# Static Complexity Analysis of Programs

Neil D. Jones* Lars Kristiansen†‡

In this paper we explicate the proof calculus introduced in Kristiansen & Jones [10]. We consider an imperative programming language that is *natural*, in the sense that it is an essential fragment of many popular real-life programming languages, and *powerful*, in the sense that it yields full Turing computability. By doing derivations in our calculus, we are able to establish useful information about the computational complexity of programs.

**Related Work:** Our proof calculus is based on an analysis of the relationship between the resource requirements of a computation and the way data might flow during the computation. This analysis extends and refines the insights acquired by research as done by Bellantoni & Cook ([2] normal and safe variables), Leivant ([16] ramification), and in particular, Kristiansen & Niggl ([11, 12] measures). The insight that there is a relationship between the absence and presence of successor-like functions and the computational complexity of a program is a part of the foundation of our calculus, see e.g., Jones [6, 7], Kristiansen & Voda [13, 14], and Kristiansen [9].

Even if our research builds on, and is comparable to, the research discussed above, it has a different emphasis, e.g., we are not aiming at implicit characterisations of complexity classes (even if such characterisations will be easy corollaries of our results). The overall goal of our research is to achieve a better understanding of the relationship between syntactical constructions in natural programming languages and the computational resources required to execute the programs.

Some research has been conducted along these lines: a thesis by Caseiro [4]; papers by Lee, Jones & Ben-Amram, and Jones & Bohr [15, 8] which analyse the relationship between program syntax and program termination; and a thesis by Frederiksen [5] that contains syntactical flow analyses sufficient to recognise that a functional program to runs in polynomial time. In a recent paper Niggl and Wunderlich [17] employ matrices reminiscent of the ones appearing in our calculus, but their method shows far fewer programs to be polynomially bounded than the method of the current paper. Finally, the recent research of Marion, and others, on resource control and quasi-interpretations seems related to the research presented in this paper, see Bonafante, Marion & Moyen [3].

## Programs, Commands and Expressions

We consider nondeterministic imperative programs that manipulate natural numbers (non-negative integers) held in a fixed number of program variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$.

*Department of Computer Science, University of Copenhagen
†Faculty of Engineering, Oslo University College
‡Department of Mathematics, University of Oslo

*This paper was presented at the NIK-2006 conference. For more information, see //www.nik.no/.*

**Syntax:** *Expressions* and *Commands* have forms given by the grammar

$$\begin{array}{rcl}
\mathtt{X} \in \text{Variable} & ::= & \mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots \\
\mathtt{k} \in \text{Constant} & ::= & \text{decimal number} \\
\mathtt{b} \in \text{Boolean exp} & ::= & \mathtt{e} = \mathtt{e}\ ,\ \mathtt{e} < \mathtt{e}\ ,\ \text{etc.} \\
\mathtt{e} \in \text{Expression} & ::= & \mathtt{X} \mid \mathtt{e}\ \mathtt{+}\ \mathtt{e} \mid \mathtt{e}\ \mathtt{*}\ \mathtt{e} \mid \mathtt{e}\ \mathtt{-}\ \mathtt{e} \mid \mathtt{k} \\
\mathtt{C} \in \text{Command} & ::= & \mathtt{skip} \mid \mathtt{X := e} \mid \mathtt{C;C} \mid \mathtt{loop\ X\ \{C\}} \\
 & \mid & \mathtt{if\ b\ then\ C\ else\ C} \mid \mathtt{while\ b\ do\ \{C\}}
\end{array}$$

The variable $\mathtt{X}$ is not allowed to occur in the body $\mathtt{C}$ of the iteration $\mathtt{loop\ X\ \{C\}}$.

**Semantics:** A command is executed as expected from its syntax, so we omit a detailed formalisation. At any point in time each variable $\mathtt{X}_i$ holds a natural number $x_i$ (possibly 0), and the expressions are evaluated in a standard way without any side effects; the operators $\mathtt{*},\mathtt{+},\mathtt{-}$ are respectively multiplication, addition and subtraction. The loop command $\mathtt{loop\,X\,\{C\}}$ executes the command $\mathtt{C}$ in its body $m$ times in a row, where $m$ is the value stored in $\mathtt{X}$ when the loop starts. The command $\mathtt{if\ b\ then\ C_1\ else\ C_2}$ executes the command $\mathtt{C}_1$ (respectively $\mathtt{C}_2$) if $\mathtt{b}$ evaluates to true (respectively false). The command $\mathtt{C}_1\mathtt{;C}_2$ executes first the command $\mathtt{C}_1$ and then the command $\mathtt{C}_2$. Commands of the form $\mathtt{X := e}$ are assignment statements, and the command $\mathtt{skip}$ does nothing. Finally, the command $\mathtt{while\ b\,do\,\{C\}}$ is an ordinary while-loop. Hence, the semantics of our language is very standard.

Let $\mathtt{C}$ be a command whose variables are a subset of $\{\mathtt{X}_1,\ldots,\mathtt{X}_n\}$. The command execution relation $[\![\mathtt{C}]\!](x_1,\ldots,x_n \rightsquigarrow x'_1,\ldots,x'_n)$ holds iff the variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$ respectively hold the numbers $x_1,\ldots,x_n$ when the execution of $\mathtt{C}$ starts, then the variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$ respectively hold the numbers $x'_1,\ldots,x'_n$ when the execution terminates. Similarly, the evaluation expression relation $[\![\mathtt{e}]\!](x_1,\ldots,x_n \rightsquigarrow v)$ holds iff the evaluation of $\mathtt{e}$, when variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$ are as above, yields numerical value $v$.

## Statements and Truth

Given command $\mathtt{C}$, our goal is to discover polynomially bounded data-flow relations between the *initial values* $x_1,\ldots,x_n$ of respectively $\mathtt{X}_1,\ldots,\mathtt{X}_n$ and the *final value* $x'_i$ of $\mathtt{X}_i$ (for $i=1,\ldots,n$) that hold whenever $[\![\mathtt{C}]\!](x_1,\ldots,x_n \rightsquigarrow x'_1,\ldots,x'_n)$. E.g., consider simple commands $\mathtt{C} \equiv \mathtt{Y := U+V}$ and $\mathtt{C}' \equiv \mathtt{Y := Y+Y}$. The sequential compositions $\mathtt{C;C}$, and $\mathtt{C;C}'$, and the iteration $\mathtt{loop\,X\,\{C\}}$ all have polynomially bounded data-flow relations, e.g., $[\![\mathtt{C;C}']\!](y,u,v \rightsquigarrow y',u',v')$ implies $y' \leq 2u+2v$ and $u' \leq u$ and $v' \leq v$. On the other hand, the data flow of the command $\mathtt{C}'' \equiv \mathtt{Y := 1\,;\,loop\,X\,\{C'\}}$ is not polynomially bounded, since $[\![\mathtt{C}'']\!](x,y \rightsquigarrow x',y')$ implies $y' = 2^x$.

