1 Karl-Heinz Niggl: Certifying polynomial time and linear/polynomial space for imperative programs

(Joint work with Henning Wunderlich and Jan Mehler)

In [6], [7], [8], stack/loop programs of μ-measure 0 are shown to characterise the polynomial-time/linear-space computable functions, respectively.

From a programming perspective, these findings might not be practically appealing, for unlike modern programming languages, those programs neither support user-friendly basic instructions, unless they are non-size-increasing, nor mixed data structures or vital instructions like assignment statements.

In this talk, we report on and improve recent research in [9]. There, it is shown how to strengthen the above characterisations to *imperative programs* built from arbitrary basic instructions by sequencing, if-then-else and for-do statements. Each of those programs operates on variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$, each of which may represent any data structure such as stacks, registers, trees or graphs, as long as it is equipped with a (reasonably) notion of *size* of an object stored in $\mathtt{X}_i$, denoted by $|\mathtt{X}_i|$. For example, if $\mathtt{X}_i$ serves as a register, then $|\mathtt{X}_i|$ might be the unary or the binary length of the number stored in $\mathtt{X}_i$, and if $\mathtt{X}_i$ serves as a stack, $|\mathtt{X}_i|$ is as usual the length of the word stored in $\mathtt{X}_i$.

The results stated below rest on a new efficient method of certifying "polynomial size boundedness" under the natural assumption that all basic instructions involved are *polynomial size bounded*, too. For programs $\mathtt{P}$ in variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$, that means there exist polynomials $p_1, \ldots, p_n$ such that

$$\{s_1 = |\mathtt{X}_1|, \ldots, s_n = |\mathtt{X}_n|\}\ \mathtt{P}\ \{|\mathtt{X}_i| \leq p_i(s_1, \ldots, s_n)\} \text{ for } i = 1, \ldots, n.$$

Thus, unlike the measure μ, that method abstracts from the concrete form of basic instructions, and instead focuses on their impact on the polynomial size bounds on the variables involved. As we shall see, polynomial size bounds provide all information on the "control" of one variable over another in a much more subtle way than the measure μ does. Central to the method is that we only store and process a finite amount of information on the class of possible polynomial size bounds for programs. For each polynomial size bound p on $\mathtt{X}_i$ w.r.t. a program $\mathtt{P}$, $p(\vec{\mathtt{X}}) = c_0 + \ldots + c_j \cdot \mathtt{X}_1^{j_1} \cdot \ldots \cdot \mathtt{X}_n^{j_n} + \ldots$ say, we only store an $(n{+}1)$-tuple $\langle p \rangle$ over the *forgetting set* $\{0, 1, \infty\}$, where (for $j = 1, \ldots, n$)

$$\langle p \rangle[j] = \begin{cases} 0 & \text{if } p \text{ is a polynomial in } \vec{\mathtt{X}} \setminus \mathtt{X}_j \\ 1 & \text{if } p = \mathtt{X}_j + q \text{ for some polynomial } q \text{ in } \vec{\mathtt{X}} \setminus \mathtt{X}_j \\ \infty & \text{else} \end{cases}$$

$$\langle p \rangle[n+1] = \begin{cases} c_0 & \text{if } c_0 \leq 1 \\ \infty & \text{else.} \end{cases}$$

In that way, the certificate for a program $\mathtt{P}$ in variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$ will be an $(n{+}1) \times (n{+}1)$ matrix $M(\mathtt{P})$ over $\{0, 1, \infty\}$, where for technical reasons the last row is always the $(n+1)$-tuple $(0, \ldots, 0, 1)$.

For example, a certificate for the assignment statement $\mathtt{X_i} = \mathtt{X_j}$ is obtained from the identity matrix 1_{n+1} by replacing row i with row j. Observe that assignment statements are neither non-size-increasing nor size-increasing.

Altogether, that results into a *matrix calculus for program certificates*. In particular, that calculus provides criteria on the certificate for the body of a loop which guarantee the existence of a certificate for the loop statement itself. We investigate *two forms of loop statements*, `loop` $\mathtt{X_h}$ [`Q`] and `powerloop` $\mathtt{X_h}$ [`Q`], where the body `Q` is executed $|\mathtt{X}_h|$ times for `loop` statements, and $2^{|\mathtt{X}_h|}-1$ times for `powerloop` statements.

Strengthening the results for μ-measure 0 programs, the following theorems are obtained [9].

Theorem A FPTIME = Certified string programs (stack programs built from any polynomial-time computable basic instructions).

Theorem B FLINSPACE = Certified general loop programs (loop programs built from any linear-space computable basic instructions)

Theorem C FPSPACE = Certified power string programs (string programs extended by powerloop statements, and any polynomial-space computable basic instructions)

The improvements over [9] concern the certification of loop statements by generalising the cases “variable/constant assignment” and “push/inc”, to a fairly general *linear case*, leading to much more certified programs and much better extracted polynomial size bounds. In fact, the present efficient method for static verification of program complexity is available as a Java-applet.

We believe that the present method is a major step towards applicability of research in the evolving field of implicit computational complexity to daily programming practice. To exemplify this, natural implementations of binary ADDITION and MULTIPLICATION, and INSERTION SORT are given and certified.

There exist several groups working on static verification of program complexity, e.g. the groups MRG [4]and CRISS [2], in particular [1] and [3], and furthermore [5]. There might be some or even strong connections between the present work and those interesting approaches; but due to the different frameworks an exact comparison is not at all obvious.

Karl-Heinz Niggl
TU Ilmenau, Fakultät für Informatik
niggl@tu-ilmenau.de
eiche.theoinf.tu-ilmenau.de/~niggl

References

[1] Bofante, G., Cichon, A., Marion, J.-Y., Touzet, H.: Algorithms with polynomial interpretation termination proof. JFP **11**, 2000

[2] CRISS (Contrôle de Ressources et d'Interférence dans les Systèmes Synchrones). See http://www.cmi.univ-mrs.fr/ amadio/Criss/criss.html

[3] Moyen, J.-Y.: Analyse de la complexité et transformation de programmes. PhD thesis, Nancy, December 2003. `http://www.loria.fr/ moyen/`

[4] MRG (Mobile Resource Guarantees). `http://groups.inf.ed.ac.uk/mrg/`

[5] Jones, N.D., Kristiansen, L.: *The flow of data and the complexity of algorithms*. Cooper, Löwe, Torenvliet (eds.): CiE'05, LNCS 3526:263-274, Springer 2005.

[6] Kristiansen, L., Niggl, K.-H.: *On the computational complexity of imperative programming languages*. TCS, Special issue on Implicit Computational Complexity, Editor J.-Y. Marion, 318(1-2):139–161, Elsevier 2004.

[7] Niggl, K.-H.: Control Structures in Programs and Computational Complexity. Habilitation Thesis, Ilmenau (2001). Available at the above home page.

[8] Kristiansen, L., Niggl, K.-H.: The Garland Measure and Computational Complexity of Stack Programs. ENTCS 90 No. 2 (2003), `URL: http://elsevier.nl/locate/entcs/volume90.html`, 19 pages

[9] Niggl, K.-H., Wunderlich, H.: Certifying polynomial time and linear/polynomial space for imperative programs. To appear in: SIAM J. Computing.