Fuzzy Array Dataflow Analysis

Denis Barthou , Jean-François Collard , Paul Feautrier*
PRiSM Laboratory Université de Versailles
45 Avenue des Etats-Unis F-78035 Versailles Cedex

Abstract

Dataflow analyses track the definitions and uses of variable values, and are useful to optimizing and parallelizing compilers. Such analyses compute, for every (array cell) value read in a right-hand-side expression, the very operation which produced it. These analyses, however, make quite stringent hypotheses on the input programs: the control flow should be known at compile-time (i.e., static), and array subscripts must be affine functions of surrounding counters and possibly of symbolic constants. On the contrary, the analysis presented in this report handles general if's and while loops, and general non-affine array subscripts.
Keywords: Dataflow, non-affine subscripts, while-loop, if-then-else, dynamic control

1 Introduction

Whereas processor and interconnection network technologies make giant leaps nearly every couple of years, the corresponding software technology lags far behind. In particular, comparatively few parallelizing compilers are used in production environments. This is partly due to the difficulty for the compiler to find in the source program the information it needs to exhibit parallelism and optimize code generation.

Vectorization and parallelization methods are mainly based on the parallelism generated by independent references to distinct parts of arrays. Various dependence tests have been proposed [1]. However, most of these tests are not exact, and, even when they are, cannot distinguish between true dependences, which describe a real information flow, and spurious dependences, in which the value purported to be transmitted is destroyed before being used. To obviate this difficulty, methods have been designed to compute, for every array cell value read in a right-hand-side expression (the "sink"), the very operation which produced it (the "source"). These methods are called *Array Dataflow Analyses* (ADA) [10, 14], or *Value-Based Dependence Analyses* [15]. These ADAs, however, make quite stringent hypotheses on the input programs. The only tractable control structures are the `do` loop and the sequence; loop counters' bounds and array subscripts must be affine functions of surrounding counters and possibly of symbolic constants, the *structure parameters*. Programs following this model have been called "static control programs" in [10]. The same paper has shown that an exact ADA can be mechanically performed on static control programs.

Obviously, there is a continuum of analyses between the detection of simple dependences and full-fledged ADA. These analyses are often designed for a special purpose (e.g., array privatization) and may need less precise information than ADA. The consequence is that they can be applied to less constrained programs.

The present paper deals with general control structures, such as `if`s and `while` loops, and with unrestricted array subscripts. Notice that we assume that unstructured programs are preprocessed, and that for instance "backward" `goto`s are first converted into `while`s. However, with such unpredictable, dynamic control structures, no exact information can be hoped for in general. Hence, the aim of this paper is threefold. First, we aim at showing that even partial information can be automatically gathered by *Fuzzy Array Dataflow Analysis* (FADA). This paper extends our previous work [6] on FADA to general, non-affine array subscripts. The second purpose of this paper is to formalize and generalize these previous proposals and to prove general results. Third, we will show that the precise, classical ADA is a special case of FADA.

*[Denis.Barthou,Jean-Francois.Collard,Paul.Feautrier]@prism.uvsq.fr

1.1 Program model

In this paper, our aim is to extend the scope of array dataflow analysis to programs respecting the following constraints:

1. The only data structures are of base types (integers, reals, etc.) and arrays thereof.
2. The only control structures are the sequence, the `do` loop, the `while` loop[1], and the `if..then..else` construct. `goto`s and procedure calls are forbidden.
3. Basic statements are assignments to scalars or array elements.
4. No pointer, `EQUIVALENCE` or aliasing is allowed.

Non-linear constraints are equations or inequalities which depend on variables other than loop counters and structure parameters, and/or are non-linearly dependent on loop counters and structure parameters. For example, non-linear constraints may come from predicates of `if` or `while` constructs or from array subscripts. Obviously, some non-linear constraints can be removed by replacing some variables by their expression in terms of loop counters and structure parameters (induction variable detection and forward substitution). Similarly, some `while` loops can be transformed into `do` loops. We will suppose here that these simplifications have been performed, when possible, by a previous phase of the compiler.

1.2 Notations

The k-th entry of vector $\vec{x}$ is denoted by $\vec{x}[k]$ or $\vec{x}_k$. The dimension of a given vector $\vec{x}$ is denoted by $|\vec{x}|$. The sub-vector built from components k to l is written as: $\vec{x}[k..l]$. If $k > l$, then this vector is by convention the vector of dimension 0, which is written []. For a set of vectors $\mathbf{A}$ of dimension m, the set $\mathbf{A}_{|n}$ denotes the set $\{\vec{x}[1..n] | \vec{x} \in \mathbf{A}\}$ if $n \leq m$, and $\{\vec{x} \,|\vec{x} \in \mathbb{Z}^n, \vec{x}[1..m] \in \mathbf{A}\}$ otherwise. By convention, the $_|$ operator has priority on all other operators on sets.

Furthermore, $\ll$ denotes the strict lexicographic order on integral vectors. When clear from the context, "max" denotes $\max_{\ll}$, i.e. the maximum operator according to the $\ll$ order. An instance of Statement `S` is denoted by $\langle \mathtt{S}, \vec{x} \rangle$, where $\vec{x}$, the iteration vector of `S`, is the vector built from the counters of loops surrounding `S` – including `while` loops – from outside inward.

By convention, program statements are labeled by capital letters in typewriter style. Sets of vectors are denoted by capital letters in bold style, properties by letters in calligraphic style, and operations (instances of statements) by the last letters of the Greek alphabet ($\varsigma, \sigma, \phi, \chi$, etc.)

2 A Motivating Example

The following example, even though already used in a previous work [6], illustrates the kind and the precision of dataflow information we want to obtain. (The reader is referred to [6] for the formal derivation of the result.)

```
      program M
      do i = 1, n
S0         a(i) = ...
           if ... then
              do j = i , n+2
S1               a(j) = a(j-2)
              enddo
           endif
      enddo
```

Assume that $n = 4$, and let us study the case of the *instance* of Statement $\mathtt{S}_1$ when $i = 3$ and $j = 4$, i.e. $\langle \mathtt{S}_1, 3, 4 \rangle$. Note that we don't even know at compile-time if this instance actually executes. If it does,

[1] Similarly to `do` loops, an iteration of a `while` loop is denoted by giving its ordinal number w in the iteration sequence.

however, then the problem is to know where and when the right-hand-side value `a(2)` was produced. This source may be an instance of $\mathtt{S}_1$, but not if $i > 3$, since this instance would execute *after* $\langle \mathtt{S}_1, 3, 4 \rangle$. Since the source must write into `a(2)`, the value of j is fixed to 2. This source cannot be an instance of $\mathtt{S}_1$ for $i = 3$ either, since one can deduce from the bounds of the j loop that $j \geq i$. Thus, *possible sources* are instances $\langle \mathtt{S}_1, 1, 2 \rangle$ and $\langle \mathtt{S}_1, 2, 2 \rangle$. Another potential source is $\langle \mathtt{S}_0, 2 \rangle$. Note moreover that $\langle \mathtt{S}_0, 2 \rangle$ overwrites the value that $\langle \mathtt{S}_1, 1, 2 \rangle$ may have written. Thus, the set of potential sources is $\{\langle \mathtt{S}_0, 2 \rangle, \langle \mathtt{S}_1, 2, 2 \rangle\}$.

Actually, the iteration points of $\mathtt{S}_1$ fall into three groups (see Fig. 1 (b)):

- A member (i, j) of the first group is such that $j \geq i + 2$. It has one and only one possible source from $\mathtt{S}_1$ (namely, $\langle \mathtt{S}_1, i, j - 2 \rangle$) since, if point (i, j) executes, then $(i, j - 2)$ did execute too.
- On the contrary, a member of the second group has an unpredictable source. However, all the members of this group have at least one source, since all the array cells they read (`a(1)` through `a(n-1)`) are written into by $\mathtt{S}_0$. Dotted edges symbolize this.
- Finally, members of the third group do not have sources in the given program.

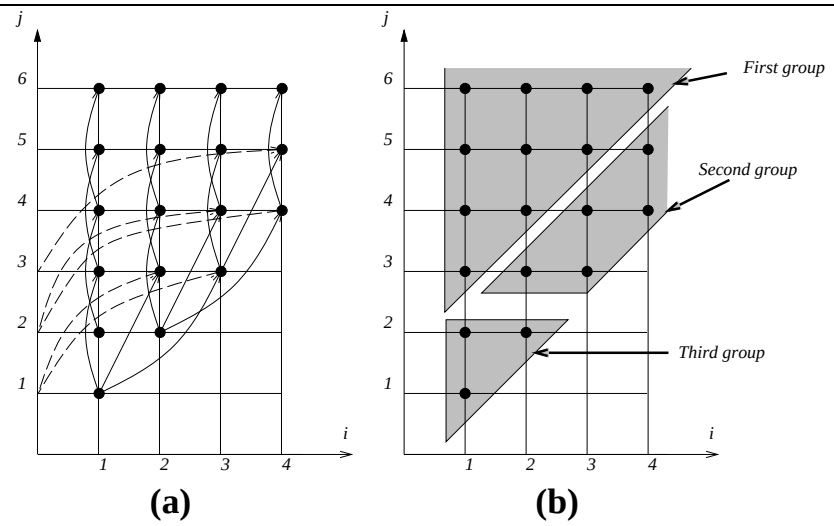

Figure 1: Dataflow graph of Program `M`.

3 An Overview of Array Dataflow Analysis

We first present the framework and the techniques used for exact array dataflow analysis and then give the idea of what will lead to fuzzy array dataflow analysis.

3.1 Exact Array Dataflow Analysis

In synthetic terms, Array Dataflow Analysis (ADA) is a very simple process. Let us first introduce some notations. A static control program is defined by its set of operations E and by a total order $\prec$ on it. If $\sigma, \tau \in E$, then $\sigma \prec \tau$ (read "σ before τ") means that operation τ does not begin executing until σ has terminated. The precise definition of $\prec$ will be given later (section 3.2).

To each operation σ are associated two sets of memory cells: $R(\sigma)$, the set of read cells, and $M(\sigma)$, the set of modified cells. For static control programs, these sets can be constructed by a simple examination of the program text.

The basic problem of array dataflow analysis is, given an operation τ (the "sink") and a memory cell c which is read by τ ($c \in R(\tau)$), to find the "source"of c in τ. The source is an operation $\sigma(c, \tau)$ which 1)

writes into c ($c \in M(\sigma(c,\tau))$), 2) which is executed before τ, and 3) such that no operation which executes between $\sigma(c,\tau)$ and τ also writes into c.

Let us consider the following set: $Q(c,\tau) = \{\phi | \, c \in M(\phi), \phi \prec \tau\}$. It is easy to see that the above definition of σ is exactly the definition of the maximum of $Q(c,\tau)$ according to $\prec$: $\sigma(c,\tau) = \max_{\prec} Q(c,\tau)$. In this section, all maxima are computed according to $\prec$. Hence this suffix will be omitted without ambiguity.

The computation of $\sigma(c,\tau)$ is discussed in depth in [10]. Let us just say here that the set $Q(c,\tau)$ can be written explicitly as a union of subsets, each of which is associated to a statement which modifies c and a dependence depth. Let us enumerate these subsets as:

$$Q(c,\tau) = \bigcup_{i=1}^{n} Q_i(c,\tau).$$

In this paper, we will repeatedly use the following general property:

Property 1 *If* $F = \bigcup_{i\in I} F_i$, *then* $\max F = \max_{i\in I} \max F_i$.

Applying this result to the present case gives:

$$\max Q(c,\tau) = \max_{i=1}^{n} \varsigma_i(c,\tau), \tag{1}$$

where

$$\varsigma_i(c,\tau) = \max Q_i(c,\tau). \tag{2}$$

The dependence from $\varsigma_i(c,\tau)$ to τ is known as a *direct dependence* [2]. The evaluation of (1) when the direct dependences are known is a simple exercise in formal computation.

3.2 Notations and basic concepts

The *depth* of a construct is the number of surrounding loops. The counter of a loop at depth k is the $(k+1)$-th component of the iteration vector.

Let $\langle \mathtt{R}, \vec{y} \rangle$ be the sink operation reading an element $\mathtt{a}(\vec{g}(\vec{y}))$ of array $\mathtt{a}$ and $\langle \mathtt{S}, \vec{x} \rangle$ be an operation writing it with subscripts $\mathtt{a}(\vec{f}(\vec{x}))$. Let $N_{\mathtt{SR}}$ be the number of loops surrounding both $\mathtt{S}$ and $\mathtt{R}$. Since the quantity $N_{\mathtt{SS}}$ occurs very often in the following sections, it will be abbreviated as $N_{\mathtt{S}}$. Let $\lhd$ be the textual order of the program. $\mathtt{S} \lhd \mathtt{T}$ iff $\mathtt{S}$ occurs before $\mathtt{T}$ in the source text. The sequential execution order, $\prec$, is:

$$\langle \mathtt{S}, \vec{x} \rangle \prec \langle \mathtt{R}, \vec{y} \rangle \equiv \bigvee_{p=0}^{N_{\mathtt{SR}}} \langle \mathtt{S}, \vec{x} \rangle \prec_p \langle \mathtt{R}, \vec{y} \rangle, \tag{3}$$

where

$$0 \leq p < N_{\mathtt{SR}} : \langle \mathtt{S}, \vec{x} \rangle \prec_p \langle \mathtt{R}, \vec{y} \rangle \Leftrightarrow (\vec{x}[1..p] = \vec{y}[1..p]) \wedge (\vec{x}[p+1] < \vec{y}[p+1]), \tag{4}$$

$$\langle \mathtt{S}, \vec{x} \rangle \prec_{N_{\mathtt{SR}}} \langle \mathtt{R}, \vec{y} \rangle \Leftrightarrow \vec{x}[1..N_{\mathtt{SR}}] = \vec{y}[1..N_{\mathtt{SR}}] \wedge \mathtt{S} \lhd \mathtt{R}. \tag{5}$$

For a given loop at depth k, $\vec{x}[k+1]$ has a minimum and a maximum which are given by the loop bounds. In the static control case, these bounds are affine functions of outer loop counters and structure parameters:

$$l_k(\vec{x}[1..k]) \leq \vec{x}[k+1] \leq u_k(\vec{x}[1..k]). \tag{6}$$

The iteration domain of a statement $\mathtt{S}$ is denoted by $\mathbf{I}(\mathtt{S})$ and is given by the conjunction of all inequalities (6) for the surrounding loops and of the predicates of all surrounding `while` and `if` constructs.