***mwp*-Bounds on Value Growth:** An *mwp*-bound (denoted $W,V,U,\ldots$) is a number-theoretic expression of form $\max(\vec{x},q(\vec{y}))+p(\vec{z})$ where $\vec{x}$, $\vec{y}$, and $\vec{z}$ are disjoint lists of variables, and $q$ and $p$ are honest polynomials[1]. The *m-variables* of an *mwp*-bound $W \equiv \max(\vec{x},q(\vec{y}))+p(\vec{z})$ are those in $\vec{x}$; the *w-variables* of $W$ are those in $\vec{y}$; and the *p-variables* of $W$ are those in $\vec{z}$. The notation $W(\vec{x};\vec{y};\vec{z})$ displays the variables in an *mwp*-bound $W$ in order, where $\vec{x}$, $\vec{y}$ and $\vec{z}$ are respectively the *m*-, *w*- and *p*-variables of $W$.

[1] A polynomial $p$ is *honest* if it is monotone in all its variables, e.g., if $y$ in $p$ implies $p(y,\vec{x}) \leq p(y+1,\vec{x})$.

(In *mwp*, *m* stands for "maximum", *p* stands for "polynomial", and *w* stands for "weak polynomial".) We will also use a more abstract notation $\mathbf{val}_W(x)$ to relate an *mwp*-bound $W$ and a variable $x$:

$$\mathbf{val}_W(x) \;=\; \begin{cases} m & \text{if } x \text{ is an } m\text{-variable of } W \\ w & \text{if } x \text{ is an } w\text{-variable of } W \\ p & \text{if } x \text{ is an } p\text{-variable of } W \\ 0 & \text{otherwise, i.e. if } x \text{ does not occur in } W \end{cases}$$

We will use *mwp*-bounds to describe bounds on the value growth of variables, e.g., if $[\![\mathtt{X_1 := X_1 + X_2}]\!](x_1, x_2 \rightsquigarrow x_1', x_2')$, then we have $x_1' \leq W(x_1;;x_2)$ where $W \equiv \max(x_1, 0) + x_2$. A slightly more sophisticated example is given in Example 1

**Example 1.** We have

$$[\![\mathtt{loop\ X_3\ \{X_1 := X_1 + X_2\}}]\!](x_1, x_2, x_3 \rightsquigarrow x_1', x_2', x_3') \Rightarrow W_1 \geq x_1' \wedge W_2 \geq x_2' \wedge W_3 \geq x_3'$$

where $W_1(x_1;;x_2,x_3) \equiv x_1 + x_2 \cdot x_3$ and $W_2(x_2;;) \equiv x_2$ and $W_3(x_3;;) \equiv x_3$. □

It is important to note that *mwp*-bounds are not unique in the sense that a number-theoretic expression might be numerically equal to several different *mwp*-bounds.

**Example 2.** The expression $x_1 + x_2$ has several different valid *mwp*-bound descriptions, e.g. $U, V, W$ where $U(;x_1,x_2;) \equiv \max(0, x_1 + x_2) + 0$, and $V(x_1;;x_2) \equiv \max(x_1, 0) + x_2$ and $W(x_2;;x_1) \equiv \max(x_2, 0) + x_1$ □

*mwp*-**Bounds Represented by Vectors and Matrices:** We will represent an *mwp*-bound $W$ over the variables $x_1, \ldots, x_n$ by a column vector

$$V = \begin{pmatrix} \mathbf{val}_W(x_1) \\ \mathbf{val}_W(x_n) \\ \vdots \\ \mathbf{val}_W(x_n) \end{pmatrix}$$

in a matrix, e.g., if $n = 5$, an *mwp*-bound of the form $\max(x_5, q(x_2, x_4)) + p(x_1)$ is represented by the vector $\begin{pmatrix} p \\ w \\ 0 \\ w \\ m \end{pmatrix}$. Such a vector representation abstracts away the exact polynomials, but preserve the form of the *mwp*-bounds. An $n \times n$ matrix consists of $n$ column vectors $(V_1, \ldots, V_n)$, and thus an $n \times n$ matrix over $\{0, m, w, p\}$ will represent $n$ *mwp*-bounds.

**Example 3.** Example 1 can now be expressed much more concisely in matrix form, but the exact polynomials involved are lost:

$$\mathtt{loop\ X_3\ \{X_1 := X_1 + X_2\}} : \begin{pmatrix} m & 0 & 0 \\ p & m & 0 \\ p & 0 & m \end{pmatrix}$$

□

**The Meaning of a Statement:** *Statements* are of the form $\mathtt{C}:M$ where $\mathtt{C}$ is a command over the program variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$ and $M$ is an $n\times n$ matrix. Example 3 indicates how a statement can be read as an assertion of the existence of certain *mwp*-bounds: For $i=1,\ldots,n$ there exists an *mwp*-bound $W_i$ such that, whenever $[\![\mathtt{C}]\!](\vec{x}\rightsquigarrow\vec{x}')$, we have $x_i'\leq W_i$ where $W_i$ has the form given by the $i$'th column vector of $M$. We will now give the formal definition of *truth*.

**Definition 1.** Let $\mathtt{e}$ be an expression over the variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$, and let $V=(\alpha_1,\alpha_2,\cdots,\alpha_n)$ be an $n\times 1$ column vector. A *statement* $\mathtt{e}:V$ *is true*, written $\models\mathtt{e}:V$, iff there exists an *mwp*-bound $W$ such that

1. $[\![\mathtt{e}]\!](x_1,\ldots,x_n\rightsquigarrow v)$ implies $W\geq v$
2. $\mathbf{val}_W(x_i)=\alpha_i$, for $i\in\{1,\ldots,n\}$.

