Amortised memory analysis using the depth of data structures

Brian Campbell

Brian.Campbell@ed.ac.uk

Laboratory for Foundations of Computer Science
University of Edinburgh

March 26, 2009

Overview

Interested in efficient memory bounds inference.

- Hofmann-Jost [POPL'03] amortised analysis gives **linear heap** space bounds in **total size** of the input.
- Such bounds are a poor fit for stack space.
  Especially for functional programming, tree structured data.
- Want *maxima* and bounds in terms of *depth*.

Keep close to efficient Linear Programming inference.

Principles of Hofmann-Jost analyses

- The type system *certifies* bounds;
- annotations describe bounding function in terms of input sizes:

$$x : \mathrm{tree}(\mathrm{bool}, 2),\ y : \mathrm{tree}(\mathrm{bool}, 3),\ 5 \ \vdash e : \ \mathrm{tree}(\mathrm{bool}, 2), 1$$

$$2 \times |x| \quad + \quad 3 \times |y| \quad +5$$

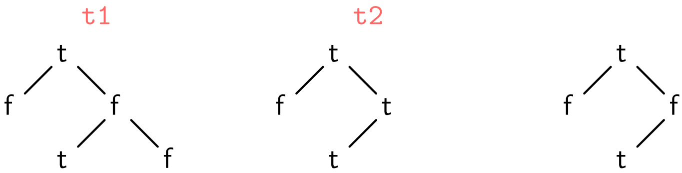
$$\longrightarrow \quad 2 \times |\mathit{result}| + 1$$

- Side conditions in rules guarantee the soundness of the bounding functions.

Inference:
- construct ‘plain’ typing
- collect side conditions together
- solve the resulting LP.

Heap memory example

The `andtrees` function computes the pointwise 'and' of two boolean trees (up to the largest common subtree):

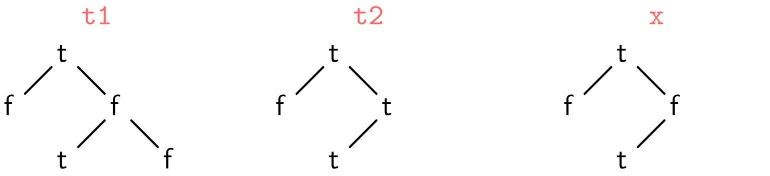


$$\text{tree}(\text{bool}, 1) \quad \times \quad \text{tree}(\text{bool}, 0), \quad 0 \quad \rightarrow \quad \text{tree}(\text{bool}, 0), \quad 0$$
$$\text{tree}(\text{bool}, 0) \quad \times \quad \text{tree}(\text{bool}, 1), \quad 0 \quad \rightarrow \quad \text{tree}(\text{bool}, 0), \quad 0$$

- means `andtrees t1 t2` uses no more than |`t1`| units of space.
- The typings (and bounds) are not unique. |`t2`| is also sufficient.

Heap memory example

The `andtrees` function computes the pointwise 'and' of two boolean trees (up to the largest common subtree):

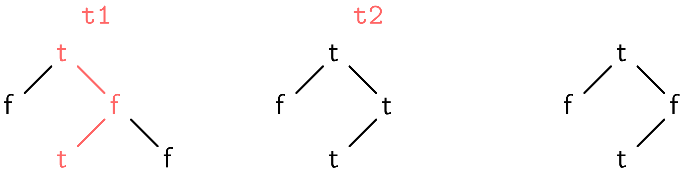


$$\text{tree}(\text{bool}, 2) \quad \times \quad \text{tree}(\text{bool}, 0), \quad 4 \quad \rightarrow \quad \text{tree}(\text{bool}, 1), \quad 4$$

`let x = andtrees t1 t2 in` e_2

- Signatures also 'translate' requirements:
- If e_2 requires $|x| + 4$ units, then $2 \times |y| + 4$ is sufficent for both allocation and $|x| + 4$ later.

Example with stack space



$$\mathsf{tree}(\mathsf{bool}, 1) \quad \times \quad \mathsf{tree}(\mathsf{bool}, 0), \quad 0 \quad \rightarrow \quad \mathsf{tree}(\mathsf{bool}, 1), \quad 0$$
$$\mathsf{tree}(\mathsf{bool}, 0) \quad \times \quad \mathsf{tree}(\mathsf{bool}, 1), \quad 0 \quad \rightarrow \quad \mathsf{tree}(\mathsf{bool}, 1), \quad 0$$

- means `andtrees t1 t2` uses at most $|\texttt{t1}|$ (or $|\texttt{t2}|$) units of stack space.
- Stack space is reusable.
- But now we want to use the depth to get a better bound (i.e., $|\texttt{t1}|_{\mathrm{d}}$).