Let us suppose that operation τ above is an iteration of Statement $\mathtt{R} : \langle \mathtt{R}, \vec{y} \rangle$ and that cell c is element $\mathtt{a}(\vec{g}(\vec{y}))$ of an array $\mathtt{a}$. Let us suppose that we are investigating candidate sources from a Statement $\mathtt{S}$ at depth $p : \langle \mathtt{S}, \vec{x} \rangle$. If the source program handles its arrays correctly, $\mathtt{S}$ necessarily writes into array $\mathtt{a}$. Let $\vec{f}(\vec{x})$ be the relevant subscripts.

The candidate source $\langle \mathtt{S}, \vec{x} \rangle$ has to satisfy several constraints: $\langle \mathtt{S}, \vec{x} \rangle$ is a valid operation (*existence predicate*), $\langle \mathtt{S}, \vec{x} \rangle$ and $\langle \mathtt{R}, \vec{y} \rangle$ access the same array cell (*subscript equation*), $\langle \mathtt{S}, \vec{x} \rangle$ is executed before $\langle \mathtt{R}, \vec{y} \rangle$ at depth

p (*sequencing predicate*) and the sources have to be computed under the hypothesis that $\langle \mathtt{R}, \vec{y} \rangle$ is a valid operation (*environment*). To sum up, let us list these predicates:

$$\vec{x} \in \mathbf{I}(\mathtt{S}) \text{ (existence predicate)} \tag{7}$$

$$\vec{f}(\vec{x}) = \vec{g}(\vec{y}),\ \vec{f} \text{ and } \vec{g} \text{ are affine functions of } \vec{x} \text{ and } \vec{y}, \text{ respectively (subscript equation)} \tag{8}$$

$$\langle \mathtt{S}, \vec{x} \rangle \prec_p \langle \mathtt{R}, \vec{y} \rangle \text{ (sequencing predicate)} \tag{9}$$

$$\vec{y} \in \mathbf{I}(\mathtt{R}) \text{ (environment)}$$

We conclude first that the Q_i in (2) are indexed in fact by $\mathtt{S}$ and p. Each $Q^p_{\mathtt{S}}(\vec{y})$ is associated to the set:

$$\mathbf{Q}^p_{\mathtt{S}}(\vec{y}) = \{\vec{x} \mid \vec{x} \in \mathbf{I}(\mathtt{S}),\ \vec{f}(\vec{x}) = \vec{g}(\vec{y}),\ \langle \mathtt{S}, \vec{x} \rangle \prec_p \langle \mathtt{R}, \vec{y} \rangle\}, \tag{10}$$

by the rule: $\langle \mathtt{S}, \vec{x} \rangle \in Q^p_{\mathtt{S}} \equiv \vec{x} \in \mathbf{Q}^p_{\mathtt{S}}(\vec{y})$. Furthermore,$\prec$ in $Q^p_{\mathtt{S}}$ corresponds to the lexicographic order $\ll$ in $\mathbf{Q}^p_{\mathtt{S}}(\vec{y})$.

Since each predicate $\prec_p$ is affine, $\mathbf{Q}^p_{\mathtt{S}}(\vec{y})$ is a Z-polytope. The *direct dependence from* $\mathtt{S}$ *to* $\mathtt{R}$ *at depth* p is given by the maximal element:

$$\vec{K}^p_{\mathtt{S}}(\vec{y}) = \max_{\ll} \mathbf{Q}^p_{\mathtt{S}}(\vec{y}). \tag{11}$$

The maximal value is computed for each depth by integer linear programming [9]. The corresponding *operation* is denoted by:

$$\varsigma^p_{\mathtt{S}}(\vec{y}) = \langle \mathtt{S}, \vec{K}^p_{\mathtt{S}}(\vec{y}) \rangle. \tag{12}$$

The result is a *quast*, i.e. a many-level conditional in which:

- Predicates are tests for the positiveness of quasi-affine forms[2] in the loop counters and structure parameters.
- Leaves are either operation names whose iteration vector components are again quasi-affine, or $-$. The special name $-$ indicates that the array cell under study is not modified by $\mathtt{S}$. A coherent way of thinking about $-$ is to consider it as the name of an operation which is executed once before all other operations of the program: $\forall \mathtt{S}, \vec{x} : - \prec \langle \mathtt{S}, \vec{x} \rangle$. In the following, $-$ will be used to denote, also, an undefined vector.

3.3 Combining direct dependences

In the following, we will consider m statements, $\mathtt{S}_k$ for $1 \leq k \leq m$, writing into array $\mathtt{a}$. Beside, we will suppose that the read statement, $\mathtt{R}$, and the read cell, c, stay fixed. We may thus write $\sigma(\vec{y})$ instead of $\sigma(c, \langle \mathtt{R}, \vec{y} \rangle)$. With this convention, the equivalent of (1) is:

$$\sigma(\vec{y}) = \max_{\prec\ 1 \leq k \leq m} \left(\max_{\prec\ 0 \leq p \leq N_{\mathtt{S}_k \mathtt{R}}} \langle \mathtt{S}_k, \vec{K}^p_{\mathtt{S}_k}(\vec{y}) \rangle \right). \tag{13}$$

When the direct dependences have been found, one must construct the real source by computing their maximum. Let q be the number of candidate sources $\varsigma^p_{\mathtt{S}_k}(\vec{y})$. To simplify the notations, we assign an index number $n, 1 \leq n \leq q$, to each $\varsigma^p_{\mathtt{S}_k}(\vec{y})$, and rename the latter into ψ_n. Then, the basic algorithm computes the recurrence: $1 \leq n \leq q, \quad \chi_n = \max_{\prec}(\chi_{n-1}, \psi_n)$, with $\chi_0 = -$. maximum of two quasts. This is done with the help of some simple rewriting rules (see [10] for details).

[2] Quasi-affine forms may include integer division.

3.4 From ADA to FADA

As soon as we extend our program model to include conditionals, `while` loops, `do` loops with non-linear bounds or subscripts, the algorithm above breaks down. The reason is that conditions (7) and (8) may contain intractable terms. One possibility is to ignore them. In this way, (7) is replaced by $\vec{x} \in \widehat{\mathbf{I}}(\mathtt{S})$, where $\widehat{\mathbf{I}}(\mathtt{S})$ is a superset of $\mathbf{I}(\mathtt{S})$ which is obtained by ignoring non-linear constraints. Supposing for the moment that the subscript condition is still linear, we may obtain an approximate set of candidate sources:

$$\widehat{\mathbf{Q}}^p_{\mathtt{S}}(\vec{y}) = \{\vec{x} \mid \vec{x} \in \widehat{\mathbf{I}}(\mathtt{S}),\ \vec{f}(\vec{x}) = \vec{g}(\vec{y}),\ \langle \mathtt{S}, \vec{x}\rangle \prec_p \langle \mathtt{R}, \vec{y}\rangle\}. \tag{14}$$

However, we can no longer say that the direct dependence is given by the lexicographic maximum of this set, since the result may precisely be one of the candidates which is excluded by the non-linear part of $\mathbf{I}(\mathtt{S})$. One solution is to take all of $\widehat{\mathbf{Q}}^p_{\mathtt{S}}(\vec{y})$ as an approximation to the direct dependence. If we do that, and with the exception of very special cases, computing the maximum of approximate direct dependences has no meaning, and the best we can do is to use their union as an approximation. Can we do better than that? Let us consider some examples.

```
     program E1
     do  x = 1 while ...
S1:      s = ...
     end do
S2:  s = ...
R :  ...  = ...  s ...
     end
```

```
     program E2
     do  x = 1, n
         if ...  then
S1:          s = ...
         else
S2:          s = ...
         end if
     end do
R :  ...  = ...  s ...
     end
```

Here and in the following examples, we will always stipulate that all relevant accesses to the memory cell we are interested in – here `s` – have been exhibited. What is the source of `s` in Statement `R` in `E1`? There are two possibilities, Statements $\mathtt{S}_1$ and $\mathtt{S}_2$. In the case of $\mathtt{S}_2$, the direct dependence is exactly $\langle \mathtt{S}_2, []\rangle$. Things are more complicated for $\mathtt{S}_1$, since we have no idea of the iteration count of the `while` loop. We may, however, give a name to this count, say N, and write the set of candidates as:

$$\mathbf{Q}^0_{\mathtt{S}_1}([]) = \{\langle \mathtt{S}_1, x\rangle | \, 1 \leq x \leq N\}.$$

We may then compute the maximum of this set, which is simply

$$\varsigma^0_{\mathtt{S}_1}([]) = \mathbf{if}\ N > 0\ \mathbf{then}\ \langle \mathtt{S}_1, N\rangle\ \mathbf{else}\ -.$$

The last step is to take the maximum of this result and $\langle \mathtt{S}_2, []\rangle$, which is $\langle \mathtt{S}_2, []\rangle$. We have thus formally derived the expected precise result. The trick here has been to give a name to an unknown quantity, N, and to solve the problem with N as a parameter. It so happens that N disappears in the solution, giving an exact result.

The other example `E2` is slightly more complicated: we assume that $n \geq 1$. What is the source of `s` in Statement `R`? We may build an approximate candidate set from $\mathtt{S}_1$ and another one from $\mathtt{S}_2$. Since both are approximate, we cannot do anything beside taking their union, and the result is highly inaccurate.

Another possibility is to partition the set of candidates according to the value x of the loop counter. Let us introduce a new boolean function $b(x)$ which represents the outcome of the test at iteration x. The x-th candidate may be written

$$\tau(x) = \mathbf{if}\, b(x)\ \mathbf{then}\ \langle \mathtt{S}_1, x\rangle\ \mathbf{else}\ \langle \mathtt{S}_2, x\rangle.$$

We then have to compute the maximum of all these candidates (this is an application of Property 1). It is an easy matter to prove that $x < x' \Rightarrow \tau(x) \prec \tau(x')$, so the source is $\tau(n)$. Since we have no idea of the value of $b(n)$, the best we can do is to say that we have a source *set*, or a *fuzzy* source, which is obtained by taking the union of the two arms of the conditional:

$$\Sigma([]) = \{\langle \mathtt{S}_1, n\rangle, \langle \mathtt{S}_2, n\rangle\}. \tag{15}$$

Notice here the precision we have been able to achieve. However, the technique we have used here is not easily generalized. Another way of obtaining the same result is the following. Let $\mathbf{L} = \{x | 1 \leq x \leq n\}$. Observe that the candidate set from $\mathtt{S}_1$ (resp. $\mathtt{S}_2$) can be written $\{\langle \mathtt{S}_1, x\rangle | x \in \mathbf{D}_{\mathtt{S}_1} \cap \mathbf{L}\}$ (resp. $\{\langle \mathtt{S}_2, x\rangle | x \in \mathbf{D}_{\mathtt{S}_2}\} \cap \mathbf{L}$) where $\mathbf{D}_{\mathtt{S}_1} = \{x | b(x) = \mathbf{true}\}$ and $\mathbf{D}_{\mathtt{S}_2} = \{x | b(x) = \mathbf{false}\}$. Obviously,

$$\mathbf{D}_{\mathtt{S}_1} \cap \mathbf{D}_{\mathtt{S}_2} = \emptyset, \tag{16}$$

and

$$\mathbf{D}_{\mathtt{S}_1} \cup \mathbf{D}_{\mathtt{S}_2} = \mathbb{Z}. \tag{17}$$

We have to compute $\beta = \max(\max \mathbf{D}_{\mathtt{S}_1} \cap \mathbf{L}, \max \mathbf{D}_{\mathtt{S}_2} \cap \mathbf{L})$. It is a general property that (17) implies that:

$$\beta = \max \mathbf{L} = n. \tag{18}$$

By (16) we know that β belongs either to $\mathbf{D}_{\mathtt{S}_1}$ or $\mathbf{D}_{\mathtt{S}_2}$ which gives again the result (15).

To summarize these observations, our method will be to give new names (or *parameters*) to the result of maxima calculations in the presence of non-linear terms. These parameters are not arbitrary. The sets they belong to – the parameter domains – are in relations to each others, as for instance (16-17). These relations can be found simply by examination of the syntactic structure of the program, or by more sophisticated techniques. From these relations between the parameter domains follow relations on the parameters, like (18), which can then be used to simplify the resulting fuzzy sources. In some cases, these relations may be so precise as to reduce the fuzzy source to a singleton, thus giving an exact result.

4 Basic Techniques for FADA

We present in this section a formal definition of fuzzy analysis. First of all, we define a representation for non-linear constraints. Thanks to this representation, the expression of the source boils down to a computable expression with linear constraints and unknown parameters. When these parameters take all the values of a set defined by linear constraints, we get a set of possible sources, called the fuzzy source. How this set of values is built will be the subject of the next sections.