Let $\mathtt{C}$ be a command over the variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$, and let $M$ be an $n\times n$ matrix. A *statement* $\mathtt{C}:M$ *is true*, written $\models\mathtt{C}:M$, iff there exist *mwp*-bounds $W_1,\ldots,W_n$ such that

1. $[\![\mathtt{C}]\!](x_1,\ldots,x_n\rightsquigarrow x_1',\ldots,x_n')\ \Rightarrow\ W_1\geq x_1'\wedge\ldots\wedge W_n\geq x_n'$
2. $\mathbf{val}_{W_j}(x_i)=M_{ij}$, for $i,j\in\{1,\ldots,n\}$.

□

The truth of $\mathtt{C}:M$ implies that any values computed by the command $\mathtt{C}$ will be polynomially bounded in the input values, and if a command $\mathtt{C}$ computes a function growing exponentially, the statement $\mathtt{C}:M$ will be false for any matrix $M$. For example, the value computed into $\mathtt{X}_1$ by the command $\mathtt{loop}\,\mathtt{X}_2\,\{\mathtt{X}_1:=\mathtt{X}_1+\mathtt{X}_1\}$ grows exponentially, and thus the command has no *mwp*-bounds, i.e., the statement $\mathtt{loop}\,\mathtt{X}_2\,\{\mathtt{X}_1:=\mathtt{X}_1+\mathtt{X}_1\}:M$ is false for any $M$.

## *mwp*-Algebra and Notation

We now introduce an *mwp*-algebra and some basic vector and matrix operations. This section is quite standard and not peculiar to our problem, except for the concrete choices of the set of scalars, and operations on them. For more on related algebra, see e.g. [1].

**Scalars:** A *scalar* is an element of $\mathcal{V}=\{0,m,w,p\}$. The elements in $\mathcal{V}$ are ordered as follows: $0<m<w<p$. We use Greek letter $\alpha,\beta,\gamma\ldots$ to denote the elements in $\mathcal{V}$. The *least upper bound* of $\alpha,\beta\in\mathcal{V}$ is denoted by $\alpha+\beta$, i.e., $\alpha+\beta=\alpha$ if $\alpha\geq\beta$; otherwise $\alpha+\beta=\beta$. Let $\alpha_1,\ldots,\alpha_n$ be a sequence of values from $\mathcal{V}$, then $\sum_{i=1\ldots n}\alpha_i\stackrel{\text{def}}{=}\alpha_1+\cdots+\alpha_n$. The *product* of $\alpha,\beta\in\mathcal{V}$ is denoted by $\alpha\times\beta$ and defined by $\alpha\times\beta=\alpha+\beta$ if $\alpha,\beta\in\{m,w,p\}$; otherwise $\alpha\times\beta=0$.

**Vectors:** We use $V,U,T\ldots$ to denote vectors over $\mathcal{V}$, and $V_i$ denotes the $i$'th element in the vector $V$. The *least upper bound* $T\oplus U$ of the vectors $T$ and $U$ is defined by $V=T\oplus U$ iff $V_i=T_i+U_i$. A *vector* should be thought of as a column vector in an $n\times n$ matrix, but sometimes, for typographical reasons, we write it horizontally: $V=(\alpha_1,\ldots,\alpha_n)$. The *scalar product* $\alpha V$ is defined by $\alpha(\alpha_1,\ldots,\alpha_n)=(\alpha\times\alpha_1,\ldots,\alpha\times\alpha_n)$.

**Matrices:** We use $M, A, B, C, \ldots$ to denote $(n \times n)$ matrices over $\mathcal{V}$, and $M_{ij}$ denotes the element in the $i$'th row and $j$'th column in the matrix $M$. The *least upper bound* $A \oplus B$ of the matrices $A$ and $B$ is defined component wise. Define the *product* $A \otimes B$ of the matrices $A$ and $B$ by $M = A \otimes B$ iff $M_{ij} = \sum_{k=1,\ldots,n} A_{ik} \times B_{kj}$ (standard matrix multiplication). The *zero matrix* is denoted by $\mathbf{0}$. We define $\mathbf{0}$ by $M = \mathbf{0}$ iff $M_{ij} = 0$ for all indices $i, j$. We have $\mathbf{0} \oplus M = M \oplus \mathbf{0} = M$ for any matrix $M$. The *identity matrix* is denoted by $\mathbf{1}$. We define $\mathbf{1}$ by $M = \mathbf{1}$ iff $M_{ij} = m$ for $i = j$, and $M_{ij} = 0$ for $i \neq j$. We have $\mathbf{1} \otimes M = M \otimes \mathbf{1} = M$ for any matrix $M$.

**The Closure Operator:** A unary operation on matrices, denoted $*$ and called *closure* or *star*, is defined by the infinite sum

$$M^* \;=\; \mathbf{1} \oplus M \oplus (M \otimes M) \oplus (M \otimes M \otimes M) \oplus (M \otimes M \otimes M \otimes M) \oplus \ldots$$

Let $\mathcal{M}$ denote the set of $(n \times n)$ matrices. The algebraic structure $(\mathcal{M}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a finite closed semiring. The closure operator is defined in any closed semiring, and we have $M^* = 1 \oplus (M \otimes M^*)$.

**Vector and Matrix Modification Notation:** Let $M$ be a matrix and let $V$ be a vector. Then $M \overset{k}{\leftarrow} V$ denotes the matrix obtained by replacing the $k$'th column vector in $M$ by the vector $V$, that is, $M' = M \overset{k}{\leftarrow} V$ iff $M'_{ij} = V_i$ if $j = k$, and $M'_{ij} = M_{ij}$ if $j \neq k$. Hence, if $n = 4$ and $V = \begin{pmatrix} m \\ p \\ 0 \\ p \end{pmatrix}$, then

$$\mathbf{1} \overset{2}{\leftarrow} V \;=\; \begin{pmatrix} m & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix} \overset{2}{\leftarrow} \begin{pmatrix} m \\ p \\ 0 \\ p \end{pmatrix} \;=\; \begin{pmatrix} m & m & 0 & 0 \\ 0 & p & 0 & 0 \\ 0 & 0 & m & 0 \\ 0 & p & 0 & m \end{pmatrix}$$