Developing an analysis with maximums

Previously we just added all the contributions from the context:

$$\begin{array}{cccccccc} l:\mathrm{tree}(\mathrm{bool},k), & r:\mathrm{tree}(\mathrm{bool},k), & v:\mathrm{bool}, & n \vdash e : \ldots \\ |l| \times k & + & |r| \times k & + & 0 & +n \end{array}$$

Now we introduce a second context former to denote 'max' (;):

$$\begin{array}{l} (l:\mathrm{tree}(\mathrm{bool},k);\ r:\mathrm{tree}(\mathrm{bool},k); v:\mathrm{bool}),\ \ n \vdash e : \ldots \\ \max\{ \quad |l|_d \times k \quad , \quad |r|_d \times k \quad , \quad 0 \quad \} + n \end{array}$$

- Note that contexts are now trees.
- Treat tree types as 'folded up' version of above context.
- So $t:\mathrm{tree}(\mathrm{bool},k)$ denotes $|t|_{\mathrm{d}} \times k$.

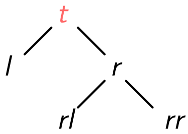
Inspired by O'Hearn's Bunched Typing.

Unfolding trees in the context

$\Gamma()$ is a context with a 'hole'.

$$\frac{\Gamma(\cdot) \vdash e_1 : T, k' \qquad \Gamma((l : \mathrm{tree}(T, k); r : \mathrm{tree}(T, k); v : T), k) \vdash e_2 : T, k'}{\Gamma(t : \mathrm{tree}(T, k)) \vdash \mathrm{match}\ t\ \mathrm{with}\ \ \mathrm{leaf} \rightarrow e_1 \mid \mathrm{node}(l, v, r) \rightarrow e_2 : T, k'} \ (\textsc{Match})$$

$\longrightarrow$



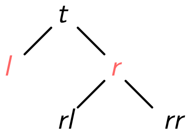
$t : \mathrm{tree}(\mathrm{bool}, k)$

Unfolding trees in the context

$\Gamma()$ is a context with a 'hole'.

$$\frac{\Gamma(\cdot) \vdash e_1 : T, k' \qquad \Gamma((l : \mathrm{tree}(T, k); r : \mathrm{tree}(T, k); v : T), k) \vdash e_2 : T, k'}{\Gamma(t : \mathrm{tree}(T, k)) \vdash \mathrm{match}\ t\ \mathrm{with}\ \ \mathrm{leaf} \rightarrow e_1 \mid \mathrm{node}(l, v, r) \rightarrow e_2 : T, k'} \ (\textsc{Match})$$

$\longrightarrow$



$$t : \mathrm{tree}(\mathrm{bool}, k)$$

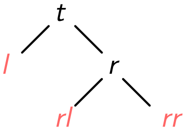
$$(l : \mathrm{tree}(\mathrm{bool}, k); r : \mathrm{tree}(\mathrm{bool}, k); v : \mathrm{bool}), k$$

Unfolding trees in the context

$\Gamma()$ is a context with a 'hole'.

$$\frac{\Gamma(\cdot) \vdash e_1 : T, k' \qquad \Gamma((l : \mathrm{tree}(T, k); r : \mathrm{tree}(T, k); v : T), k) \vdash e_2 : T, k'}{\Gamma(t : \mathrm{tree}(T, k)) \vdash \mathrm{match}\ t\ \mathrm{with}\ \ \mathrm{leaf} \rightarrow e_1 \mid \mathrm{node}(l, v, r) \rightarrow e_2 : T, k'} \ (\textsc{Match})$$

$\rightarrow$



$$t : \mathrm{tree}(\mathrm{bool}, k)$$

$$(l : \mathrm{tree}(\mathrm{bool}, k); r : \mathrm{tree}(\mathrm{bool}, k); v : \mathrm{bool}), k$$

$$\Big(l : \mathrm{tree}(\mathrm{bool}, k); \big((rl : \mathrm{tree}(\mathrm{bool}, k); rr : \mathrm{tree}(\mathrm{bool}, k); rv : \mathrm{bool}), k\big); v : \mathrm{bool}\Big), k$$

let expressions

$$\text{let } x = e_1 \text{ in } e_2$$

- Overall bound is $\max\{\text{bound for } e_1, \text{bound for } e_2\}$.
- But we also need to translate bound for e_2.

Instead replace subcontext for x with that needed to produce it, Δ:

$$\frac{\Delta \vdash e_1 : T_0, n_0 \qquad \Gamma(x : T_0, n_0) \vdash e_2 : T, k'}{\Gamma(\Delta) \vdash \text{let } x = e_1 \text{ in } e_2 : T, k'} \quad (\textsc{Let})$$

(Only sound for stack discipline.)

New rules

We need to be able to manipulate contexts to get the right shape. Hence new rules such as:

$$\frac{\Gamma(\Delta') \vdash e : T, n' \qquad \Delta \cong \Delta'}{\Gamma(\Delta) \vdash e : T, n'} \quad (\equiv)$$

$$\Gamma, (\Delta; \Delta') \cong (\Gamma, \Delta); (\Gamma, \Delta') \quad \text{(distribution)}$$

$$\Gamma \cong \Gamma; \Gamma \quad \text{(max-contraction)}$$

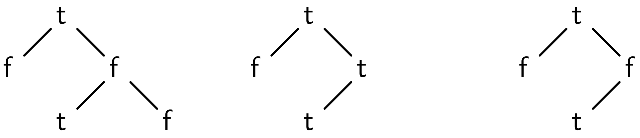
$$\Gamma \cong q\Gamma, (1 - q)\Gamma \quad q \in [0, 1] \quad \text{(plus-contraction)}$$

All preserve the bounding functions derived from the context.