4.1 Non-linear constraints

Let us first have a close look at the non-linear constraints. Notice that they come either from the predicate of a `while` or `if`, from a non-linear loop bound appearing in the existence predicate (7), or from a non-linear array subscript appearing in the conflicting access predicate (8). Each constraint can be numbered according to its apparition order in the text of the program. Let $\mathbf{C}$ denote the set of integers that index non-linear constraints. Given a constraint c_h, $h \in \mathbf{C}$, we note $\mathtt{T}_h$ the statement in which it appears. This statement is either the `then` or `else` branch of a conditional, or a loop with non-affine bounds, or an assignment statement in which a non-linear subscript is used in an array access.

If c_h appears in the set of candidate sources $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$, the write operation $\langle \mathtt{S}_k, \vec{x}\rangle$ depends on the value of c_h at the operation $\langle \mathtt{T}_h, \vec{x}[1..N_h]\rangle$, where N_h equals $N_{\mathtt{T}_h}$ if $\mathtt{T}_h$ is a conditional or an assignment, and $N_{\mathtt{T}_h} + 1$ if $\mathtt{T}_h$ is a `do` or a `while`.

In $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$, the expression of the non-linear constraint c_h is $c_h(\vec{z}, \vec{y})$, $\vec{z} = \vec{x}[1..N_h]$, where $\vec{z} \in \mathbf{I}(\mathtt{T}_h)$ is N_h-dimensional. c_h depends on $\vec{y}$ in the case it comes from Equation (8). However, since the only term depending on p is the sequencing predicate which is linear, non-linear constraints cannot depend on p.

Definition 1 (parameter set) *Let $\mathbf{P}_h(\vec{y})$ be the set of iteration vectors for which the constraint c_h is true. It is called the* parameter set *and is defined by:*

$$\mathbf{P}_h(\vec{y}) = \left\{ \vec{z} \,\middle|\, \vec{z} \in \mathbb{Z}^{N_h}, c_h(\vec{z}, \vec{y}) \right\}.$$

Definition 2 (parameter domain) *Let* $\mathbf{C}_{\mathtt{S}_k} \subseteq \mathbf{C}$ *denote the set of the indices of the constraints involved in the computation of* $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$ *and* $M_{\mathtt{S}_k} = \max_{h \in \mathbf{C}_{\mathtt{S}_k}} N_h$. *The set:*

$$\mathbf{D}_{\mathtt{S}_k}(\vec{y}) = \left\{ \vec{z} \;\middle|\; \vec{z} \in \mathbb{Z}^{M_{\mathtt{S}_k}}, \bigwedge_{h \in \mathbf{C}_{\mathtt{S}_k}} (\vec{z}[1..N_h] \in \mathbf{P}_h(\vec{y})) \right\},$$

is the set of iteration vectors for which all of the constraints indexed by $\mathbf{C}_{\mathtt{S}_k}$ *are true. This set is called the* parameter domain *of* $\mathtt{S}_k$.

Note that $M_{\mathtt{S}_k}$ does not depend on $\vec{y}$ and that $M_{\mathtt{S}_k} \leq N_{\mathtt{S}_k}$. By convention, when all constraints in $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$ are linear, $\mathbf{D}_{\mathtt{S}_k}(\vec{y}) = \mathbb{Z}^{N_{\mathtt{S}_k}}$.

The following piece of code illustrates these definitions:

```
    program E3

T1: do x=1 while f(x)>0
S1:    a(x)=x
       if p(x)
T2:    then
S2:         a(x)=2*x
T3:    else
S3:         a(x)=3*x
       end if
    end do
    do y=1,n
R :    r=a(y)
    end do
    end
```

The non-linear constraints are: $c_1(x,y) = (f(x) > 0)$ from $\mathtt{T}_1$, $c_2(x,y) = p(x)$ from $\mathtt{T}_2$, $c_3(x,y) = \neg p(x)$ from $\mathtt{T}_3$. The parameter sets are: $\mathbf{P}_1(y) = \{x | f(x) > 0\}$, $\mathbf{P}_2(y) = \{x | p(x)\}$ and $\mathbf{P}_3(y) = \{x | \neg p(x)\} = \overline{\mathbf{P}_2(y)}$.
The domains are $\mathbf{D}_{\mathtt{S}_1}(y) = \mathbf{P}_1(y)$, $\mathbf{D}_{\mathtt{S}_2}(y) = \mathbf{P}_1(y) \cap \mathbf{P}_2(y)$ and $\mathbf{D}_{\mathtt{S}_3}(y) = \mathbf{P}_1(y) \cap \overline{\mathbf{P}_2(y)}$.

4.2 Parameterization

Let us recall the definition (13) of the source:

$$\sigma(\vec{y}) = \max_{\prec \; 1 \leq k \leq m} \left(\max_{\prec \; 0 \leq p \leq N_{\mathtt{S}_k \mathtt{R}}} \langle \mathtt{S}_k, \vec{K}^p_{\mathtt{S}_k}(\vec{y}) \rangle \right).$$

The purpose of parameterization is to code (13) as a linear problem, so as to enable the computation of the source $\sigma(\vec{y})$ (or perhaps an approximation of this source) using linear programming methods and tools, even in the presence of non-linear constraints. We give thereafter the steps to transform (13) in a parametric linear problem. Let us also recall the definition (11) of the direct dependence:

$$\vec{K}^p_{\mathtt{S}_k}(\vec{y}) \quad = \quad \max_{\ll} \mathbf{Q}^p_{\mathtt{S}_k}(\vec{y}). \tag{19}$$

We first partition each set $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$ into subsets defined by parametric linear constraints. Let $\mathbf{L}^p_{\mathtt{S}_k}$ denote the set of vectors of dimension $N_{\mathtt{S}_k}$ defined by the linear constraints appearing in $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$. The set of candidate sources is:

$$\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y}) = \mathbf{L}^p_{\mathtt{S}_k}(\vec{y}) \bigcap \mathbf{D}_{\mathtt{S}_k}(\vec{y})_{|N_{\mathtt{S}_k}}.$$

Partitioning $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$ is obtained by partitioning $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$ as the union of its elements:

$$\mathbf{D}_{\mathtt{S}_k}(\vec{y}) = \bigcup_{\vec{\alpha} \in \mathbf{D}_{\mathtt{S}_k}(\vec{y})} \{\vec{\alpha}\}.$$

Let $\mathbf{Q}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}) = \mathbf{L}^p_{\mathtt{S}_k}(\vec{y}) \cap \{\vec{\alpha}\}_{|N_{\mathtt{S}_k}}$ denote a subset of the partition of $\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y})$. Then:

$$\mathbf{Q}^p_{\mathtt{S}_k}(\vec{y}) = \bigcup_{\vec{\alpha} \in \mathbf{D}_{\mathtt{S}_k}(\vec{y})} \mathbf{Q}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}). \tag{20}$$

From Equations (19) and (20), we have:

$$\vec{K}^p_{\mathtt{S}_k}(\vec{y}) \quad = \quad \max_{\ll} \left(\bigcup_{\vec{\alpha} \in \mathbf{D}_{\mathtt{S}_k}(\vec{y})} \mathbf{Q}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}) \right). \tag{21}$$

From Equation (21) and Property 1, we obtain:

$$\vec{K}^p_{\mathtt{S}_k}(\vec{y}) \quad = \quad \max_{\ll \ \vec{\alpha} \in \mathbf{D}_{\mathtt{S}_k}(\vec{y})} \left(\max_{\ll} \mathbf{Q}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}) \right). \tag{22}$$

An elementary direct dependence $\vec{K}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha})$ can then be evaluated for each subset $\mathbf{Q}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha})$ as a function of its parameters:

$$\vec{K}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}) = \max_{\ll} \mathbf{Q}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}), \tag{23}$$

which is computable by parametric integer programming. From Equations (22) and (23), we have:

$$\vec{K}^p_{\mathtt{S}_k}(\vec{y}) \quad = \quad \max_{\ll \ \vec{\alpha} \in \mathbf{D}_{\mathtt{S}_k}(\vec{y})} \vec{K}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\alpha}). \tag{24}$$

If the maximum as defined by (24) exists, then it is reached in at least one vector of $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$ since there is a finite number of candidate sources. Such a vector is called a *parameter of the maximum*:

Definition 3 (parameter of the maximum) *All the vectors in $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$ for which (24) is defined are called* parameters of the maximum of $\mathbf{D}_{\mathtt{S}_k}$ for Statement $\mathtt{S}_k$ at depth p. *Let $\vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$ be one such vector. (If the maximum does not exist, we set $\vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$ to an undefined value.) The following equality always holds:*

$$\vec{K}^p_{\mathtt{S}_k}(\vec{y}) = \vec{K}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\beta}^p_{\mathtt{S}_k}(\vec{y})). \tag{25}$$

In other words:

$$\vec{\beta}^p_{\mathtt{S}_k}(\vec{y}) = \max_{\ll} \left\{ \vec{\alpha} \;\middle|\; \vec{\alpha} \in \mathbf{D}_{\mathtt{S}_k}(\vec{y}), \vec{\alpha} = \left(\max_{\ll} \mathbf{Q}^p_{\mathtt{S}_k}(\vec{y}) \right)_{|M_{\mathtt{S}_k}} \right\}. \tag{26}$$

Thus, (13) implies that the source can be written as:

$$\sigma(\vec{y}) \quad = \quad \max_{\prec \ 1 \le k \le m} \left(\max_{\prec \ 0 \le p \le N_{\mathtt{S}_k \mathtt{R}}} \langle \mathtt{S}_k, \vec{K}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\beta}^p_{\mathtt{S}_k}(\vec{y})) \rangle \right). \tag{27}$$

We can extend (12) into:

$$\varsigma^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\beta}^p_{\mathtt{S}_k}(\vec{y})) = \langle \mathtt{S}, \vec{K}^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\beta}^p_{\mathtt{S}_k}(\vec{y})) \rangle. \tag{28}$$

4.3 Fuzziness

To sum things up, we enumerated each set $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$ of non-linear constraints by parameters α. Among these parameters, we distinguished one element for each p, the parameter of the maximum $\vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$. The benefit is that Expression (27) is computable exactly by parametric integer programming as a function of the parameters of the maximum.

However, parameters of the maximum cannot themselves be computed, because the sets $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$ of non-linear constraints cannot be handled.

A very simple method is to compute a *set of possible sources* – or a fuzzy source – by giving all possible values to the parameters. This would mean that we would not even try to take non-linear constraints into account. Obviously, this is a safety net for a FADA analyzer and is similar to the "panic mode" in Wonnacott's work [15]. A variant of this solution is to keep the non-linear expressions in the solution, without trying to interpret them. In this case, the analyzer just hopes that a later phase of the compiler will be able to handle this expression.

A better approach is to reduce the size of $\Sigma(\vec{y})$. The first idea is to try to find properties on $\vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$. This was the method used in our initial work [6] and by Wonnacott.

The second idea, proposed in this paper, is to handle separately the non-linear constraints. To do that, we will try to find properties (call them $\mathcal{P}$) on the parameter domains $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$. From these properties $\mathcal{P}$ on $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$, we will deduce linear properties (call them $\mathcal{P}^*$) on the parameters $\vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$. The benefit of this approach is that we can then prove, for some $\mathcal{P}$, that the properties found on parameters of the maximum are the most precise that can be derived. That is, there is no loss of information when deriving $\mathcal{P}^*$ from $\mathcal{P}$.

Therefore, the method to be presented in the next sections will proceed in five steps:

1. Properties $\mathcal{P}$ will be derived from the parameter domains (Section 5.2).
2. We will consider all sets, call them $\mathbf{G}_k$, satisfying properties $\mathcal{P}$. Note that for all $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$, there is at least one set which satisfies $\mathcal{P}$, namely $\mathbf{D}_{\mathtt{S}_k}(\vec{y})$.
3. For each set $\mathbf{G}_k$, we consider a parameter of the maximum $\vec{\gamma}^p_k$ for Statement $\mathtt{S}_k$ at depth p. Note that when $\mathbf{G}_k = \mathbf{D}_{\mathtt{S}_k}(\vec{y})$ then $\vec{\gamma}^p_k = \vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$. We must use as many $\vec{\gamma}^p_k$ as there are depths, since each parameter of the maximum is used to describe the set $\mathbf{L}^p_{\mathtt{S}_k}(\vec{y}) \bigcap \mathbf{D}_{\mathtt{S}_k}(\vec{y})_{|N_{\mathtt{S}_k}}$ which depends on p.
4. We derive properties $\mathcal{P}^*$ defining exactly the set of parameters $\vec{\gamma}^p_k$ (Section 6).
5. We build the set of sources corresponding to each $\vec{\gamma}^p_k$:

$$\Sigma(\vec{y}) = \left\{ \max_{\prec\ 1\le k\le m} \left(\max_{\prec\ 0\le p\le N_{\mathtt{S}_k \mathtt{R}}} \varsigma^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\gamma}^p_k) \right) \middle| \vec{\gamma}^p_k \in \mathbb{Z}^{M_{\mathtt{S}_k}}, \mathcal{P}^* \left(\vec{\gamma}^0_1, \ldots, \vec{\gamma}^{N_{\mathtt{S}_m \mathtt{R}}}_m \right) \right\}. \tag{29}$$

which can be computed exactly if $\mathcal{P}^*$ is a conjunction or disjunction of linear constraints.

The fuzziness of the source depends on the precision with which $\mathcal{P}^*$ abstracts the relations existing among the parameters of the maximum $\vec{\beta}^p_{\mathtt{S}_k}(\vec{y})$, $k = 1..m$.

4.4 Removing Parameters

The term $\max_{\prec 1\le k\le m} \left(\max_{\prec 0\le p\le N_{\mathtt{S}_k \mathtt{R}}} \varsigma^{*p}_{\mathtt{S}_k}(\vec{y}, \vec{\gamma}^p_k) \right)$ in (29) is a quast which is computed as in Section 3.3. Consider a leaf in which some parameters appear. This leaf represents the set of sources obtained by giving all possible values to these parameters. The set of possible values is obtained by "anding" all predicates in the unique path from the root of the quast to the leaf in question.

