AHA:

Amortised Heap-Space Usage Analysis

Sponsored by NWO, Grantnr. 612.063.511

Marko van Eekelen, Bart Jacobs, Erik Poll, Sjaak Smetsers and Olha Shkaravska

Security of Systems, Institute for Computing and Information Sciences, Radboud University Nijmegen, the Netherlands
{marko, B.Jacobs, E.Poll, S.Smetsers, shkarav}@cs.ru.nl

Radboud University Nijmegen

NWO
Netherlands Organisation for Scientific Research

Relevance

Estimating heap consumption becomes more and more an issue in many applications, such as programming for *small devices*, e.g. smart cards, mobile phones, embedded systems and *distributed computing*, e.g GRID. Good heap estimates can help to *prevent buffer overflow* and *denial of service attacks*.

Amortization

An *amortized estimate* of a resource does not target a single operation but a *sequence of operations*. One assigns to an operation some amortized cost that may be higher or lower than its actual cost. An amortized cost of the sequence lies between its actual cost and the simple multiplication of the worst-case of one operation by the length of the sequence.

Aim

Amortized analysis is developed for strict functional programs with linear heap consumption. The **AHA** project [1] aims to extend this methodology to *non-linear* bounds within (lazy) functional programs and transfer the results to the object-oriented programming.

First Results: Non-monotonous Polynomial Sizes

To extend linear heap-space analysis to non-linear bounds one needs to know output-on-input size dependencies.

(Polynomially) Shapely Functions

We start with a first-order language over matrix-like structures:
$[[1,\,3],\,[1,\,4],\,[1,\,5]] \in [[\texttt{Int}]^2]^3$

For (polynomially) *shapely* programs the sizes of output structures are (not necessarily monotonous) polynomial functions of the sizes of the inputs. Typical examples of such programs are standard implementations for:

$$\begin{array}{l}\texttt{append} :: [\alpha]^n \times [\alpha]^m \to [\alpha]^{n+m}\\ \texttt{cprod} :: [\alpha]^n \times [\alpha]^m \to [[\alpha]^2]^{n*m}\\ \texttt{sqdiff} :: [\alpha]^n \times [\alpha]^m \to [[\alpha]^2]^{(n-m)^2}\end{array}$$

Sized-type checking

We have shown [2] that checking is in general *undecidable* by reducing it to *Hilbert's tenth problem*. To check

$$\begin{array}{l}\texttt{case}\ f_0(\texttt{x}_1,\ldots,\texttt{x}_n)\ \texttt{of}\ \ \texttt{nil} \Rightarrow \texttt{nil}\\ \qquad\qquad\qquad\qquad\quad \mid (\texttt{h} : \texttt{t}) \Rightarrow (\texttt{1} : \texttt{nil})\end{array}$$

one needs to know if $f_0(\texttt{x}_1,\ldots,\texttt{x}_n)$ has roots or not.

Type-checking is decidable for programs where `case` *is allowed only on program parameters or variables bound to program parameters by other* `case`*-constructs* [2].

Sized-type checking amounts to checking if two polynomial are equal. E.g., for

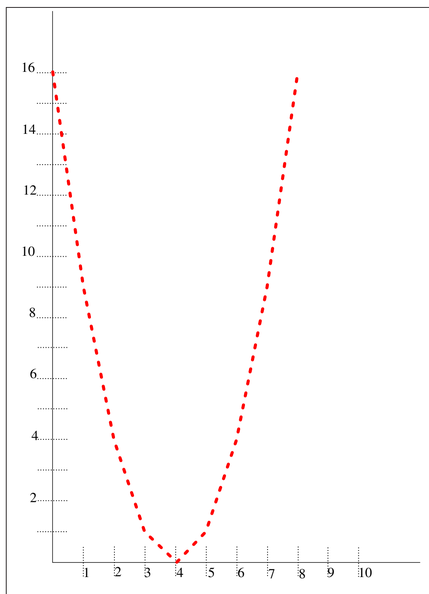
$$\begin{array}{l}\texttt{sqdiff} :: [\alpha]^n \times [\alpha]^m \to [[\alpha]^2]^{n^2+m^2-2nm}\\ \texttt{sqdiff}(\texttt{x},\,\texttt{y}) =\\ \texttt{case}\ \texttt{x}\ \texttt{of}\ \ \texttt{nil} \Rightarrow \texttt{cprod}(\texttt{y},\,\texttt{y})\\ \qquad\qquad \mid (\texttt{h} : \texttt{t}) \Rightarrow \texttt{case}\ \texttt{y}\ \texttt{of}\ \ \texttt{nil} \Rightarrow \texttt{cprod}(\texttt{x},\,\texttt{x})\\ \qquad\qquad\qquad\qquad\qquad\quad \mid (\texttt{h}' : \texttt{t}') \Rightarrow \texttt{sqdiff}(\texttt{t},\,\texttt{t}')\end{array}$$

one has to check

$$\begin{array}{l}n = 0 \ \vdash n^2 + m^2 - 2nm = m^2\\ m = 0 \vdash n^2 + m^2 - 2nm = n^2\\ \qquad\vdash n^2 + m^2 - 2nm = (n-1)^2 + (m-1)^2 - 2(n-1)(m-1)\end{array}$$