Also: weakening and a max-to-plus approximate conversion.

Example with stack space

Function signatures are also 'structured'.

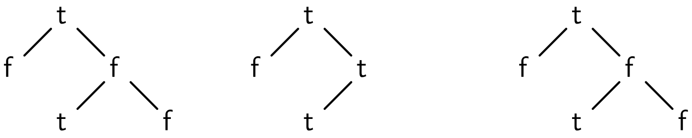


$$\mathrm{tree}(\mathrm{bool}, 1) \quad ; \quad \mathrm{tree}(\mathrm{bool}, 0) \quad \rightarrow \quad \mathrm{tree}(\mathrm{bool}, 1)$$
$$\mathrm{tree}(\mathrm{bool}, 0) \quad ; \quad \mathrm{tree}(\mathrm{bool}, 1) \quad \rightarrow \quad \mathrm{tree}(\mathrm{bool}, 1)$$

- means `andtrees t1 t2` uses at most $|\texttt{t1}|_d$ or $|\texttt{t2}|_d$ units of stack space.

Example with stack space

Function signatures are also ‘structured’.

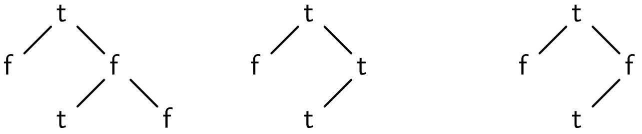


$$\text{tree}(\text{bool}, 1) \quad ; \quad \text{tree}(\text{bool}, 1) \quad \rightarrow \quad \text{tree}(\text{bool}, 1)$$

- means `andtreesmax t1 t2` uses at most $\max\{|\texttt{t1}|_d, |\texttt{t2}|_d\}$ units of stack space.
- We can now also type a version of `andtrees` which returns false for all the nodes which are only in one of the arguments.

Example with stack space

Function signatures are also 'structured'.



$$\mathsf{tree}(\mathsf{bool}, 1) \quad ; \quad \mathsf{tree}(\mathsf{bool}, 1) \quad \to \quad \mathsf{tree}(\mathsf{bool}, 1)$$

$$\frac{k \geq \mathrm{stack}(f) \qquad k + k_1' \geq k' \qquad \Sigma(f) = \Gamma \to T, k_1'}{\Gamma[x_1, \ldots, x_p/\mathrm{namesof}(\Gamma)], k \vdash f(x_1, \ldots, x_p) : T, k'} \ (\textsc{Fun})$$

Extra benefit from maxima

```
let maybeleft(t,b) =
  match t with leaf -> leaf | node(l,r,v) ->
    if b then l else t
```

In heap analysis need to sum requirements because of use of contraction at `match`. Doubles the bound unnecessarily.

- In depth type system we can use max-contraction.
- So requirement goes $|t| \Rightarrow \max\{|t|, |t|\} \Rightarrow \max\{|t|, |l|\}$.
- Context goes $t : \text{tree} \Rightarrow t : \text{tree}; t : \text{tree} \Rightarrow t : \text{tree}; l : \text{tree}$

Stack space inference

- Would like to take advantage of linear programming again
- But new context manipulation rules are not syntax-directed

We add an extra stage to the inference process:

source program (plain types)

$\downarrow$

additional terms for context manipulation

$\downarrow$

bound (using linear programming)

Assume context structure given for function signatures to make problem more tractable.

Basic ideas for inference

- Work from the leaves of the expression outwards.
- At every stage, keep track of a generated context derived from subexpressions and the typing rule.

$$\frac{\Gamma \vdash e_1 \mapsto \Gamma_1 \qquad \Gamma \vdash e_2 \mapsto \Gamma_2}{\Gamma \vdash \text{if } x \text{ then } e_1 \text{ else } e_2 \mapsto \Gamma_1; \Gamma_2; x : \text{bool}}$$

Need to add context manipulation at two points:

1. where binding occurs, to deal with contraction, etc;
2. to make the generated context match the function signature.

Expand context to max-of-sums form, factor out bound variables.

The full analysis

- Have algebraic data types, not just trees.
- Can specify the form of bounds:

  in terms of *depth*, *total size*, or a *mixture*.

  Bounds w.r.t. total size useful when depth analysis fails.
- Resource polymorphism (different function signatures at different points).

Implementation in Standard ML.

Conclusion and Further work

Type-based amortized analysis can provide good bounds on stack space for programs using tree-structured data.

- Nested containers don't behave that well due to the definition of depth.
  May be possible to separate structure of containers from their contents.
- Inferring the structure of function signatures.
- Deal with construction of log-depth trees (e.g, heap sort).

Potentially the most interesting:

Find sound LET rule for heap space, get max-plus bounds for heap.