Vogler: Compiler-based Implementation of Syntax-Directed Functional Programming

91-11 R. Loogen / S. Winkler: Dynamic Detection of Determinism in Functional Logic Languages

91-12 K. Indermark / M. Rodríguez Artalejo (Eds.): Granada Workshop on the Integration of Functional and Logic Programming

91-13 R. Hager / W. Kremer: The Adaptive Priority Scheduler: A More Fair Priority Service Discipline.

91-14 A. Fasbender / W. Kremer: A New Approximation Algorithm for Tandem Networks with Priority Nodes.

91-15 J. Börstler / A. Zündorf: Revisiting extensions to Modula-2 to improve reusability.

91-16 J. Börstler / T. Janning: Bridging the gap between Requirements Analysis and Design

91-17 A. Zündorf / A. Schürr: Nondeterministic Control Structures for Graph Rewriting Systems

91-18 M. Jarke / J. Mylopoulos / J. W. Schmidt / Y. Vassiliou: DAIDA: An Environment for Evolving Information Systems

91-19 M. Jeusfeld / M. Jarke: From Relational to Object-Oriented Integrity Simplification

91-20 G. Hogen / A. Kindler / R. Loogen: Automatic Parallelization of Lazy Functional Programs

91-21 O. Spaniol: ODP (Open Distributed Processing): Yet another Viewpoint

91-22 H. Kuchen / F. Lücking / H. Stoltze: The Topology Description Language TDL

91-23 S. Graf / B. Steffen: Compositional Minimization of Finite State Systems

91-24 R. Cleaveland / J. Parrow / B. Steffen: The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems

91-25 R. Mathar/J. Mattfeldt: Optimal Transmission Ranges for Mobile Communication in Linear Multihop Packet Radio Networks

91-26 M. Jeusfeld/ M. Staudt: Query Optimization in Deduktive Object Bases

91-27 J. Knoop/B. Steffen: The Interprocedural Coincidence Theorem