Rule 1 *Let $A(\vec{\gamma})$ be a leaf governed by l predicates $P_1, \ldots, P_l$ in the unique path from the root to the leaf. Then $A(\vec{\gamma})$ is transformed into $\{A(\vec{\gamma}) | \bigwedge_{i=1}^{l} P_i\}$.*

After a systematic application of this rule, any leaf in which parameters occur is transformed into a *set* in which the parameters are bound by the predicates governing the leaf. Leaves which do not depend on parameters become singletons.

Now consider the quast: **if** $C(\vec{\gamma})$ **then** $\mathcal{A}$ **else** $\mathcal{B}$. Thanks to Rule 1, $\mathcal{A}$ and $\mathcal{B}$ are sets of sources. Since the exact value of $\vec{\gamma}$ is unknown, we cannot predict the outcome of the test. The best we can do is to take the union $\mathcal{A} \cup \mathcal{B}$ as an approximation :

Rule 2 *A quast* **if** $C(\vec{\gamma})$ **then** $\mathcal{A}$ **else** $\mathcal{B}$ *is transformed into* $\mathcal{A} \cup \mathcal{B}$.

These observations are enough for solving example `E1`. There is one non-linear constraint, which is associated to the `while` loop at depth one. This gives rise to one parameter domain $\mathbf{D}_{\mathtt{S}_1}(\vec{y})$ and one parameter of the maximum, $\vec{\gamma}_1^0$, with no special properties. The equivalent of (23):

$$\vec{K}^0_{\mathtt{S}_1}([], \vec{\gamma}_1^0) = \max\{w | 1 \leq w, w = \vec{\gamma}_1^0\},$$

gives the solution $\varsigma^0_{\mathtt{S}_1}([], \vec{\gamma}_1^0) =$ **if** $\vec{\gamma}_1^0 \geq 1$ **then** $\langle \mathtt{S}_1, \vec{\gamma}_1^0 \rangle$ **else** $\perp$. The computation of the direct dependence from $\mathtt{S}_2$ to $\mathtt{S}_3$ is exact, since all constraints are linear. Their combination gives the final results:

$$\sigma([]) = \max(\langle \mathtt{S}_1, \textbf{if } \vec{\gamma}_1^0 \geq 1 \textbf{ then } \vec{\gamma}_1^0 \rangle \textbf{ else } \perp, \langle \mathtt{S}_2, [] \rangle) = \langle \mathtt{S}_2, [] \rangle.$$

Example `E2` is more complicated and needs more sophisticated techniques.

5 Finding Properties on Parameter Domains

Our aim now is to find all interesting properties of the parameter domains. Several techniques have been proposed that find mostly properties of each parameter domain, independently of each other. The two algorithms presented in Sections 5.2 and 7 find relations between the parameter domains. We will first define the general type of property we want to handle. Step 4 of the previous approach will thus be independent of the analysis technique.

5.1 General properties

The first kind of properties gives constraints on the elements of a parameter domain, independently of any other parameter domain. For instance, a polyhedron may be included in the parameter domain under study. This is the case when $\vec{y}$ is in a parameter domain and we will show that in this case there is no fuzziness at all in the computation of some direct dependences. Another example is when the vectors of the parameter domain satisfy a system of linear constraints. This system is provided by a detailed analysis of the non-linear constraints. Most of the properties found by Dumay [8] are of this kind and Maslov [13] has proved that for some specific non-linear constraints, the parameter domain is equal to a polyhedron. Given a known polyhedron $\mathbf{A}(\vec{y})$, this kind of properties can be written as: $\mathbf{A}(\vec{y}) \subseteq \mathbf{D}_{\mathtt{S}_k}(\vec{y})$ or $\mathbf{D}_{\mathtt{S}_k}(\vec{y}) \subseteq \mathbf{A}(\vec{y})$.

Another kind of properties involves two or more parameter domains. Such a property can be an inclusion using the union or intersection of parameter domains. For instance, in Program `E3`, we have $\mathbf{D}_{\mathtt{S}_2}(\vec{y}) \cup \mathbf{D}_{\mathtt{S}_3}(\vec{y}) = \mathbf{D}_{\mathtt{S}_1}(\vec{y})$ and $\mathbf{D}_{\mathtt{S}_2}(\vec{y}) \cap \mathbf{D}_{\mathtt{S}_3}(\vec{y}) = \emptyset$, which entails that the source can only come from Statement 2 or 3 and cannot come from both at the same time (no kill between 2 and 3).

Finally, the relations can involve parameter domains or their image by a simple affine function, so as to express the fact that a parameter domain is built from another parameter domain by translation, for instance. Such considerations are taken into account by Dumay and suggested by Wonnacott as an improvement of his methods. A simple affine function will be defined as a monotone increasing affine function, according to the lexicographic order.

In order to take into account the existing methods for finding properties of parameter domains, we will consider properties that can be written as conjunction of relations of inclusion between two sets. These sets are generated by union and intersection from the parameter domains and arbitrary polyhedra. To do this, we provide an algorithm that finds properties on the parameter domains that can be deduced from the structure of the program itself. The advantage of this method is that no case-by-case detailed analysis of the non-linear constraints is needed.

5.2 Structural Analysis Algorithm

In this section, we take benefit of the *structure* of the source program. Even though we only consider structured Fortran, we nevertheless have a problem: Fortran has no independent notation for compound statements. We have already tacitly extended Fortran by using non-numerical labels and the PL/I-like `do while` loop. In the same vein, we will use C-like braces {} to indicate statement grouping.

The starting point of the algorithm is a pruned version of the abstract syntax tree (A.S.T.), in which the only statements are the candidate sources $\mathtt{S}_k, 1 \leq k \leq m$, the read Statement `R` and all the control statements which surround them. We will extend the concept of a parameter domain to all statements in this simplified A.S.T. Consider for instance a compound statement $\mathtt{T}_0 : \{\mathtt{T}_1; \ldots; \mathtt{T}_n\}$: the parameter domain of $\mathtt{T}_0$, $\mathbf{D}_{\mathtt{T}_0}(\vec{y})$ is associated to the non-linear part of the conditions under which $\mathtt{T}_0$ is executed. (Again, $\vec{y}$ is the iteration vector of the read Statement `R`.) Depending on the nature of Statement $\mathtt{T}_j$, $1 \leq j \leq n$, we may say that $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{D}_{\mathtt{T}_j}(\vec{y})$, or at least that $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) \supseteq \mathbf{D}_{\mathtt{T}_j}(\vec{y})_{|M_{\mathtt{T}_0}}$.

The algorithm is a recursive descent in the A.S.T that yields one or several relations from each visited nodes. A special symbol, $\mathbf{E}(\vec{y})$, will be used to denote the non-linear part of the environment (the conditions under which the read statement is executed). Note that the parameter domain associated to the compound statement representing the whole program is the set $\{[]\}$. At the end of the algorithm, a post-processing phase, which will be specified later, will eliminate unwanted information from the original result.

Structural analysis algorithm

1. $\mathtt{T}_0 : \{\mathtt{T}_1; \ldots; \mathtt{T}_n\}$: For $i = 1, \ldots, n$ do:
   (a) If $\mathtt{T}_i$ is another control statement, emit $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{D}_{\mathtt{T}_i}(\vec{y})$, then visit $\mathtt{T}_i$.
   (b) If $\mathtt{T}_i$ is one of the source statements, $\mathtt{S}_k$: `a(` $\vec{f}(\vec{x})$ `) = ...` and if $\vec{f}$ is linear, then emit: $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{D}_{\mathtt{T}_i}(\vec{y})$, else emit: $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) \supseteq \mathbf{D}_{\mathtt{T}_i}(\vec{y})_{|M_{\mathtt{T}_0}}$.
   (c) If $\mathtt{T}_i$ is the read statement: `R : ... = ... a(` $\vec{g}(\vec{y})$ `) ...`, then emit $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{E}(\vec{y})$.
2. $\mathtt{T}_0$: `do w = 1 while` p $\mathtt{T}_1$ `end do` : If p is linear[3] then emit: $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{D}_{\mathtt{T}_1}(\vec{y})$ else emit: $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) \supseteq \mathbf{D}_{\mathtt{T}_1}(\vec{y})_{|M_{\mathtt{T}_0}}$. Visit $\mathtt{T}_1$.
3. $\mathtt{T}_0$: `if` p `then` $\mathtt{T}_1$ `else` $\mathtt{T}_2$ `endif`: If p is non-linear then emit $\mathbf{D}_{\mathtt{T}_1}(\vec{y}) \cap \mathbf{D}_{\mathtt{T}_2}(\vec{y}) = \emptyset$ and $\mathbf{D}_{\mathtt{T}_1}(\vec{y}) \cup \mathbf{D}_{\mathtt{T}_2}(\vec{y}) = \mathbf{D}_{\mathtt{T}_0}(\vec{y})$, else emit: $\mathbf{D}_{\mathtt{T}_1}(\vec{y}) = \mathbf{D}_{\mathtt{T}_2}(\vec{y}) = \mathbf{D}_{\mathtt{T}_0}(\vec{y})$. Visit $\mathtt{T}_1$ and $\mathtt{T}_2$.
4. $\mathtt{T}_0$: `if` p `then` $\mathtt{T}_1$ `endif` : If p is non-linear then emit $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) \supseteq \mathbf{D}_{\mathtt{T}_1}(\vec{y})$ else emit: $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{D}_{\mathtt{T}_1}(\vec{y})$. Visit $\mathtt{T}_1$.
5. $\mathtt{T}_0$: `do i =` lb, ub $\mathtt{T}_1$ `end do` : If both lb and ub are linear, then emit: $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) = \mathbf{D}_{\mathtt{T}_1}(\vec{y})$, else emit $\mathbf{D}_{\mathtt{T}_0}(\vec{y}) \supseteq \mathbf{D}_{\mathtt{T}_1}(\vec{y})_{|M_{\mathtt{T}_0}}$. Visit $\mathtt{T}_1$.

As the algorithm needs to go through the reduced A.S.T once, the complexity is $\mathcal{O}(m.s)$, with s the maximum number of nested control structures and m the number of write statements. m also gives a bound on the number of leaves visited in the abstract tree: $\mathcal{O}(m)$.

[3] This indicates that the `while` loop may be transformed into a `for` loop and should not occur in restructured programs.

Post-processing phase The idea is to eliminate all domains except Environment $\mathbf{E}$ and the domains associated to potential sources. Emitted equations of the form $\mathbf{D} = \mathbf{D}'$ can be used to eliminate either $\mathbf{D}$ or $\mathbf{D}'$. Let us rank all domains in an arbitrary order, except that the domains of the source statements and $\mathbf{E}$ (the *protected* domains) are ranked last. Select an equation in which the highest ranking domain occurs, use it for eliminating this domain from all other relations, discard the equation and start again. The process stops as soon as the highest ranking domain is protected. At this point, discard all relations which contain unprotected domains. This phase may take as much as $\mathcal{O}(m^2)$ time.

Exact analysis Among the results may occur relations of the form: $\mathbf{E}(\vec{y}) = \mathbf{D}_{\mathtt{S}_k}(\vec{y})$ or $\mathbf{D}_{\mathtt{S}_k}(\vec{y}) \supseteq \mathbf{E}(\vec{y})_{|M_{\mathtt{S}_k}}$. Since we are computing sources under the hypothesis that the read statement is executed, we know that $\vec{y}$ belongs to $\mathbf{E}(\vec{y})$. Suppose then that the prefix $\vec{y}[1..M_{\mathtt{S}_k}]$ of $\vec{y}$ is in $\mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}}$. Thus, as the parameters of the maximum are lexicographically lower than $\vec{y}$ due the sequencing predicate, this entails that $\vec{y}[1..M_{\mathtt{S}_k}]$ is a parameter of the maximum and the analysis is exact.

An example of such an exact case is when the only `while` loop in the source program is the outermost statement. This result was proved by other, less general means in [5, 6] and justifies a conjecture in [4].

6 Constructing Properties on Parameters

In the previous section, the purpose was to extract properties $\mathcal{P}$ on the parameter domains. The purpose of this section is to derive properties $\mathcal{P}^*$ on parameters of the maximum from properties $\mathcal{P}$ on parameter domains, without forgetting sources (*correctness*) and without adding fuzziness (*precision*). For each relation on domains that is of the form given in Section 5.1, we will find a relation on the parameters that preserves both correctness and precision. Moreover, we prove that $\mathcal{P}^*$ is a conjunction or disjunction of linear inequalities thus enabling the exact computation of (29).

Notice that from (19) and (26), we immediately deduce the following result: the parameter of the maximum is equal to the $M_{\mathtt{S}_k}$ first components of $\vec{K}^p_{\mathtt{S}_k}(\vec{y})$ when the latter is defined. This can be generalized to the following property:

Property 2 *Let $\vec{\gamma}^p_k$ be a parameter of the maximum of any set $\mathbf{G}_k$ for Statement $\mathtt{S}_k$ at depth p. The value of $\vec{\gamma}^p_k$ is given by: $\vec{\gamma}^p_k = \max \mathbf{G}_k \cap \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}}$.*

This gives a characterization of the parameters of the maximum. We will use repeatedly this property in the following.

In the sequel, we will consider properties $\mathcal{P}$ that are inclusions between union of and intersection of sets. These sets are either parameter domains, or arbitrary sets defined by linear constraints. Moreover, the inclusion properties we consider are such that:

- The left-hand-side of $\subseteq$ only consists of intersections.
- The right-hand-side of $\subseteq$ only consists of unions.