Sized-type inference using dynamic testing

Sized-type inference [3] is based on the fact that *a (finite) polynomial is defined by a finite number of points*.



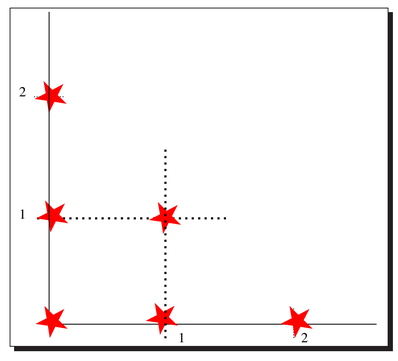
Parabola $(x-4)^2$ is defined by three points, for instance by $(x=0,\,p(x)=16)$, $(x=1,\,p(x)=9)$, $(x=2,\,p(x)=4)$.

By measuring sizes of actual outputs we create a hypothesis to be verified by a type-checker. For instance, testing `sqdiff` gives (note that a test does not always produce a size value):

x	y	n	m	output	$p_2(n,m)$	$p_1(n,m)$
[]	[]	0	0	[]	no value	0
[1]	[]	1	0	[[1, 1]]	2	1
[]	[1]	0	1	[[1, 1]]	2	1
[1]	[1]	1	1	[]	no value	0
[1, 2]	[]	2	1	[[1, 1], [1, 2], [2, 1], [2, 2]]	2	4
[]	[1, 2]	2	1	[[1, 1], [1, 2], [2, 1], [2, 2]]	2	4

We have shown [3] that *for a terminating program one can always find enough test nodes that will assure the existence and uniqueness of a solution of the system of linear equations for the coefficients of the polynomial*.

The "test grid" for `sqdiff` looks as follows:



Assuming that for `sqdiff` we have $p_1(x,\,y) = a_{20}\,x^2 + a_{02}\,y^2 + a_{11}\,x\,y + a_{10}\,x + a_{01}\,y + a_{00}$, we construct from the test results the following system of linear equations:

$$\begin{array}{rl}a_{00} &= 0\\ a_{20} + a_{10} + a_{00} &= 1\\ a_{02} + a_{01} + a_{00} &= 1\\ a_{20} + a_{02} + a_{11} + a_{10} + a_{01} + a_{00} &= 0\\ 4\,a_{20} + 2\,a_{10} + a_{00} &= 4\\ 4\,a_{02} + 2\,a_{01} + a_{00} &= 4\end{array}$$

The unique solution is $a_{20} = a_{02} = 1$, $a_{11} = -2$ with the rest of the coefficients zero. Thus, the hypothesis for the type is $\texttt{sqdiff} :: [\alpha]^n \times [\alpha]^m \to [[\alpha]^2]^{n^2+m^2-2nm}$, which is accepted by the type-checker.

Conclusions

Size checking and inference are important mile-stones on the way to amortized heap analysis. We have formulated the condition under which checking is decidable for (non-monotonous) polynomially shapely programs. We have developed a semi-algorithm of inference of polynomial size dependencies for such programs.

We are working on extending the technique to non-shapely programs and general algebraic data structures.

References

[1] M. van Eekelen, O. Shkaravska, R. van Kesteren, B. Jacobs, E. Poll and S. Smetsers. AHA: Amortized Heap Space Usage Analysis. Project Paper. In Marco Morazán and Henrik Nillsson (eds). *Selected Papers of the Eighth Symposium on Trends in Functional Programming, TFP 2007, New York City, USA, 2-4 april 2007. Intellect Publishers.* To appear.

[2] O. Shkaravska, R. van Kesteren and M. van Eekelen. Polynomial size analysis for first-order functions. In S. Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications (TLCA'2007), Paris, France*, volume 4583 of *LNCS*, pages 351 – 366. Springer, 2007.

[3] R. van Kesteren, O. Shkaravska and M. van Eekelen. Inferring static non-monotonically sized types through testing. In Rachid Echahed, editor, *Preliminary Proceedings of the 16th International Workshop on Functional and (Constraint) Logic Programming (WFLP07), Paris, France*, pages 123 – 139. CNAM, France, 2007.