**Compact Vector and Matrix Notations:** Occasionally we will write a non-0 vector entry $V_i = \alpha$ vertically as $\overset{\alpha}{}_i$, and identify the vector $V$ with the set of all of its non-0 entries, e.g., the set $\{\overset{m}{}_1, \overset{m}{}_3, \overset{p}{}_4\}$ is identified with the vector $(m, 0, m, p)$. We also identify matrix $M$ with its set of non-0 entries $M_{ij}$, writing $M \equiv \{\overset{\alpha}{}_i \!\!\rightarrow j \mid M_{ij} = \alpha \neq 0\}$. Following this line, a column vector $\{\overset{\alpha_1}{}_{i_1} \!\!\rightarrow j, \overset{\alpha_2}{}_{i_2} \!\!\rightarrow j, \ldots, \overset{\alpha_k}{}_{i_k} \!\!\rightarrow j\}$ can be abbreviated to $\{\overset{\alpha_1 \alpha_2 \ldots \alpha_k}{}_{i_1\ i_2\ \ldots\ i_k} \!\!\rightarrow j\}$. Hence, when $n = 4$, the matrix

$$\begin{pmatrix} m & 0 & 0 & 0 \\ m & p & p & 0 \\ m & m & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix}$$

can be written $\{\overset{mmm}{}_{1\ 2\ 3} \!\!\rightarrow 1, \overset{pm}{}_{2\ 3} \!\!\rightarrow 2, \overset{pm}{}_{2\ 3} \!\!\rightarrow 3, \overset{m}{}_4 \!\!\rightarrow 4\}$.

# A Calculus for Deriving Statements

In this section we give a proof calculus for deriving true statements. We just give derivation rules for so-called *core* expressions, i.e. the expressions built up from variables by applying the operators + (addition) and $*$ (multiplication). The calculus easily extends to a calculus for the full language, that is, a calculus that also can handled assignments of constants in expressions and the subtraction operator. Due to space considerations we will not discuss the inference rule for the while-loop.

**Assigning Vectors to Expressions:** The *variables of expression* $\mathtt{e}$, written $var(\mathtt{e})$, is a set of natural numbers. Let $i \in var(\mathtt{e})$ iff the variable $\mathtt{X}_i$ occurs in $\mathtt{e}$.

We derive $\vdash \mathtt{e} : V$, for core expression $\mathtt{e}$ and vector $V$, by the rules

$$(E1) \quad \vdash \mathtt{X}_i : \{{}^{m}_{i}\} \qquad\qquad (E2) \quad \vdash \mathtt{e} : \{{}^{w}_{i} \mid i \in var(\mathtt{e})\}$$

$$(E3) \quad \frac{\vdash \mathtt{e}_1 : U \qquad \vdash \mathtt{e}_2 : V}{\vdash \mathtt{e}_1\mathtt{+}\mathtt{e}_2 : pU \oplus V} \qquad (E4) \quad \frac{\vdash \mathtt{e}_1 : U \qquad \vdash \mathtt{e}_2 : V}{\vdash \mathtt{e}_1\mathtt{+}\mathtt{e}_2 : U \oplus pV}$$

When $\vdash \mathtt{e} : V$, we will say that the calculus *assigns* the vector $V$ to the expression $\mathtt{e}$. Further, if $\vdash \mathtt{e} : V$ and $V_i = \alpha \in \{m, w, p\}$, we will say that the variable $\mathtt{X}_i$ in the expression $\mathtt{e}$ is labeled $\alpha$.

The rule (E1) says that if an expression is just a single variable, then this variable can be labeled $m$. The rule (E2) says that we always can label all the variables in an expression by $w$'s. The calculus might assign several different vectors to the same expression, in particular, the rules (E3) and (E4) give several options for how to label the variables in an expression containing the operator $\mathtt{+}$. The various options corresponds to various forms of valid $mwp$-bounds, see Example 2.

**Example 4.** We have $\vdash \mathtt{X}_1\mathtt{+}\mathtt{X}_2 : \{{}^{ww}_{1\,2}\}$ by (E2). Further, we have the derivation

$$\frac{\vdash \mathtt{X}_1 : \{{}^{m}_{1}\} \qquad \vdash \mathtt{X}_2 : \{{}^{m}_{2}\}}{\vdash \mathtt{X}_1\mathtt{+}\mathtt{X}_2 : p\{{}^{m}_{1}\} \oplus \{{}^{m}_{2}\}}$$

by (E1) and (E3), and thus, since $p\{{}^{m}_{1}\} \oplus \{{}^{m}_{2}\} = \{{}^{pm}_{1\,2}\}$, we have $\vdash \mathtt{X}_1\mathtt{+}\mathtt{X}_2 : \{{}^{pm}_{1\,2}\}$. By a symmetric derivation applying (E4), we also have $\vdash \mathtt{X}_1\mathtt{+}\mathtt{X}_2 : \{{}^{mp}_{1\,2}\}$. Thus the calculus assigns (at least) three different vectors to the expression $\mathtt{X}_1\mathtt{+}\mathtt{X}_2$. The reader should compare the tree vectors to the three $mwp$-bounds given in Example 2. □

A net effect of the rules is that at most one variable in an expression can be labeled $m$, and in the case when one variable is labeled $m$, all the other variables have to be labeled $p$. This corresponds to the fact that if $q$ is an honest polynomial depending on $\vec{x},\vec{y},\vec{z}$, and $W(\vec{x};\vec{y};\vec{z})$ is an $mwp$-bound such that $q \leq W(\vec{x};\vec{y};\vec{z})$, then the list of $\vec{x}$ of $m$-variables will contain at most one variable; further, if the list $\vec{x}$ contain one variable, then the list $\vec{y}$ of $w$-variables has to be empty.

**Assigning Matrices to Commands:** We derive $\vdash \mathtt{C} : M$, for command $\mathtt{C}$ and matrix $M$, by the rules

$$(S) \quad \vdash \mathtt{skip} : \mathbf{1} \qquad\qquad (A) \quad \frac{\vdash \mathtt{e} : V}{\vdash \mathtt{X}_j \mathtt{:=} \mathtt{e} : \mathbf{1} \overset{j}{\leftarrow} V}$$

$$(C) \quad \frac{\vdash \mathtt{C}_1 : A \qquad \vdash \mathtt{C}_2 : B}{\vdash \mathtt{C}_1\,\mathtt{;}\,\mathtt{C}_2 : A \otimes B} \qquad (I) \quad \frac{\vdash \mathtt{C}_1 : A \qquad \vdash \mathtt{C}_2 : B}{\vdash \mathtt{if\ b\ then\ C}_1\ \mathtt{else\ C}_2 : A \oplus B}$$

In contrast to the rules above, the rules for loop statement have side a condition

$$(L) \quad \frac{\vdash \mathtt{C} : M}{\vdash \mathtt{loop\ X}_\ell\ \mathtt{\{C\}} : M^* \oplus \{\, {}^{p}_{\ell} \!\!\rightarrow j \mid \exists i\,[M^*_{ij} = p]\,\}} \quad (\text{if } \forall i[M^*_{ii} = m])$$

The side condition says that the closure $M^*$ should have nothing but $m$'s on its diagonal. The rule (L) is not applicable if this condition is not fulfilled.