To simplify the study of such relations, notice that:

$$\cup_i F_i \subseteq \cup_j F_j \Longleftrightarrow \forall i, F_i \subseteq \cup_j F_j\,, \tag{30}$$

$$\cap_i F_i \subseteq \cap_j F_j \Longleftrightarrow \forall j, \cap_i F_i \subseteq F_j\,. \tag{31}$$

Notice also that, until Theorem 1, we do not take into account the application of linear functions to parameter domains.

We first present some relations deduced from Property 2 that must be verified by any parameter of the maximum. We then give some simple results for the case where $\mathcal{P}$ is a relation of inclusion involving at most one parameter domain on each side of the inclusion. Then we introduce the use of the union, of the intersection and finally present the general case, in Theorem 1.

6.1 Characterization of parameters of the maximum

Given a set $\mathbf{G}_k$, for all $0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$, the parameter of the maximum $\vec{\gamma}_k^p$ of $\mathbf{G}_k$ for Statement $\mathtt{S}_k$ at depth p must verify Property 2. We will find now Property $\mathcal{P}^*$ that must be verified by any parameter of the maximum of any set $\mathbf{G}_k$, for all $1 \leq k \leq m$.

Construction of $\mathcal{P}^*$ According to Property 2, for $0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$, $\vec{\gamma}_k^p$ is an element of $\mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}}$ or is $-$:

$$\left(\vec{\gamma}_k^p \in \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}}\right) \vee (\vec{\gamma}_k^p = -)\,. \tag{32}$$

In particular, when $M_{\mathtt{S}_k} \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$, (4) and (5) imply that $\mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_k}$ is equal to $\{\vec{y}[1..M_{\mathtt{S}_k}]\}$ or $\emptyset$. Therefore, when $\vec{y}[1..M_{\mathtt{S}_k}] \notin \mathbf{G}_k$, $\vec{\gamma}_k^p = -$ for $M_{\mathtt{S}_k} \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$. To sum up this relation, for all $M_{\mathtt{S}_k} \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$:

$$\textbf{if } \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} = \{\vec{y}[1..M_{\mathtt{S}_k}]\} \textbf{ then } \left(\bigwedge_{M_{\mathtt{S}_k} \leq p \leq N_{\mathtt{S}_k\mathtt{R}}} \vec{\gamma}_k^p = -\right) \vee \left(\vec{\gamma}_k^p = \vec{y}[1..M_{\mathtt{S}_k}]\right). \tag{33}$$

Property $\mathcal{P}^*$ is then defined by Equations (32) and (33), for $1 \leq k \leq m$.

How much fuzziness is added? Consider a set of vectors $\vec{\gamma}_k^p$, for $1 \leq k \leq m$, $0 \leq p \leq M_{\mathtt{S}_k}$, verifying $\mathcal{P}^*$ defined by Equations (32) and (33). In order to prove that $\mathcal{P}^*$ is an exact characterization of the parameters of the maximum, we want to exhibit $\mathbf{G}_1, \ldots, \mathbf{G}_m$ such that $\vec{\gamma}_k^p$ is a parameter of the maximum of $\mathbf{G}_k$ for Statement $\mathtt{S}_k$ at depth p, for $1 \leq k \leq m, 0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$. (Intuitively, we want to prove that *any* $\vec{\gamma}_k^p$ satisfying $\mathcal{P}^*$ *may* yield the actual exact source.) We define these sets by: $\mathbf{G}_k = \left\{\vec{\gamma}_k^p \middle| \, 0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}\right\}$, for $1 \leq k \leq m$. We try to show that, according to Property 2:

$$\vec{\gamma}_k^p = \max \mathbf{G}_k \cap \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}}\,. \tag{34}$$

For $p < \min(M_{\mathtt{S}_k}, N_{\mathtt{S}_k\mathtt{R}})$, notice that $\mathbf{L}^q_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} \cap \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} = \emptyset$ if $q \neq p$ thanks to the sequencing condition (9). Equation (32) then shows that $\mathbf{G}_k \cap \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} = \{\vec{\gamma}_k^p\}$, thus (34) is verified. For $p \geq M_{\mathtt{S}_k}$, (33) and the above remark imply (34).

Hence $\mathcal{P}^*$ as defined by (32) and (33) describes exactly the set of the parameters of the maximum of all possible sets, for Statement $\mathtt{S}_k$ at depth p, for $1 \leq k \leq m, 0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$.

6.2 Inclusion between two parameter domains

Suppose now that Property $\mathcal{P}$ on the parameter domains is

$$\mathbf{D}_{\mathtt{S}_i}(\vec{y})_{|\min(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})} \cap \mathbf{A}_i(\vec{y}) \subseteq \mathbf{D}_{\mathtt{S}_j}(\vec{y})_{|\min(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})} \cup \mathbf{A}_j(\vec{y}),$$

where $\mathbf{A}_i(\vec{y})$ and $\mathbf{A}_j(\vec{y})$ are two polyhedra, of dimension $M = \min(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})$. Let us consider all sets $\mathbf{G}_i, \mathbf{G}_j$ verifying $\mathcal{P}$ and such that the dimension of the vectors of $\mathbf{G}_i$ (resp. $\mathbf{G}_j$) is $M_{\mathtt{S}_i}$ (resp. $M_{\mathtt{S}_j}$). Let $\vec{\gamma}_i^p$ and $\vec{\gamma}_j^p$ be the respective parameters of the maximum for Statements $\mathtt{S}_i$ and $\mathtt{S}_j$ at depth p. The general expression of $\mathcal{P}$ is:

$$\mathcal{P}(\mathbf{G}_i, \mathbf{G}_j) = (\mathbf{G}_{i|M} \cap \mathbf{A}_i(\vec{y})) \subseteq (\mathbf{G}_{j|M} \cap \mathbf{A}_j(\vec{y})).$$

Construction of $\mathcal{P}^*$ Let us try to find a necessary condition for $\vec{\gamma}_i^p$ and $\vec{\gamma}_j^q$ to be parameters of the maximum of $\mathbf{G}_i$ at depth p and of $\mathbf{G}_j$ at depth q, respectively, for all $0 \leq p \leq N_{\mathtt{S}_i\mathtt{R}}, 0 \leq q \leq N_{\mathtt{S}_j\mathtt{R}}$. According

to 6.1, Equations (32) and (33) are verified by $\vec{\gamma}_i^p$ and $\vec{\gamma}_j^q$. Besides, for $0 \leq p \leq N_{\mathtt{S}_i\mathtt{R}}, 0 \leq q \leq N_{\mathtt{S}_j\mathtt{R}}$, if $\vec{\gamma}_i^p[1..M] \in \mathbf{L}^q_{\mathtt{S}_j}(\vec{y})_{|M} \cap \mathbf{L}^p_{\mathtt{S}_i}(\vec{y})_{|M} \cap \mathbf{A}_i(\vec{y})$, then either $\vec{\gamma}_i^p[1..M] \in \mathbf{A}_j(\vec{y})$ or, thanks to Property 2:

$$\begin{array}{rcl}
\vec{\gamma}_i^p[1..M] & = & \max \mathbf{G}_{i|M} \cap \mathbf{A}_i(\vec{y}) \cap \mathbf{L}^q_{\mathtt{S}_j}(\vec{y})_{|M} \cap \mathbf{L}^p_{\mathtt{S}_i}(\vec{y})_{|M} \\
 & & \Downarrow \text{Property } \mathcal{P} \text{ on } \mathbf{G}_i \text{ and } \mathbf{G}_j, \text{ and } \vec{\gamma}_i^p[1..M] \notin \mathbf{A}_j(\vec{y}) \\
 & \lessapprox & \max\left(\mathbf{G}_j \cap \mathbf{L}^q_{\mathtt{S}_j}(\vec{y})_{|M_{\mathtt{S}_j}}\right)_{|M} \\
 & & \Downarrow \text{Property 2} \\
 & \lessapprox & \vec{\gamma}_j^q[1..M].
\end{array}$$

When $M_{\mathtt{S}_i} > M_{\mathtt{S}_j}$, this is equivalent to $\vec{\gamma}_i^p[1..M_{\mathtt{S}_j}] \lessapprox \vec{\gamma}_j^q$, otherwise: $\vec{\gamma}_i^p \lessapprox \vec{\gamma}_j^q[1..M_{\mathtt{S}_i}]$.

Thus, if $\mathcal{P}$ is defined by $\mathcal{P}(\mathbf{G}_i, \mathbf{G}_j) = \mathbf{G}_{i|M} \cap \mathbf{A}_i(\vec{y}) \subseteq \mathbf{G}_{j|M} \cap \mathbf{A}_j(\vec{y}))$ then $\mathcal{P}^*$ can be defined by the conjunction of (32), (33) and, for all $0 \leq p \leq N_{\mathtt{S}_i\mathtt{R}}, 0 \leq q \leq N_{\mathtt{S}_j\mathtt{R}}$:

$$\textbf{if } \vec{\gamma}_i^p[1..M] \in \mathbf{L}^p_{\mathtt{S}_i}(\vec{y})_{|M} \cap \mathbf{L}^q_{\mathtt{S}_j}(\vec{y})_{|M} \cap \mathbf{A}_i(\vec{y}) \textbf{ then } \vec{\gamma}_i^p[1..M] \in \mathbf{A}_j(\vec{y}) \vee \vec{\gamma}_i^p[1..M] \lessapprox \vec{\gamma}_j^q[1..M]. \tag{35}$$

Notice that thanks to the sequencing predicate (9), when p or q is lower than $\min(M, N_{\mathtt{S}_i\mathtt{R}}, N_{\mathtt{S}_j\mathtt{R}})$ and $p \neq q$, then $\mathbf{L}^p_{\mathtt{S}_i}(\vec{y})_{|M} \cap \mathbf{L}^q_{\mathtt{S}_j}(\vec{y})_{|M} = \emptyset$.

How much fuzziness is added? Let us now pick a set of parameters $\vec{\gamma}_k^p$, $k = 1..m, p = 0..N_{\mathtt{S}_k\mathtt{R}}$ verifying $\mathcal{P}^*$ defined by (32),(33) and (35). In order to prove that no fuzziness is added, we want to exhibit $(\mathbf{G}_1, \ldots, \mathbf{G}_m)$ such that $\mathcal{P}(\mathbf{G}_i, \mathbf{G}_j)$ is true and $\vec{\gamma}_k^p$ is the parameter of the maximum of $\mathbf{G}_k$ for Statement $\mathtt{S}_k$ at depth p, for all $1 \leq k \leq m, 0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}$.

Let us define some new vectors $\vec{\gamma}_{ij}^p$ of dimension $M_{\mathtt{S}_j}$, for all $0 \leq p \leq N_{\mathtt{S}_i\mathtt{R}}$:

$$\left\{\begin{array}{l}
\vec{\gamma}_{ij}^p[1..M] = \vec{\gamma}_i^p[1..M] \\
\vec{\gamma}_{ij}^p[M+1..M_{\mathtt{S}_j}] = \min_{q \in 0..N_{\mathtt{S}_j\mathtt{R}}} \vec{\gamma}_j^q[M+1..M_{\mathtt{S}_j}]
\end{array}\right.$$

If $\vec{\gamma}_i^p = -$ then $\vec{\gamma}_{ij}^p = -$.

Let us define the sets $\mathbf{G}_k$ by:

$$\left\{\begin{array}{l}
\mathbf{G}_k = \{\vec{\gamma}_k^p \,|\, 0 \leq p \leq N_{\mathtt{S}_k\mathtt{R}}\} \text{ for } k \neq j, \\
\mathbf{G}_j = \{\vec{\gamma}_j^q \,|\, 0 \leq q \leq N_{\mathtt{S}_j\mathtt{R}}\} \cup \{\vec{\gamma}_{ij}^q \,|\, 0 \leq q \leq N_{\mathtt{S}_i\mathtt{R}}, \vec{\gamma}_{ij}^q[1..M] \in \mathbf{A}_i(\vec{y}), \vec{\gamma}_{ij}^q[1..M] \notin \mathbf{A}_j(\vec{y})\}.
\end{array}\right.$$

These sets verify the two conditions:

- $\mathbf{G}_{i|M} \cap \mathbf{A}_i(\vec{y}) \subseteq \mathbf{G}_{j|M} \cup \mathbf{A}_j(\vec{y})$
- $\vec{\gamma}_k^p$ is a parameter of the maximum of $\mathbf{G}_k$

The proof follows the guidelines of the proof given in 6.1.

Therefore the conjunction of (32),(33) and (35) defines exactly the set of the parameters of the maximum of all sets $\mathbf{G}_1, \ldots, \mathbf{G}_m$ verifying $\mathbf{G}_{i|M} \cap \mathbf{A}_i(\vec{y}) \subseteq \mathbf{G}_{j|M} \cup \mathbf{A}_j(\vec{y})$. No fuzziness is added when deriving $\mathcal{P}^*$ from $\mathcal{P}$.

Particular cases The properties on the parameters of the maximum corresponding to relations on the parameter domains defined by:

$$\mathbf{A}'_k(\vec{y}) \subseteq \mathbf{D}_{\mathtt{S}_k}(\vec{y}) \cup \mathbf{A}_k(\vec{y}) \text{ or } \mathbf{D}_{\mathtt{S}_k}(\vec{y}) \cap \mathbf{A}'_k(\vec{y}) \subseteq \mathbf{A}_k(\vec{y}),$$

where $\mathbf{A}_k(\vec{y})$ and $\mathbf{A}'_k(\vec{y})$ are sets of vector size $M_{\mathtt{S}_k}$ defined by affine constraints, can be derived in the same way as above.

