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:

\[ 1 + \sum_{n \in \text{list}} 3n + 1 \]

We need both:

- Length of the list, \( \alpha \)
- 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.).

\[
\text{listnum} \rightarrow [] \\
\text{listnum} \rightarrow .(\text{num, listnum}) \\
\text{listnum}^{(\alpha, \beta)} \left( \text{num}^{(\gamma, \delta)} \right)
\]

- 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.

\[
\{ [1,2,3,4], [2,4] \} \\
\text{listnum}^{(3,5)} \left( \text{num}^{(1,4)} \right)
\]

We infer such sized types using abstract interpretation.
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.
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 \textit{and subterms (at any depth)} occurring at different argument positions in logic predicates.
- Views and processes sized types as an \textit{abstract domain}.
- This allows using a standard abstract interpreter (PLAI) and then some advanced features come for free, such as \textit{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.
Given a call pattern \texttt{listfact(+L, -FL)} we want to analyze

\texttt{listfact([E|R], [F|FR]) :- fact(E, F), listfact(R, FR).}

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

\[ a_1 > 1, b_1 > 1 \]

\[ c_3 = c_1, d_3 = d_1, \]
\[ a_4 = a_1 - 1, b_4 = b_1 - 1, c_4 = c_1, d_4 = d_1, \]
\[ c_5 = \text{fact}_c(c_3), d_5 = \text{fact}_d(d_3) \]
\[ a_6 = \text{listfact}_a(a_4, b_4, c_4, d_4), b_6 = \text{listfact}_b(\ldots), c_6 = \text{listfact}_c(\ldots), d_6 = \text{listfact}_d(\ldots), \]
\[ a_2 = a_6 + 1, b_2 = b_6 + 1, c_2 = \min(c_5, c_6), d_2 = \max(d_5, d_6) \]
Multivariance and Subtyping

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

- The widening operator $\nabla$ 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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>Previous</td>
</tr>
<tr>
<td>append</td>
<td>+1</td>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>appendAll2</td>
<td>+3</td>
<td>$a_1 a_2 a_3$</td>
<td>$a_1$</td>
</tr>
<tr>
<td>coupled</td>
<td>+1</td>
<td>$\mu$</td>
<td>0</td>
</tr>
<tr>
<td>dyade</td>
<td>+2</td>
<td>$\alpha_1 \alpha_2$</td>
<td>$\alpha_1 \alpha_2$</td>
</tr>
<tr>
<td>erathos</td>
<td>+1</td>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>fib</td>
<td>=</td>
<td>$\phi^\mu$</td>
<td>$\phi^\mu$</td>
</tr>
<tr>
<td>hanoi</td>
<td>=</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>isort</td>
<td>+1</td>
<td>$\alpha^2$</td>
<td>$\alpha^2$</td>
</tr>
<tr>
<td>isortlist</td>
<td>+2</td>
<td>$a_1^2$</td>
<td>$a_1^2$</td>
</tr>
<tr>
<td>listfact</td>
<td>+1</td>
<td>$\alpha \gamma$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>listnum</td>
<td>+1</td>
<td>$\mu$</td>
<td>$\mu$</td>
</tr>
<tr>
<td>minsort</td>
<td>+1</td>
<td>$\alpha^2$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>nub</td>
<td>+2</td>
<td>$a_1$</td>
<td>$a_1$</td>
</tr>
<tr>
<td>partition</td>
<td>+1</td>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>zip3</td>
<td>+1</td>
<td>$\min(\alpha_i)$</td>
<td>0</td>
</tr>
</tbody>
</table>
Thank you

Thanks for your attention