When $\vdash \mathtt{C} : M$, we will say that the calculus *assigns* the matrix $M$ to the command $\mathtt{C}$. Further, we will say that a command is *derivable* if the calculus assign at least one matrix to the command.

**Theorem 1 (Soundness).** $\vdash \mathtt{C} : M$ implies $\models \mathtt{C} : M$.

The proof of Theorem 1 is not yet published, but we have carried out all the details of the extensive and highly non-trivial proof.

## Discussion and Explanation of the Calculus

We arrived at our calculus by thoroughly analysing how data might flow in computations. The observation that certain pattern of flow guaranteed polynomial bounds on the computed values, whereas other patterns did not, eventually led us to the *mwp*-bounds and their particular form. Moreover, we observed that when it comes to establish the *existence* of polynomial bounds, it will be profitable keeping track of certain vital information on the data flow rather than information of more number-theoretic nature, like e.g. coefficients and degrees of polynomials. This led us to a proof calculus where matrices are used in a book-keeping process recording information on data flow.

Our calculus records three different types of flow: $m$-flow, $w$-flow and $p$-flow. An $n \times n$ matrix $M$ over $\{0, m.w, p\}$ represents flow relations over the variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$ in an natural way, that is, $M_{ij} = \alpha \neq 0$ iff there is an $\alpha$-flow from the source variable $\mathtt{X}_i$ to the target variable $\mathtt{X}_j$; and $M_{ij} = 0$ iff the source variable $\mathtt{X}_i$'s value is not used to compute the target variable $\mathtt{X}_j$'s new value, that is, "no flow" from $\mathtt{X}_i$ to $\mathtt{X}_j$. If the statement $\mathtt{C} : M$ is derivable in the calculus, the matrix $M$ gives sound data-flow relations for the command $\mathtt{C}$.

**m-Flow:** $m$-Flow is harmless in the sense that data flowing in a program where $m$-flow only occur, will be trivially polynomially bounded. None of the input values will ever be increased, and hence, any value computed by such a program will be bounded by $\max(\vec{x})$ where $\vec{x}$ are the input values. Thus, our calculus does not impose any restrictions on the $m$-flow in the derivable programs, still, the calculus will keep track of the $m$-flow in programs. We will now study some examples illustrating the book-keeping facilities of the calculus. The examples show derivations of commands where data $m$-flows between the variables $\mathtt{X}_1, \mathtt{X}_2, \mathtt{X}_3, \mathtt{X}_4$.

**Primitive Commands:** By the axiom (S) we have $\vdash \mathtt{skip} : \mathbf{1}$ where $\mathbf{1}$ is the unity matrix, i.e. $\mathbf{1} = \{ {}^{m}_{1}\!\!\rightarrow 1, {}^{m}_{2}\!\!\rightarrow 2, {}^{m}_{3}\!\!\rightarrow 3, {}^{m}_{4}\!\!\rightarrow 4 \}$. The derivation reflects the fact that in a command doing nothing, there is an $m$-flow from each variable to itself.

The derivation

$$\frac{\vdash \mathtt{X}_2 : \begin{pmatrix} 0 \\ m \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_1 := \mathtt{X}_2 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ m & m & 0 & 0 \\ 0 & 0 & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix}}$$

starts by an application of the axiom (E1) and proceeds an application of the assignment rule (A). The derivation registers the $m$-flow from $\mathtt{X}_2$ to $\mathtt{X}_1$, and the $m$-flow from $\mathtt{X}_i$ to $\mathtt{X}_i$ for $i = 2, 3, 4$.

**Composition and Matrix Multiplication:** In the command $\mathtt{X}_1 := \mathtt{X}_2\,;\ \mathtt{X}_2 := \mathtt{X}_3\,;\ \mathtt{X}_3 := \mathtt{X}_1$ there is obviously $m$-flow from $\mathtt{X}_2$ to $\mathtt{X}_1$, and $m$-flow from $\mathtt{X}_3$ to $\mathtt{X}_2$. There is also $m$-flow from $\mathtt{X}_2$ to $\mathtt{X}_3$ since when the assignment $\mathtt{X}_3 := \mathtt{X}_1$ takes places, the initial content of $\mathtt{X}_1$ will be replaced by the initial content of $\mathtt{X}_2$. The next derivation shows how our calculus keeps track of the data flow in the program by matrix multiplication.

$$\dfrac{\dfrac{\vdash \mathtt{X}_2 : \begin{pmatrix} 0 \\ m \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_1 := \mathtt{X}_2 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ m & m & 0 & 0 \\ 0 & 0 & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix}} \quad \dfrac{\vdash \mathtt{X}_3 : \begin{pmatrix} 0 \\ 0 \\ m \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_2 := \mathtt{X}_3 : \begin{pmatrix} m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & m & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix}}}{\vdash \mathtt{X}_1 := \mathtt{X}_2\,;\ \mathtt{X}_2 := \mathtt{X}_3 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ m & 0 & 0 & 0 \\ 0 & m & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix}} \quad \dfrac{\vdash \mathtt{X}_1 : \begin{pmatrix} m \\ 0 \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_3 := \mathtt{X}_1 : \begin{pmatrix} m & 0 & m & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m \end{pmatrix}}$$

$$\vdash \mathtt{X}_1 := \mathtt{X}_2\,;\ \mathtt{X}_2 := \mathtt{X}_3\,;\ \mathtt{X}_3 := \mathtt{X}_1 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ m & 0 & m & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & 0 & m \end{pmatrix}$$

The bottom line of the derivation is written

$$\vdash \mathtt{X}_1 := \mathtt{X}_2\,;\ \mathtt{X}_2 := \mathtt{X}_3\,;\ \mathtt{X}_3 := \mathtt{X}_1 : \{ {}^{m}_{2}\!\!\rightarrow 1, {}^{m}_{3}\!\!\rightarrow 2, {}^{m}_{2}\!\!\rightarrow 3, {}^{m}_{4}\!\!\rightarrow 4 \}$$

in compact notation. Notice that it is easy to read off the data flow from the compact notation as ${}^{\alpha}_{i}\!\!\rightarrow j$ signifies $\alpha$-flow from $\mathtt{X}_i$ to $\mathtt{X}_j$.