The property $\mathcal{P}^*$ corresponding to $\mathbf{A}'_k(\vec{y}) \subseteq \mathbf{D}_{\mathtt{S}_k}(\vec{y}) \cup \mathbf{A}_k(\vec{y})$ is defined by (32), (33) and:

$$\textbf{if } \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} \cap \mathbf{A}'_k(\vec{y}) \neq \emptyset \textbf{ then } \max \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} \cap \mathbf{A}'_k(\vec{y}) \in \mathbf{A}_k(\vec{y}) \vee \max \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} \cap \mathbf{A}'_k(\vec{y}) \underline{\ll} \vec{\gamma}^p_k,$$

and the property $\mathcal{P}^*$ corresponding to $\mathbf{D}_{\mathtt{S}_k}(\vec{y}) \cap \mathbf{A}'_k(\vec{y}) \subseteq \mathbf{A}_k(\vec{y})$ is defined by (32), (33) and:

$$\textbf{if } \vec{\gamma}^p_k \in \mathbf{L}^p_{\mathtt{S}_k}(\vec{y})_{|M_{\mathtt{S}_k}} \cap \mathbf{A}'_k(\vec{y}) \textbf{ then } \vec{\gamma}^p_k \in \mathbf{A}_k(\vec{y}).$$

6.3 Union of parameter domains

We now extend the previous results to properties using the union operator on both sides of the inclusion. As $\cup_i F_i \subseteq \cup_j F_j$ is equivalent to $F_i \subseteq \cup_j F_j, \forall i$, we will consider the following property $\mathcal{P}$ on the parameter domains:

$$\mathbf{D}_{\mathtt{S}_i}(\vec{y})_{|M} \cap \mathbf{A}_i(\vec{y}) \subseteq \bigcup_{j \in J} \mathbf{D}_{\mathtt{S}_j}(\vec{y})_{|M} \cup \mathbf{A}(\vec{y}),$$

where $M = \min(M_{\mathtt{S}_i}, \min_{j \in J}(M_{\mathtt{S}_j}))$, $\mathbf{A}_i(\vec{y})$ and $\mathbf{A}(\vec{y})$ are two sets defined by linear constraints of vector dimension M and J is a set of indices not including i. Let us consider all sets $\mathbf{G}_i$ and $\mathbf{G}_j, j \in J$ verifying $\mathcal{P}$ and such that the dimension of the vectors of $\mathbf{G}_i$ (resp. $\mathbf{G}_j$) is $M_{\mathtt{S}_i}$ (resp. $M_{\mathtt{S}_j}$). Let $\vec{\gamma}^p_i$ and $\vec{\gamma}^p_j$ be the respective parameters of the maximum for Statements $\mathtt{S}_i$ and $\mathtt{S}_j$ at depth p.

Construction of $\mathcal{P}^*$ As in 6.2 the parameters $\vec{\gamma}^p_k$ are constrained by (32) and (33). Moreover, it can be shown that, for all $0 \leq p \leq N_{\mathtt{S}_i\mathtt{R}}, 0 \leq q_j \leq N_{\mathtt{S}_j\mathtt{R}}$,

$$\textbf{if } \vec{\gamma}^p_i[1..M] \in \mathbf{L}^p_{\mathtt{S}_i}(\vec{y})_{|M} \bigcap_{j \in J} \mathbf{L}^{q_j}_{\mathtt{S}_j}(\vec{y})_{|M} \cap \mathbf{A}_i(\vec{y}) \textbf{ then } \vec{\gamma}^p_i[1..M] \in \mathbf{A}(\vec{y}) \bigvee_{j \in J} \vec{\gamma}^p_i[1..M] \underline{\ll} \vec{\gamma}^{q_j}_j[1..M]. \tag{36}$$

Thus if $\mathcal{P}$ is defined by $\mathcal{P}(\mathbf{G}_i, (\mathbf{G}_j)_{j \in J}) = \mathbf{G}_{i|M} \cap \mathbf{A}_i \subseteq \bigcup_{j \in J} \mathbf{G}_{j\,|M} \cup \mathbf{A}(\vec{y})$ then $\mathcal{P}^*$ is defined by the conjunction of the equations (32), (33) and (36).

How much fuzziness is added? It can be shown in the same manner as in 6.2 that $\mathcal{P}^*$ defines exactly the set of the parameters of the maximum of all the sets $\mathbf{G}_i, \mathbf{G}_j, j \in J$ verifying $\mathcal{P}$.

This property is exactly what is needed to express the fact that at least one branch of a conditional is taken each time the conditional is executed.

Particular case When $\mathcal{P}$ is defined on the parameter domains by:

$$\mathbf{A}(\vec{y}) \subseteq \bigcup_{j \in J} \mathbf{D}_{\mathtt{S}_j}(\vec{y})_{|\min_{j \in J} M_{\mathtt{S}_j}} \cup \mathbf{A}'(\vec{y}),$$

then the corresponding property on the parameters of the maximum is defined by (32), (33) and:

$$\textbf{if } \bigcap_{j \in J} \mathbf{L}^{q_j}_{\mathtt{S}_j}(\vec{y})_{|\min_{j \in J} M_{\mathtt{S}_j}} \cap \mathbf{A}(\vec{y}) \neq \emptyset \textbf{ then } \vec{\gamma} \in \mathbf{A}'(\vec{y}) \bigvee_{j \in J} \vec{\gamma} \underline{\ll} \vec{\gamma}^{q_j}_j[1..\min_{j \in J} M_{\mathtt{S}_j}],$$

where $\vec{\gamma}$ stands for $\max \bigcap_{j \in J} \mathbf{L}^{q_j}_{\mathtt{S}_j}(\vec{y})_{|\min_{j \in J} M_{\mathtt{S}_j}} \cap \mathbf{A}(\vec{y})$.

6.4 Intersection of parameter domains

Let us examine now relations involving intersections of parameter domains. This situation occurs when we want to express the fact that exactly one branch of a conditional is taken each time the conditional is executed.

We first examine the particular property $\mathbf{D}_{\mathtt{S}_i}(\vec{y})_{|\min(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})} \cap \mathbf{D}_{\mathtt{S}_{i_2}}(\vec{y})_{|\min(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})} = \emptyset$. Let us consider all the sets $\mathbf{G}_i$ and $\mathbf{G}_j$ respectively of vector size $M_{\mathtt{S}_i}$ and $M_{\mathtt{S}_j}$ verifying this property. Let M denote $\min(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})$.

Construction of $\mathcal{P}^*$ Clearly, if $\vec{\gamma}_i^p$ and $\vec{\gamma}_j^p$ are the parameters of the maximum of $\mathbf{G}_i$ and $\mathbf{G}_j$ then $\vec{\gamma}_i^p[1..M] \neq \vec{\gamma}_j^q[1..M]$. $\mathcal{P}^*$ will then be defined by this equation and by (32) and (33).

How much fuzziness is added? The above definition of $\mathcal{P}^*$ defines exactly the parameters of the maximum of all the sets $\mathbf{G}_i$ and $\mathbf{G}_j$ such that $\mathbf{G}_{i|M} \cap \mathbf{G}_{j|M} = \emptyset$. Indeed, given $\vec{\gamma}_i^p$ and $\vec{\gamma}_j^q$, for all $0 \leq p \leq N_{\mathtt{S}_i\mathtt{R}}, 0 \leq q \leq N_{\mathtt{S}_j\mathtt{R}}$, verifying $\mathcal{P}^*$, the sets $\{\vec{\gamma}_i^q | \, 0 \leq q \leq N_{\mathtt{S}_i\mathtt{R}}\}$ and $\{\vec{\gamma}_j^q | \, 0 \leq q \leq N_{\mathtt{S}_j\mathtt{R}}\}$ have an empty intersection and $\vec{\gamma}_i^p$ (resp. $\vec{\gamma}_j^p$) is the parameter of the maximum of $\mathbf{G}_i$ (resp. $\mathbf{G}_j$) for Statement $\mathtt{S}_i$ (resp. $\mathtt{S}_j$) at depth p (for the proof, see Section 6.1)

For the general case, we define three new sets:

- $\mathbf{G}_{i\cap j} = \mathbf{G}_{i|M_{\max}} \cap \mathbf{G}_{j|M_{\max}}$,
- $\mathbf{G}_{i-j} = \mathbf{G}_i - \mathbf{G}_{j|M_{\mathtt{S}_i}}$ and
- $\mathbf{G}_{j-i} = \mathbf{G}_j - \mathbf{G}_{i|M_{\mathtt{S}_j}}$,

with $M_{\max} = \max(M_{\mathtt{S}_i}, M_{\mathtt{S}_j})$. We have $\mathbf{G}_i = \mathbf{G}_{i-j} \cup \mathbf{G}_{i\cap j|M_{\mathtt{S}_i}}$ and $\mathbf{G}_j = \mathbf{G}_{j-i} \cup \mathbf{G}_{i\cap j|M_{\mathtt{S}_j}}$. Moreover, each of the three new sets is disjointed from the two others. Therefore, we can replace a property using $\mathbf{G}_i$ and $\mathbf{G}_j$ by an equivalent property using $\mathbf{G}_{i-j}$,$\mathbf{G}_{j-i}$ and $\mathbf{G}_{i\cap j}$. Doing repeatedly such transformations on Property $\mathcal{P}$, we will eventually get a property using only relations of inclusion between unions of sets and relations of empty intersections of sets. Both relations can be transformed into relations on parameters of the maximum without adding fuzziness.

6.5 General relations

This theorem sums up the results obtained in this section and gives the steps for constructing Property $\mathcal{P}^*$ from a Property $\mathcal{P}$ verifying the hypotheses stated in 5.1.

Theorem 1 *For every property $\mathcal{P}$ on parameter domains in the class of properties defined in 5.1, the corresponding $\mathcal{P}^*$ is defined by a union of polyhedra which can be built from $\mathcal{P}$ and therefore the set of sources can be exactly computed.*

> **Proof** We first consider properties $\mathcal{P}$ with at most one relation, simplified with (30) and (31). All the intersections between parameter sets are transformed into new sets thanks to Section 6.4. The new property gives a Property $\mathcal{P}^*$ by using the results of Section 6.1 and 6.3. $\mathcal{P}^*$ is defined as a conjunction or disjunction of linear terms on the parameters of the maximum.
>
> Concerning the application of a monotone increasing function t to parameter domains, the monotony preserves the parameters of the maximum: if $\vec{\gamma}_k^p$ is the parameter of the maximum of $\mathbf{G}_k$ for $\mathtt{S}_k$ at depth p then $t(\vec{\gamma}_k^p)$ is the parameter of the maximum of $t(\mathbf{G}_k)$ for $\mathtt{S}_k$ at depth p. Therefore the previous results apply easily to parameter domains transformed by linear monotone increasing functions.
>
> Finally, it can be easily shown that when Property $\mathcal{P}$ is a conjunction of several relations of inclusion, Property $\mathcal{P}^*$ is the conjunction of the properties on the parameters of the maximum corresponding to each relation.

6.6 Example

We present thereafter the formal computation of the source of Statement `R` of Program `E2` (see Section 3.4). We recall the property $\mathcal{P}$ on the parameter domains:

$$\mathcal{P}(\mathbf{D}_{\mathtt{S}_1}, \mathbf{D}_{\mathtt{S}_2}) = (\mathbf{D}_{\mathtt{S}_1} \cap \mathbf{D}_{\mathtt{S}_2} = \emptyset) \wedge (\mathbf{D}_{\mathtt{S}_1} \cup \mathbf{D}_{\mathtt{S}_2} = \mathbb{Z}).$$

Note that in this case the parameter domains do not depend on y, they are sets of scalars and $N_{\mathtt{S}_1\mathtt{R}} = N_{\mathtt{S}_2\mathtt{R}} = 0$. From $\mathbf{D}_{\mathtt{S}_1} \cap \mathbf{D}_{\mathtt{S}_2} = \emptyset$ and Section 6.4, we deduce one conjunct of $\mathcal{P}^*$: $\gamma_1 \neq \gamma_2$. From Section 6.1, we have the relations: $\gamma_1 \in \mathbf{L}^0_{\mathtt{S}_1}(y) \vee \gamma_1 = -$, $\gamma_2 \in \mathbf{L}^0_{\mathtt{S}_2}(y) \vee \gamma_2 = -$. Relation (33) is obviously verified since $M_{\mathtt{S}_1} = M_{\mathtt{S}_2} = 1 > 0 = N_{\mathtt{S}_1\mathtt{R}} = N_{\mathtt{S}_2\mathtt{R}}$. The relation $\mathbf{D}_{\mathtt{S}_1} \cup \mathbf{D}_{\mathtt{S}_2} = \mathbb{Z}$ can be written $\mathbb{Z} \subseteq \mathbf{D}_{\mathtt{S}_1} \cup \mathbf{D}_{\mathtt{S}_2}$. Applying the result of the particular case of Section 6.3 with $\mathbf{A}(\vec{y}) = \mathbb{Z}$ and $\mathbf{A}'(\vec{y}) = \emptyset$, we get the relation:

$$\textbf{if } \mathbf{L}^0_{\mathtt{S}_1}(y) \cap \mathbf{L}^0_{\mathtt{S}_2}(y) \neq \emptyset \textbf{ then } \bigvee_{1 \leq q \leq 2} \max \mathbf{L}^0_{\mathtt{S}_1}(y) \cap \mathbf{L}^0_{\mathtt{S}_2}(y) \underline{\ll} \gamma_q.$$

Therefore, $\mathcal{P}^*$ is defined by:

$$\begin{aligned} \mathcal{P}^*(\gamma_1, \gamma_2) \quad = \quad & (\gamma_1 \neq \gamma_2) \ \wedge \ (\gamma_1 \in \mathbf{L}^0_{\mathtt{S}_1}(y) \vee \gamma_1 = -) \ \wedge \ (\gamma_2 \in \mathbf{L}^0_{\mathtt{S}_2}(y) \vee \gamma_2 = -) \\ & \wedge \ (\textbf{if } \mathbf{L}^0_{\mathtt{S}_1}(y) \cap \mathbf{L}^0_{\mathtt{S}_2}(y) \neq \emptyset \textbf{ then } \bigvee_{1 \leq q \leq 2} \max \mathbf{L}^0_{\mathtt{S}_1}(y) \cap \mathbf{L}^0_{\mathtt{S}_2}(y) \underline{\ll} \gamma_q). \end{aligned}$$

As $\mathbf{L}^0_{\mathtt{S}_1}(y) = \mathbf{L}^0_{\mathtt{S}_2}(y) = \{x | \ 1 \leq x \leq n\}$ and we assumed that $1 \leq n$, $\mathbf{L}^0_{\mathtt{S}_1}(y) \cap \mathbf{L}^0_{\mathtt{S}_2}(y)$ is not empty and its maximum is n. We may rewrite $\mathcal{P}^*$ as:

$$\mathcal{P}^*(\gamma_1, \gamma_2) \quad = \quad (\gamma_1 \neq \gamma_2) \ \wedge \ (1 \leq \gamma_1 \leq n \vee \gamma_1 = -) \ \wedge \ (1 \leq \gamma_2 \leq n \vee \gamma_2 = -) \ \wedge \ (n \leq \gamma_1 \vee n \leq \gamma_2).$$

It can be shown easily that as a consequence $(\gamma_1 = n \wedge \gamma_2 < n) \vee (\gamma_1 < n \wedge \gamma_2 = n)$.

