Sized Type Analysis for Logic Programs

Alejandro Serrano, Pedro Lopez-Garcia,
Francisco Bueno, and Manuel Hermenegildo

ICLP 2013 – August 27, 2013

Description of the Problem and Motivation

Goal

Infer rich descriptions of bounds on run-time term sizes (including subterms)

- Useful on its own as info for programmer (e.g., size, bandwidth verification).
- *Very useful in cost analysis:*

```
listfact([],[]).
listfact([E|R],[F|FR]) :-
  fact(E,F), listfact(R,FR).

fact(0,1).
fact(N,F) :-
  N > 0, N1 is N-1,
  fact(N1,F1), F is N*F1.
```

The number of resolution steps is:

$$1 + \sum_{n \in list} 3n + 1$$

We need both:

- Length of the list, α
- Bounds on the elements, $n \leq \beta$

to derive a good bound, $1 + \alpha(3\beta + 1)$

- No previous analysis for logic programs could infer non-trivial cost bounds in such cases (bounds depend on the sizes of input terms and their subterms).
- However, these programs are very common!

Our Solution: Sized Types (and Their Inference)

We encode bounds on the size of a term and its subterms in *sized types:*

- The shape of *sized types* is derived from the types of the arguments.
- We extend the domain of regular types (Dart and Zobel, etc.).

```
listnum -> []
listnum -> .(num, listnum)
```

$$listnum^{(\alpha,\beta)}\left(num^{(\gamma,\delta)}_{\langle\cdot,1\rangle}\right)$$

- The superscripts express bounds on the number of rule (functor) applications.
- Each argument expresses bounds on the subterms, in every position that is not recursive.

```
{ [1,2,3,4], [2,4] }
```

$$listnum^{(3,5)}\left(num^{(1,4)}_{\langle\cdot,1\rangle}\right)$$

We infer such sized types *using abstract interpretation*.

Relations Between Sized Types

The analysis finds relations (inequalities) between sized types with variables.

- These are called *sized type schemas*.
- We use a special notation for these relations.

How do we derive these relations? Setting up and solving *recurrence equations*.

- Powerful method, covers many types of equations with good performance.
- Unfortunately, not a complete method.

Our Contributions to the State-of-the-art

- Developed a *novel sized type analysis using abstract interpretation* and integrated in CiaoPP (using PLAI):
  - Allows relating (lower and upper bounds on) the sizes of terms *and subterms (at any depth)* occurring at different argument positions in logic predicates.
  - Views and processes sized types as an *abstract domain*.
  - This allows using a standard abstract interpreter (PLAI) and then some advanced features come for free, such as *multivariance*: analysis results for different call patterns of the same predicate.
- Assessed both the accuracy of the new size analysis and its usefulness in resource usage estimation:
  - Developed novel resource usage and cardinality analyses, also using abstract interpretation and integrated in CiaoPP (detailed in WLPE 2013).
  - The proposed sized types are a substantial improvement.
  - They benefit the resource analysis considerably.

Analysis Example

Given a call pattern `listfact(+L, -FL)` we want to analyze

```
listfact([E|R], [F|FR]) :- fact(E, F), listfact(R, FR).
```

$$
\begin{array}{rcll}
L & \rightarrow & listnum^{(a_1,b_1)}(num^{(c_1,d_1)}), & input \\
FL & \rightarrow & listnum^{(a_2,b_2)}(num^{(c_2,d_2)}), & output \\
E & \rightarrow & num^{(c_3,d_3)}, & clausal \\
R & \rightarrow & listnum^{(a_4,b_4)}(num^{(c_4,d_4)}), & clausal \\
F & \rightarrow & num^{(c_5,d_5)}, & clausal \\
FR & \rightarrow & listnum^{(a_6,b_6)}(num^{(c_6,d_6)}), & clausal
\end{array}
$$

$$a_1 > 1, b_1 > 1$$

$$c_3 = c_1, d_3 = d_1,$$

$$a_4 = a_1 - 1, b_4 = b_1 - 1, c_4 = c_1, d_4 = d_1,$$

$$c_5 = fact_c(c_3), d_5 = fact_d(d_3)$$

$$a_6 = listfact_a(a_4, b_4, c_4, d_4), b_6 = listfact_b(\ldots), c_6 = listfact_c(\ldots), d_6 = listfact_d(\ldots),$$

$$a_2 = a_6 + 1, b_2 = b_6 + 1, c_2 = \min(c_5, c_6), d_2 = \max(d_5, d_6)$$

Multivariance and Subtyping

Multivariance is the ability to return different analysis results for different call patterns of the same predicate.

- The widening operator ∇ developed identifies which recurrence relations refer to the same *predicate version*.

Regular types support *structural subtyping:*

- It is crucial for analyzing many predicates.
- We have developed an inclusion algorithm, based on that of Dart and Zobel, which also returns relations between sized types.

Experimental Results

Program	Size An.	Resource An. (LB)			Resource An. (UB)				
		New	*Previous*		*New*	*Previous*		*RAML*	
append	**+1**	α	α	=	β	β	=	β	=
appendAll2	**+3**	$a_1 a_2 a_3$	a_1	+	$b_1 b_2 b_3$	∞	+	$b_1 b_2 b_3$	=
`coupled`	+1	μ	0	+	ν	∞	+	ν	=
`dyade`	+2	$\alpha_1\alpha_2$	$\alpha_1\alpha_2$	=	$\beta_1\beta_2$	$\beta_1\beta_2$	=	$\beta_1\beta_2$	=
`erathos`	+1	α	α	=	β^2	β^2	=	β^2	=
fib	=	ϕ^μ	ϕ^μ	=	$\boldsymbol{\phi^\nu}$	ϕ^ν	=	**infeasible**	+
`hanoi`	=	1	0	+	2^ν	∞	+	infeasible	+
`isort`	+1	α^2	α^2	=	β^2	β^2	=	β^2	=
isortlist	+2	a_1^2	a_1^2	=	$\mathbf{b_1^2 b_2}$	$\boldsymbol{\infty}$	+	$b_1^2 b_2$	=
`listfact`	+1	$\alpha\gamma$	α	+	$\beta\delta$	∞	+	unknown	?
`listnum`	+1	μ	μ	=	ν	ν	=	unknown	?
`minsort`	+1	α^2	α	+	β^2	β^2	=	β^2	=
`nub`	+2	a_1	a_1	=	$b_1^2 b_2$	∞	+	$b_1^2 b_2$	=
`partition`	+1	α	α	=	β	β	=	β	=
`zip3`	+1	$\min(\alpha_i)$	0	+	$\min(\beta_i)$	∞	+	β_3	+

Thank you

Thanks for your attention