**Loops and the Closure Operator:** Let $\mathtt{C}^0 \equiv \mathtt{skip}$ and $\mathtt{C}^{n+1} \equiv \mathtt{C}\,;\,\mathtt{C}^n$. Further, let $M^0 = \mathbf{1}$ and $M^{n+1} = M \otimes M^n$. In general, the calculus keeps track of the flow in the command $\mathtt{C}_1\,;\,\mathtt{C}_2$ by multiplying the matrix assigned to $\mathtt{C}_1$ by the matrix assigned to $\mathtt{C}_2$. Hence, if the matrix $M$ records the $m$-flow in the command $\mathtt{C}$, then the matrix $\mathbf{1}$ keeps track of the $m$-flow in $\mathtt{C}^0$; the matrix $M^1$ keeps track of the $m$-flow in $\mathtt{C}^1$; the matrix $M^2$ keeps track of the $m$-flow in $\mathtt{C}^2$; and so on. The closure $M^*$ where

$$M^* \;=\; \mathbf{1} \oplus M \oplus (M \otimes M) \oplus (M \otimes M \otimes M) \oplus (M \otimes M \otimes M \otimes M) \oplus \ldots$$

will keep track of the $m$-flow in the commands $\mathtt{loop}\,\mathtt{X}\,\{\mathtt{C}\}$ and $\mathtt{while\,b\,do}\,\{\mathtt{C}\}$.

Let us return to our example. We have assigned the matrix

$$A = \{ {}^{m}_{2}\!\!\rightarrow 1, {}^{m}_{3}\!\!\rightarrow 2, {}^{m}_{2}\!\!\rightarrow 3, {}^{m}_{4}\!\!\rightarrow 4 \}$$

to the command $\mathtt{C} \equiv \mathtt{X}_1 := \mathtt{X}_2\,;\ \mathtt{X}_2 := \mathtt{X}_3\,;\ \mathtt{X}_3 := \mathtt{X}_1$, and now we want to assign a matrix to the command $\mathtt{loop}\ \mathtt{X}_4\ \{\mathtt{C}\}$ by applying the inference rule

$$(L) \quad \dfrac{\vdash \mathtt{C} : M}{\vdash \mathtt{loop}\ \mathtt{X}_\ell\ \{\mathtt{C}\} : M^* \oplus \{ {}^{p}_{\ell}\!\!\rightarrow j \mid \exists i\,[M^*_{ij} = p] \}} \quad (\text{if } \forall i [M^*_{ii} = m])$$

We have $A^* \;=\; \{ {}^{m}_{2}\!\!\rightarrow 1, {}^{m}_{3}\!\!\rightarrow 2, {}^{m}_{2}\!\!\rightarrow 3, {}^{m}_{4}\!\!\rightarrow 4 \}^* = \{ {}^{mmm}_{1\,2\,3}\!\!\rightarrow 1, {}^{mm}_{2\,3}\!\!\rightarrow 2, {}^{mm}_{2\,3}\!\!\rightarrow 3, {}^{m}_{4}\!\!\rightarrow 4 \}$. There are only $m$'s on the diagonal of $A^*$, and thus the side condition of the inference rule is satisfied. Further, we have $A^* \oplus \{ {}^{p}_{\ell}\!\!\rightarrow j \mid \exists i\,[A^*_{ij} = p] \} \;=\; A^* \oplus \mathbf{0} \;=\; A^*$ and thus

$$\dfrac{\vdash \mathtt{C} : A}{\vdash \mathtt{loop}\ \mathtt{X}_4\ \{\mathtt{C}\} : A^*}$$

is a valid instantiation of inference rule (L).

**Conditionals and Matrix Addition:** The inference rule for the if-then-else construction works rather straightforwardly, and no examples should be required to convince the reader that if the matrix $A$ keeps tracks of the $m$-flow in the command $\mathtt{C}_1$, and the matrix $B$ keeps track of the $m$-flow in the command $\mathtt{C}_2$, then the matrix $A \oplus B$ will keep track of (gives an upper bound for) the $m$-flow in the command $\mathtt{if\ b\ then\ C}_1\ \mathtt{else\ C}_2$.

***w*-Flow and *p*-Flow:** The technical machinery for tracing $w$-flow and $p$-flow is of course an extension of the machinery tracing the $m$-flow. Unfortunately, we do not have any room for explicating the details, but the reader should note that the ordering of the scalars, i.e. $0 < m < w < p$, plays a significant role. However, in contrast to $m$-flow, $w$-flow and $p$-flow might be harmful in the sense that certain patterns of such flow might not be polynomially bounded. Thus, our calculus has to impose restrictions on the $w$- and $p$-flow of the derivable commands, and it turns out that it is sufficient to preclude *reflexive* $w$- and $p$-flow inside loops. In contrast to irreflexive $p$-flow, irreflexive $w$-flow is *iteration-independent*, and by taking advantage of this nuance between $p$-flow and $w$-flow, we achieve a more complete calculus, that is, a calculus where the inference rules are not weaker than necessary. In the following we will study some examples and elaborate on $w$-flow, $p$-flow, (ir)reflexivity and iteration-(in)dependency.

**Reflexivity:** We will say that the $w$-flow ($p$-flow) in a command is reflexive when there is a $w$-flow ($p$-flow) from a variable to itself. When there is $w$-flow ($p$-flow) from a source variable $\mathtt{X}_j$ to a target variable $\mathtt{X}_i$, the data flowing might be increased, e.g. doubled, and hence, if data $w$-flows ($p$-flows) from $\mathtt{X}_i$ to $\mathtt{X}_i$ in a command $\mathtt{C}$, the content of $\mathtt{X}_i$ might be doubled by executing $\mathtt{C}$. Such a doubling entails that the data flow in the program $\mathtt{loop\ X}_\ell\ \mathtt{\{C\}}$ not will be be polynomially bounded.

The side condition of the rule (L) prevents derivations of commands where reflexive $w$- and $p$-flow takes place inside loops. To apply the rule, it is required that the closure of matrix in the premise has nothing but $m$'s on the diagonal. If there are only $m$'s on the diagonal of the closure, there cannot be any $w$'s or $p$'s there, and this signifies that there will be no reflexive $w$-flow or $p$-flow when the command in the premise is executed inside a loop.