For each clause of $\mathcal{P}^*$ in which there is a conditional or disjunction, there will be two different contexts for the computation of the source. Hence the quast of the source begins with:

> **if** $\gamma_1 = n \wedge \gamma_2 < n$
>
> **then** Plug in the result given by PIP in context $\gamma_1 = n, \gamma_2 < n$.
>
> **else** Plug in the result given by PIP in context $\gamma_1 < n, \gamma_2 = n$

The parametric sets of candidates are $\mathbf{Q}^{*0}_{\mathtt{S}_1}(y, \alpha) = \mathbf{Q}^{*0}_{\mathtt{S}_2}(y, \alpha) = \{x | \ 1 \leq x \leq n, x = \alpha\}$. The parametric direct dependences are:

$$\vec{K}^{*0}_{\mathtt{S}_1}(y, \alpha) = \vec{K}^{*0}_{\mathtt{S}_2}(y, \alpha) = \textbf{ if } 1 \leq \alpha \leq n \textbf{ then } \alpha \textbf{ else } -.$$

Hence the parametric source, after simplification, is:

$$\textbf{if } \gamma_1 = n \wedge \gamma_2 < n \textbf{ then } \langle \mathtt{S}_1, n \rangle \textbf{ else } \langle \mathtt{S}_2, n \rangle,$$

and the fuzzy source is: $\Sigma(y) = \{\langle \mathtt{S}_1, n \rangle, \langle \mathtt{S}_2, n \rangle\}$. Therefore no previous value of `s` can reach Statement `R`.

7 Iterative analysis

The key remark in this section is that two values of the same variable at two different steps of the execution are equal if they have the same *source*. Thanks to this remark, we will show that we may go one step further in data-flow analyses. That is, that the result of a first application of the FADA analysis may in turn help a second application in deriving a more precise result.

To see this, suppose that the same array occurs in the left hand side of two statements, with differing variables as subscripts. These variables are supposed not to depend linearly on induction variables. Dataflow analyses do not make assumptions on the values of variables, and therefore are not able to give the exact source. We may, however, try to prove that whatever the values of these variables, these values are equal. As hinted above, we may apply a dataflow analysis on the subscripting variables themselves, thus iterating the overall process of the analysis. Similarly, two constraints that are the same function but appear at different places in the program have the same value if the variables they use are the same and have the same values.

Therefore, the purpose of iterative analysis is to find relational properties between the non-linear constraints appearing in the existence predicates (7) and in the conflicting access constraints (8) of different write statements. This method may use the results of dataflow analysis on the variables of the non-linear constraints so as to find more accurate relations. As this dataflow analysis can be fuzzy, the method can then be applied once more and eventually the fuzziness will be reduced by successive analyses. This method finds some relations between the parameter sets and then extends these relations to the real domains of parameters.

7.1 Variables in non-linear constraints

To formalize the previous paragraph, let c_h and $c_{h'}$ be two non-linear constraints. Our purpose is to decide whether the value of c_h at operation τ is the same as the value of $c_{h'}$ at operation ϕ:

$$c_{h\,\tau} = c_{h'\,\varphi}. \tag{37}$$

So far, we have defined constraints as functions of $\vec{y}$ and of the iteration vector of the surrounding loops. As a matter of fact, a constraint c_h depends on variables that are functions of the iteration vector. Let $\mathbf{V}(h) = (v_1^h, \ldots, v_{l_h}^h)$ denote the list of the variables appearing in the expression of c_h. At operation φ, the value of these variables is denoted $\mathbf{V}(h)_\varphi$.

The following result is used in the sequel:

Property 3 *If c_h and $c_{h'}$ define the same function (perhaps because they are syntactically equal), Equation (37) holds if $\mathbf{V}(h) = \mathbf{V}(h')$ and if the sources of $\mathbf{V}(h)$ at operation τ and $\mathbf{V}(h')$ at operation φ are the same.*

Indeed, if these variables have the same exact source, then they have the same value. In the case of fuzzy sources, two variables have the same source if they have the same parameter of the maximum. This equality between parameters of the maximum can be obtained by comparing the parameter domains for both read statements, and this may need another FADA.

7.2 Relations on parameter sets

The iterative analysis yields properties on parameter domains, as in 5.2. So as to produce more precise results, we are trying to find relations on the parameter sets and then extend them to parameter domains. We give thereafter the list of the relations that are detected between two parameter sets $\mathbf{P}_h$ and $\mathbf{P}_{h'}$ and a description of their detection.

Notice that comparing two sets of parameters is useless if the corresponding parameter domains cannot themselves be compared. This occurs when a parameter domain is defined w.r.t. a non-linear constraint which does not appear anywhere else, or w.r.t. a variable which does not appear in any set of parameters of the other domain.

7.2.1 Partial equality

Equality $\mathbf{P}_h = \mathbf{P}_{h'}$ holds if $\mathbf{V}(h) = \mathbf{V}(h')$ and if the value of $\mathbf{V}(h)$ at operation $\langle \mathtt{T}_h, \vec{x}[1..N_h] \rangle$ and the value of $\mathbf{V}(h')$ at operation $\langle \mathtt{T}_{h'}, \vec{x}[1..N_{h'}] \rangle$ have the same source. Detecting this case consists in the computation and comparison of the sources of $\mathbf{V}(h)$ and $\mathbf{V}(h')$.

Partial equality This is a more general case: only some quast leaves in the sources of $\mathbf{V}(h), \mathbf{V}(h')$ are equal. The context then takes into account the different conditions from the branches of the quast for which these leaves are actually sources. Let $\mathbf{F}$ denote the set of iteration vectors verifying these conditions. Then the partial equality corresponds to the equality: $\mathbf{P}_h \cap \mathbf{F} = \mathbf{P}_{h'} \cap \mathbf{F}$.

7.2.2 Image of a parameter set

We now generalize the equality of parameter sets to the case where one parameter set is equal to the image of the second set by a function.

Our purpose is to detect cases in which the value of a non-linear constraint c_h at a given step of the execution is equal to the value of another constraint $c_{h'}$ at a previous step. That is, we are looking for a function $\vec{e}$ such that:

$$c_h\langle T_k, \vec{x}[1..N_h]\rangle = c_{h'}\langle T_k, \vec{e}(\vec{x}[1..N_h])\rangle$$

Relations between a set and the image of a set can thus be detected. So as to verify the hypotheses of 5.1 on the relations between parameter domains, $\vec{e}$ has to be a monotone increasing affine function with respect to loop counters and structure parameters. Note also that we may have partial equality of a set of parameters and the image of another set by function $\vec{e}$.

Analyzing the following example brings into play partial equality and the image of a parameter set by a function.

```
S0: z=0
    do x=1,n
S1:    a(z)=x
S2:    z=f(x)
S3:    a(z)=0
    end do
    do y=1,n
R :    r=a(y)
    end do
```

Our aim is to find the source of `a(y)` in operation $\langle \mathtt{R}, y\rangle$. For the two candidate sources $\mathtt{S}_1$ and $\mathtt{S}_3$, parameter domains are $\mathbf{D}_{\mathtt{S}_1}(x,y) = \{x | \mathtt{z}_{\langle \mathtt{S}_1, x\rangle} = y\}$ and $\mathbf{D}_{\mathtt{S}_3}(x,y) = \{x | \mathtt{z}_{\langle \mathtt{S}_3, x\rangle} = y\}$. The constraints are the same and the subscripting expressions are both equal to variable z. We will thus first apply a dataflow analysis to z.

First iterate As far as Statement $\mathtt{S}_1$ is concerned, the source of z is

$$\textbf{if } x \geq 2 \textbf{ then } \langle \mathtt{S}_2, x-1\rangle \textbf{ else } \langle \mathtt{S}_0, []\rangle.$$

For Statement $\mathtt{S}_3$, the source is $\langle \mathtt{S}_2, x\rangle$. Let f be the function: $f(x) = x - 1$. We then have:

$$f(\mathbf{G}_1 \cap \{i | 2 \leq x \leq n\}) = \mathbf{G}_3 \cap \{x | 1 \leq x \leq n-1\}.$$

We thus have the additional environment:

$$\textbf{if } 2 \leq x \leq n \textbf{ then } \beta_3 = \beta_1 - 1. \tag{38}$$

Second iterate The set of candidate sources for Statement `R` from Statement $\mathtt{S}_1$ is:

$$\mathbf{Q}^{*0}_{\mathtt{S}_1}(y,\alpha) = \{x | 1 \leq x \leq n, x = \alpha, x = y\},$$

whose maximum is: $\vec{K}^{*0}_{\mathtt{S}_1}(y,\beta_1) = \textbf{if } \beta_1 = y \textbf{ then } \beta_1 \textbf{ else } -$. The direct dependence from Statement $\mathtt{S}_3$ is similar. From (38) we can compute the source of `a(y)`:

$$\begin{array}{rcl}
\sigma(y) & = & \max\limits_{\prec} \left(\begin{array}{l} \textbf{if } \beta_1 = y \textbf{ then } \langle \mathtt{S}_1, \beta_1 \rangle \textbf{ else } -, \\ \textbf{if } \beta_3 = y \textbf{ then } \langle \mathtt{S}_3, \beta_3 \rangle \textbf{ else } - \end{array} \right) \\
 & = & \left| \begin{array}{l} \textbf{if } 2 \leq \beta_1 \wedge \beta_1 = y \\ \textbf{then } \max_{\prec} (\langle \mathtt{S}_1, \beta_1 \rangle, \langle \mathtt{S}_3, \beta_1 - 1 \rangle) \\ \quad \left| \begin{array}{l} \textbf{if } \beta_1 = y = 1 \\ \textbf{then } \langle \mathtt{S}_1, \beta_1 \rangle \end{array} \right. \\ \textbf{else} \quad \left| \begin{array}{l} \textbf{if } \beta_3 = y = n \\ \textbf{then } \langle \mathtt{S}_3, n \rangle \end{array} \right. \\ \quad\quad \textbf{else } - \end{array} \right. \\
\Sigma(j) & = & \{-, \langle \mathtt{S}_1, 1 \rangle, \langle \mathtt{S}_3, n \rangle\} \cup \{ \langle \mathtt{S}_1, \gamma_1 \rangle \mid 2 \leq \gamma_1 \leq n \}.
\end{array}$$

7.2.3 Composition of a constraint with an affine function

Let us now examine a more general case where constraints c_h and $c_{h'}$ are different but there exists some function e such that $c_h = c_{h'} \circ e$. From a practical point of view, c_h and $c_{h'}$ have to be affine functions of the variables of the program. All possible affine functions e verifying this equality are found by Gaussian elimination.

So as to reuse previous results, our aim is to find a function f such that

$$e(\mathbf{V}(h)_{\langle T_h, \vec{x}[1..N_h] \rangle}) = \mathbf{V}(h)_{\langle T_h, f(\vec{x}[1..N_h]) \rangle}.$$

Since this expression is the formal definition of a recurrence as given by Redon [17], this problem boils down to the detection of a recurrence on $\mathbf{V}(h)$. Notice that detecting recurrences requires the computation of a dataflow graph, thus additional iterative analyses and recurrence detections may have to be applied.

We now have the following equality:

$$c_h(\mathbf{V}(h)_{\langle T_h, \vec{x}[1..N_h] \rangle}) \;=\; c_{h'}(e(\mathbf{V}(h)_{\langle T_h, \vec{x}[1..N_h] \rangle})) \;=\; c_{h'}(\mathbf{V}(h)_{\langle T_h, f(\vec{x}[1..N_h]) \rangle}).$$

We then try to find a relation between $\mathbf{V}(h)_{\langle T_h, f(\vec{x}[1..N_h]) \rangle}$ and $\mathbf{V}(h')_{\langle T'_h, \vec{x}[1..N_{h'}] \rangle}$. Such a relation is a partial equality or a property on the image of a set of parameters. Finding such a relation would allow us to find a relation between $c_h(\mathbf{V}(h)_{\langle T_h, \vec{x}[1..N_h] \rangle})$ and $c_{h'}(\mathbf{V}(h')_{\langle T'_h, \vec{x}[1..N_{h'}] \rangle})$.

Obviously, we can generalize this result to relations between $\mathbf{V}(h)_{\langle T_h, f^n(\vec{x}[1..N_h]) \rangle}$ and $\mathbf{V}(h')_{\langle T'_h, \vec{x}[1..N_{h'}] \rangle}$, where n is a positive integer, as illustrated below.