There is harmful reflexive flow in the command $\mathtt{X}_1 := \mathtt{X}_1 + \mathtt{X}_1$ as data flowing from $\mathtt{X}_1$ to $\mathtt{X}_1$ might be increased during the flow. (If $\mathtt{X}_1$ holds any number different from 0, the data *will* be increased.) Let us study what happens when we search for a derivation of the command $\mathtt{loop\,X}_2\,\mathtt{\{X}_1 := \mathtt{X}_1 + \mathtt{X}_1\mathtt{\}}$. The calculus assigns two different vectors to the expression $\mathtt{X}_1 + \mathtt{X}_1$. We have $\vdash \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} w \\ 0 \end{smallmatrix}\right)$ by (E2), and we have

$$\frac{\vdash \mathtt{X}_1 : \left(\begin{smallmatrix} m \\ 0 \end{smallmatrix}\right) \qquad \vdash \mathtt{X}_1 : \left(\begin{smallmatrix} m \\ 0 \end{smallmatrix}\right)}{\vdash \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} p \\ 0 \end{smallmatrix}\right) \oplus_p \left(\begin{smallmatrix} m \\ 0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} p \\ 0 \end{smallmatrix}\right)}$$

by (E1) and (E4). These are the only vectors the calculus assigns to the expression. (Though, the assignment $\vdash \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} p \\ 0 \end{smallmatrix}\right)$ has several derivations.) Let us search for a derivation from both assignments. We proceed by applying the assignment rule (A), and then, we try apply the loop rule (L).

$$\frac{\dfrac{\vdash \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} w \\ 0 \end{smallmatrix}\right)}{\vdash \mathtt{X}_1 := \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} w & 0 \\ 0 & m \end{smallmatrix}\right)}}{\vdash \mathtt{loop\,X}_2\,\mathtt{\{X}_1 := \mathtt{X}_1 + \mathtt{X}_1\mathtt{\}} : ?} \qquad \frac{\dfrac{\vdash \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} p \\ 0 \end{smallmatrix}\right)}{\vdash \mathtt{X}_1 := \mathtt{X}_1 + \mathtt{X}_1 : \left(\begin{smallmatrix} p & 0 \\ 0 & m \end{smallmatrix}\right)}}{\vdash \mathtt{loop\,X}_2\,\mathtt{\{X}_1 := \mathtt{X}_1 + \mathtt{X}_1\mathtt{\}} : ?}$$

In both cases we find that the side condition $\forall i[M^*_{ii} = m]$ for applying the rule (L) is violated, and we conclude that the command is not derivable.

**Iteration-Independence:** We will explicate the difference between *w*-flow and *p*-flow by studying some simple examples. In the two commands $\mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1$ and $\mathtt{X}_2 := \mathtt{X}_2 + \mathtt{X}_1$ data flows from $\mathtt{X}_1$ to $\mathtt{X}_2$, and in either case the content of $\mathtt{X}_2$ might be increased.

In the first command data *w*-flows as the content of $\mathtt{X}_1$ *will not be* added to the old content of $\mathtt{X}_2$, whereas in the second command data *p*-flows as the data in $\mathtt{X}_1$ *will be* added to the old content of $\mathtt{X}_2$. What happens when we execute each of the commands $k$ times in a row? When we study the commands

$$\underbrace{\mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1 \,;\, \ldots \,;\, \mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1}_{k} \quad \text{and} \quad \underbrace{\mathtt{X}_2 := \mathtt{X}_2 + \mathtt{X}_1 \,;\, \ldots \,;\, \mathtt{X}_2 := \mathtt{X}_2 + \mathtt{X}_1}_{k}$$

it is easy to see that if $k > 0$, the value flowing into $\mathtt{X}_2$ *does not* depend on $k$ in the first case whereas it *does* in the second. The difference does matter when we the commands are executed inside loops.

In the command $\mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2 := \mathtt{X}_2 + \mathtt{X}_1\}$, a bound on the value flowing into $\mathtt{X}_2$ depends on the iteration count, and thus, data will *p*-flow from the iteration variable $\mathtt{X}_3$ to $\mathtt{X}_2$. In the command $\mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1\}$, a bound on the value flowing into $\mathtt{X}_2$ does not depend on the iteration count, and there will be no flow at all from the iteration variable $\mathtt{X}_3$ to $\mathtt{X}_2$. (If $\mathtt{X}_3$ stores 0, the assignment $\mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1$ will not be executed; otherwise it will be executed. Thus, even though the value computed into $\mathtt{X}_2$ does depend on $\mathtt{X}_3$, there *exists a polynomial bound* on the value which does not.)

In general, if data *p*-flows from *some* source variable to a target variable inside a loop, data will will also *p*-flow from the loop's iteration variable into the target variable; if data does not *p*-flow from any variable to a target variable, there will be no flow at all from the loop's iteration variable to the target variable. The rule

$$(L)\ \frac{\vdash \mathtt{C} : M}{\vdash \mathtt{loop}\,\mathtt{X}_\ell\,\{\mathtt{C}\} : M^* \oplus \{\, {}^{p}_{\ell}\!\!\rightarrow j \mid \exists i\,[M^*_{ij} = p]\,\}}\ (\text{if } \forall i[M^*_{ii} = m])$$

records the flow from the loop's iteration variable $\mathtt{X}_\ell$ to the variables in the loop's body by adding the matrix $\{\, {}^{p}_{\ell}\!\!\rightarrow j \mid \exists i\,[M^*_{ij} = p]\,\}$ to the closure $M^*$.

We will now give two derivations of $\mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1\}$. This is the command where irreflexive *w*-flow, but no *p*-flow, takes place in the loop's body. We have

$$\dfrac{\dfrac{\dfrac{\vdash \mathtt{X}_1 : \begin{pmatrix} m \\ 0 \\ 0 \end{pmatrix} \qquad \vdash \mathtt{X}_1 : \begin{pmatrix} m \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_1 + \mathtt{X}_1 : \begin{pmatrix} p \\ 0 \\ 0 \end{pmatrix}}}{\vdash \mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1 : \begin{pmatrix} m & p & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}}}{\vdash \mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2 := \mathtt{X}_1 + \mathtt{X}_1\} : \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & p & m \end{pmatrix}}$$

by the inference rules (E1), (E4), (A) and (L). The application of the loop rule is correct since

$$\begin{pmatrix} m & p & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}^* \oplus \{\, {}^{p}_{\ell}\!\!\rightarrow j \mid \exists i\,[\begin{pmatrix} m & p & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}^*_{ij} = p]\,\} = \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & p & 0 \end{pmatrix} = \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & p & m \end{pmatrix}.$$

We also have

$$\dfrac{\dfrac{\vdash \mathtt{X}_1+\mathtt{X}_1 : \begin{pmatrix} w \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_2:=\mathtt{X}_1+\mathtt{X}_1 : \begin{pmatrix} m & w & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}}}{\vdash \mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2:=\mathtt{X}_1+\mathtt{X}_1\} : \begin{pmatrix} m & w & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}}$$

by (E2), (A) and (L). Now, the application of the loop rule is correct since

$$\begin{pmatrix} m & w & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}^* \oplus \{ \overset{p}{\ell} \!\!\rightarrow j \mid \exists i [ \begin{pmatrix} m & w & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}^*_{ij} = p ] \} = \begin{pmatrix} m & w & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} m & w & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}.$$

Now, assume

$$[\![\mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2:=\mathtt{X}_1+\mathtt{X}_1\}]\!](x_1,x_2,x_3 \rightsquigarrow x_1',x_2',x_3') \,.$$

By The Soundness Theorem the two derivations yields *mwp*-bounds for the output value $x_2'$ in terms of the input values $x_1,x_2,x_3$. The second derivation yields a bound $x_2' \leq \max(x_2, poly(x_1))$, whereas the first one yields a bound $x_2' \leq x_2 + poly(x_1,x_3)$. Thus, the second derivation is the preferred one in the sense that the derivation actually records that we can find a polynomial bound on the value computed into the variable $\mathtt{X}_2$ not depending on the input of $\mathtt{X}_3$.

Finally, let us search for derivations of $\mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2:=\mathtt{X}_2+\mathtt{X}_1\}$. This the command where irreflexive *p*-flow takes place in the loop's body. If we start the derivation by applying (E2) and proceed by applying (A), we get

$$\dfrac{\dfrac{\vdash \mathtt{X}_2+\mathtt{X}_1 : \begin{pmatrix} w \\ w \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_2:=\mathtt{X}_2+\mathtt{X}_1 : \begin{pmatrix} m & w & 0 \\ 0 & w & 0 \\ 0 & 0 & m \end{pmatrix}}}{\vdash \mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_1:=\mathtt{X}_1+\mathtt{X}_2\} : ?}$$

and then we are stuck. The side condition for applying the loop rule (L) is not fulfilled as the closure $\begin{pmatrix} m & w & 0 \\ 0 & w & 0 \\ 0 & 0 & m \end{pmatrix}^* = \begin{pmatrix} m & w & 0 \\ 0 & w & 0 \\ 0 & 0 & m \end{pmatrix}$ has a $w$ on its diagonal. If we start the derivation by applying (E1), and then proceed by (E4) and (A), we get

$$\dfrac{\dfrac{\dfrac{\vdash \mathtt{X}_2: \begin{pmatrix} 0 \\ m \\ 0 \end{pmatrix} \qquad \vdash \mathtt{X}_1: \begin{pmatrix} m \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_2+\mathtt{X}_1 : \begin{pmatrix} p \\ m \\ 0 \end{pmatrix}}}{\vdash \mathtt{X}_2:=\mathtt{X}_1+\mathtt{X}_2 : \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}}}{\vdash \mathtt{loop}\,\mathtt{X}_3\,\{\mathtt{X}_2:=\mathtt{X}_1+\mathtt{X}_1\}: \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & p & m \end{pmatrix}}$$

The loop rule (L) becomes applicable since the closure $\begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}^* = \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}$ has only $m$'s on the diagonal. Note that the rule force us to "add a $p$" in matrix of the conclusion. The $p$ signifies that data $p$-flows from the iteration variable $\mathtt{X}_3$ to the the variable $\mathtt{X}_2$.

# References


[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Co., 1975.

[2] S.J. Bellantoni and S. Cook. *A New Recursion-Theoretic Characterization of the Polytime Functions*. Computational Complexity 2 (1992), 97-110.

[3] G. Bonfante, J.-Y. Marion and J.-Y. Moyen. *Quasi-interpretations – a way to control resources*. Submitted.

[4] V.H. Caseiro *Equations for Defining Poly-time Functions*. PhD thesis, Dept. of Informatics, University of Oslo, February 1997

[5] C.C. Frederiksen. *Automatic runtime analysis for first order functional programs*. Master Thesis, Dep. of Computer Science, University of Copenhagen, 2002.

[6] N.D. Jones. *The expressive power of higher-order types or, life without CONS*. Journal of Functional Programming 11 (2001), 55-94.

[7] N.D. Jones. *LOGSPACE and PTIME characterized by programming languages*. Theoretical Computer Science 228 (1999), 151-174.

[8] N.D. Jones and N. Bohr. *Termination analysis of the untyped lambda-calculus*. Rewriting Techniques and Applications, Springer LNCS Volume 3091 (2004), 1-23.

[9] L. Kristiansen. *Neat function algebraic characterizations of LOGSPACE and LINSPACE*. Computational Complexity 14 (2005), 72-88.

[10] L. Kristiansen and N.D. Jones. *The Flow of Data and the Complexity of Algorithms*. CiE'05:New Computational Paradigms, Springer LNCS 3526 (2005), 263-274.

[11] L. Kristiansen and K.-H. Niggl. *On the computational complexity of imperative programming languages.* Theoretical Computer Science 318 (2004), 139-161.

[12] L. Kristiansen and K.-H. Niggl. *The Garland Measure and Computational Complexity of Stack Programs.* ENTCS 90 (2003), Elsevier.

[13] L. Kristiansen and P.J. Voda. *Complexity classes and fragments of C.* Information Processing Letters 88 (2003), 213-218.

[14] L. Kristiansen and P.J. Voda. *Programming languages capturing complexity classes.* Nordic Journal of Computing 12 (2005), 89-115.

[15] C.S. Lee, N. Jones, and A.M. Ben-Amram. *The size-change principle for program termination.* ACM Principles of Programming Languages ACM Press 2001, 81-92.

[16] D. Leivant. *Intrinsic theories and computational complexity.* Leivant (ed.), "LCC '94: Selected Papers.", Springer (1995) 177-194.

[17] K.-H. Niggl and H. Wunderlich. *Certifying polynomial space and linear/polynomial space for imperative programs*. SIAM Journal on Computing 35 (2006), 1122-1147.