The following example is an application of these ideas:

```
S0: b(0)=...
    do x=1,n
S1:    b(x)=b(x)+2
S2:    if b(x)=x then a(16)=5*x
S3:    if b(x)=x+4 then a(16)=3*x
    end do
R:  z=a(16)
```

The parameter domains for direct dependences from Statements $\mathtt{S}_2$ and $\mathtt{S}_3$, respectively, are: $\mathbf{D}_{\mathtt{S}_2}([]) = \{x | \mathtt{b}_{\langle \mathtt{S}_2, x \rangle} = x\}$ and $\mathbf{D}_{\mathtt{S}_3}([]) = \{x | \mathtt{b}_{\langle \mathtt{S}_3, x \rangle} = x + 4\}$. Non-linear constraints are different: let $c_2(z,i) = z - i$, $c_3(z,i) = z - i - 4$ and $\vec{g}_{\mu,\lambda}(z,i) = (\mu z - 4 + \lambda, \mu i + \lambda)$. We have:

$$c_2(\mathtt{b}_{\langle \mathtt{S}_3, x \rangle}, x) = c_3(\vec{g}_{\mu,\lambda}(\mathtt{b}_{\langle \mathtt{S}_3, x \rangle}, x)).$$

Parameterized functions like $\vec{g}_{\mu,\lambda}$ are found by resolution of a system of linear equations, and describe the set of possible solutions.

We then seek a recurrence on z so as to eliminate $\vec{g}_{\mu,\lambda}$ and to reduce our problem to the case of an image of a domain of parameters. Recurrence detection shows that:

$$\textbf{if } x > 1 \textbf{ then } \mathtt{b}_{\langle \mathtt{S}_3, x\rangle} = \mathtt{b}_{\langle \mathtt{S}_3, x-1\rangle} + 2 \textbf{ else } \mathtt{b}_{\langle \mathtt{S}_3, 1\rangle} = \mathtt{b}_{\langle \mathtt{S}_0, []\rangle}.$$

Let us consider functions $\vec{e}(z,x) = (z-2, x-1)$ and $f(x) = x-1$. When $x > 1$, we get: $\vec{e}(\mathtt{b}_{\langle \mathtt{S}_3, x\rangle}) = (\mathtt{b}_{\langle \mathtt{S}_3, f(x)\rangle}, f(x))$. We notice that if $n = 2$ and $\mu = 1$ and $\lambda = -2$, then:

$$c_2(\vec{g}_{\mu,\lambda}(\mathtt{b}_{\langle \mathtt{S}_3, x\rangle}, x)) = c_2(\vec{e}^2(\mathtt{b}_{\langle \mathtt{S}_3, x\rangle}, x)) = c_2(\mathtt{b}_{\langle \mathtt{S}_3, f^2(x)\rangle}, f^2(x)),$$

when $x > 2$. Moreover, a dataflow analysis on b shows that $\mathtt{b}_{\langle \mathtt{S}_2, x\rangle}$ and $\mathtt{b}_{\langle \mathtt{S}_3, x\rangle}$ have the same source. We thus come down to a partial image of a domain of parameters, such that:

$$c_3(\mathtt{b}_{\langle \mathtt{S}_3, x\rangle}, x) = c_2(\mathtt{b}_{\langle \mathtt{S}_2, x-2\rangle}, x-2),$$

when $x > 2$.

This eventually allows us to prove that the write in $\mathtt{S}_2$ covers the write which occurred in $\mathtt{S}_3$ two iterations before. Thus, the sources are:

$$\{-\} \cup \{\langle \mathtt{S}_2, \gamma_2\rangle | 1 \leq \gamma_2 \leq n\} \cup \{\langle \mathtt{S}_3, \gamma_3\rangle | 1 \leq \gamma_3 \leq \min(2, n)\}.$$

Finally, note that the process of finding the source of a variable to reduce the fuzziness of the computation of another source may not terminate. Indeed, this may happen in programs using for instance `a(a(x))`. Such a case can be detected by building a graph of the analyses. There is an edge from the analysis of `a` in statement `S` to the analysis of `b` in statement `T` iff `S` is a write into `b` where `a` is used in a non linear constraint. Analyses should be carried according to a linearization of this graph. Cycles in this graph indicate potential non terminating analyses. It remains to see if one can expect to find a fixpoint in such cases.

8 Related Work

Work on non-linear constraints in dependence analysis can be divided in two classes. In the first one, the dependence analyzer uses a limited amount of mathematical knowledge to decide whether dependences exist. In the second class, to which this paper belongs, no such knowledge is needed, but the results are less precise.

An example of the first approach is found in Dumay PhD thesis [8] where techniques borrowed from formal algebra are used to prove or disprove memory based dependences. With some information on polynomials and exponentials and the computation of derivatives, Dumay's system is able to parallelize familiar kernels like bloc matrix product or the Fast Fourier Transform.

Using a different approach, Maslov noticed in [13] that the set of integer points in a convex body may sometime be defined by linear inequalities. For instance $xy \geq 1, x \geq 0, y \geq 0$ is equivalent to $x \geq 1, y \geq 1$. There are two difficulties with this method:

- The number of necessary linear constraints may grow very fast or even becomes infinite (consider e.g. $xy \geq z$).
- If the non-linear relation defines a non-convex body, one has to introduce disjunction, which complicates the subsequent analysis.

Still another example of this class of algorithms is the work of Masdupuy [12] in which modulo constraints are handled exactly.

In the other class of methods, one uses syntactical information only. This may include the structure of the original program, the shape of subscript expressions and the list of variables which occur in them.

The work nearest to our own in that direction is the one by Pugh and Wonnacott [15, 16]. To compare these two approaches, one must recall that the engine behind Pugh's Array Dataflow Analysis is the Omega calculator, a logical formula simplifier. The formulae which are handled by this system are Number Theory

formulae with multiplication and division omitted and constitute what is known as Presburger arithmetic. It is easy to see that this is enough as long as one considers static control programs only. To handle more general situations, the authors introduce uninterpreted function symbols. For instance, the iteration domain of `S` in the following program:

```
    do i = 1,n
       do w = 1 while ...
S : ....
```

is given by: $1 \leq i \leq n, 1 \leq w \leq f(i)$, where f is an uninterpreted function. Now, while Presburger arithmetic is decidable, adding uninterpreted functions renders it equivalent to full Number Theory, which is undecidable. The Omega calculator has been extended to handle particular cases in which a simplification is still possible. The outcome may be:

- a formula in which all uninterpreted functions have been eliminated. This is the equivalent of an exact FADA.
- a formula in which the uninterpreted functions are used to describe a fuzzy relation. This is the counterpart of our use of parameters of the maximum.
- In some cases, the structure of the formula to be simplified is such that it cannot be handled by the Omega calculator. The offending term is replaced by a special marker, *unknown*. This case does not seem to have a counterpart in FADA.

Comparison of Pugh and Wonnacott technique with our own is difficult, because it depends on detailed knowledge of the inner behavior of the Omega calculator. Some observations on example `E2` may be of interest here. In Pugh and Wonnacott's terms, there is a (memory based) flow dependence relation between Statements $\mathtt{S}_1$ and `T` which is described by:

$$\{[x] \rightarrow [] | \, 1 \leq x \leq n, p(x)\},$$

where p is an uninterpreted boolean function which represents the outcome of the test. To obtain the value-based dependence, one has to add the condition that no write to `s` intervenes between $\langle \mathtt{S}_1, x \rangle$ and $\langle \mathtt{R}, [] \rangle$. The part of this condition relating to $\langle \mathtt{S}_1, x' \rangle$ is:

$$\neg \exists x' s.t. (1 \leq x' \leq n, x < x', p(x')).$$

None of the constraints in the above formula is strong enough to fix the value of x'. Hence, the application of a function to a quantified variable cannot be avoided, and this is not handled by the Omega simplifier ([20], section 8.4.1).

There are probably cases in which Pugh and Wonnacott's method may give more precise results than FADA. This is especially true since Wonnacott ([20] Section 8.3.1) uses semantic knowledge to improve the selection of uninterpreted functions. This is an example of the mixed approach, in which an attempt is made to use all available information, whether syntactical or semantical, to improve the dependence calculation. This is clearly the road toward a better understanding of dynamic control programs.

From the results of ADA or FADA, one may deduce many useful abstractions, like reaching definitions, upward and downward exposed regions, and so on. In the case of scalars, this information can be obtained more directly by iterative dataflow analysis. These methods can be extended to arrays: an example is the work of Peng Tu [19, 18]. Regions are approximated by coarser objects than polyhedra: for instance, regular sections [3]. When solving dataflow equations, one has to compute unions and complements of regular sections, which are not regular sections in general. Hence, one introduces approximate operations. The information obtained in this way is less precise than the one given by ADA or FADA, but the analysis is faster and is precise enough for solving some problems like array privatization. Another case in point is the work of Duesterwald et al.[7]. In our minds, the main interest of FADA is that it gives an exhaustive analysis of the source program, and hence is more versatile than other, less precise techniques.

9 Conclusions

This paper gives a method to build a conservative approximation of the flow of values in programs whose control flow and array accesses cannot be known at compile-time. Such programs include control-flow constructs such as `while`s and `if..then..else` constructs, making both control and data flow unpredictable at compile-time. In this paper, we have shown that we can extend the notion of a unique source to that of a source *set*, and have designed a set of algorithms which give, in many cases, surprisingly precise results. A fuzzy array dataflow analyzer is being implemented in Lisp within the PAF project at PRiSM Laboratory.

Our method is generic in so far as it gives a framework for fuzzy analysis that may be adapted to most exact analysis algorithms. More importantly, the net effect of our handling of `while` loops and tests is to add *equations* to the definition of the candidate set, thus improving the probability of success of fast analysis schemes like [14, 11].

Applications of FADA to automatic parallelization include static scheduling, array privatization and register allocation [7]. As a concluding remark, note that a $-$ in a source set points to a possible programming error. Beyond automatic parallelization, a fuzzy array dataflow analysis may therefore be a general tool for translators, compilers and program checkers, as array dataflow analysis was.

Acknowledgments We would like to thank Bill Pugh, Dave Wonnacott and the anonymous referees for helping us improve the presentation of this paper.

References

[1] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Boston / Dordrecht / London, 1988.

[2] T. Brandes. The importance of direct dependences for automatic parallelization. In *ACM Int. Conf. on Supercomputing*, St Malo, France, July 1988.

[3] D. Callahan and K. Kennedy. Compiling programs for distributed memory multiprocessors. *The Journal of Supercomputing*, 2:151–169, 1988.

[4] J.-F. Collard. Space-time transformation of while-loops using speculative execution. In *Proc. of the 1994 Scalable High Performance Computing Conf.*, pages 429–436, Knoxville, TN, May 1994. IEEE.

[5] J.-F. Collard. Automatic parallelization of while-loops using speculative execution. *Int. J. of Parallel Programming*, 23(2):191–219, April 1995.

[6] J.-F. Collard, D. Barthou, and P. Feautrier. Fuzzy array dataflow analysis. In *Proc. of 5th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 92–101, Santa Barbara, CA, July 1995.

[7] E. Duesterwald, R. Gupta, and M.-L. Soffa. A practical data flow framework for array reference analysis and its use in optimization. In *ACM SIGPLAN'93 Conf. on Prog. Lang. Design and Implementation*, pages 68–77, June 1993.

[8] A. Dumay. *Traitement des Indexations non linéaires en parallélisation automatique : une méthode de linéarisation contextuelle.* PhD thesis, Université P. et M. Curie, December 1992.

[9] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, September 1988.

[10] P. Feautrier. Dataflow analysis of scalar and array references. *Int. J. of Parallel Programming*, 20(1):23–53, February 1991.

[11] C. Heckler and L. Thiele. Computing linear data dependencies in nested loop programs. *Parallel Processing Letters*, 4(3):193–204, 1994.

[12] F. Masdupuy. Semantic analysis of interval congruences. In D. Borner, M. Broy, and I.V. Pottosin, editors, *Int. Conf. on Formal Methods in Programming and their Applications*, volume 735 of *LNCS*, pages 142–155, Academgorodok, Novosibirsk, Russia, June 1993. Springer Verlag.

[13] V. Maslov and W. Pugh. Simplifying polynomial constraints over integers to make dependence analysis more precise. Technical Report CS-TR-3109.1, University of Maryland, February 1994.

[14] D. E. Maydan, S. P. Amarasinghe, and M. S. Lam. Array dataflow analysis and its use in array privatization. In *Proc. of ACM Conf. on Principles of Programming Languages*, pages 2–15, January 1993.

[15] W. Pugh and D. Wonnacott. An exact method for analysis of value-based array data dependences. In *Lecture Notes in Computer Science 768: Sixth Annual Workshop on Programming Languages and Compilers*, Portland, OR, August 1993. Springer-Verlag.

[16] W. Pugh and D. Wonnacott. Nonlinear array dependence analysis. In *Third Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, Troy, New York, May 1995.

[17] X. Redon and P. Feautrier. Detection of reductions in sequentials programs with loops. In Arndt Bode, Mike Reeve, and Gottfried Wolf, editors, *Procs of the 5th International Parallel Architectures and Languages Europe, LNCS 694*, pages 132–145, June 1993.

[18] P. Tu. *Array Privatization and Demand Driven Symbolic Analysis.* PhD thesis, University of Illinois at Urbana-Champlain, 1995.

[19] P. Tu and D. Padua. Array privatization for shared and distributed memory machines. September 1992.

[20] D. G. Wonnacott. *Constraint-Based Array Dependence Analysis.* PhD thesis, University of Maryland, 1995.