Amortised memory analysis using the *depth* of data structures

Brian Campbell

Brian.Campbell@ed.ac.uk

Laboratory for Foundations of Computer Science

September 9, 2008

Principles of Hofmann-Jost-style analyses

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

$$x : \text{bool tree}[2],\ y : \text{bool tree}[3],\ 5 \ \vdash e : \ \text{bool tree}[2], 1$$

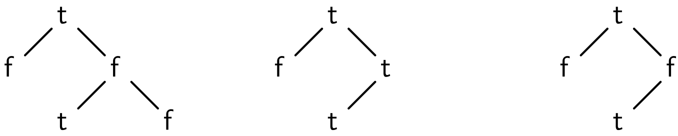
$$2 \times |x| \quad + \quad 3 \times |y| \quad +5 \quad | \quad 2 \times |\mathit{result}| + 1$$

- Side conditions guarantee bounding functions sound.

Inference by collecting conditions together and solve resulting LP.

Heap memory example

The `andtrees` function computes the pointwise ‘and’ of two boolean trees (up to the smaller tree):

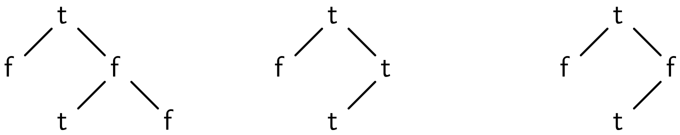


$$\text{bool tree } [1] \times \text{bool tree } [0], \quad 0 \quad \to \quad \text{bool tree } [0], \quad 0$$
$$\text{bool tree } [0] \times \text{bool tree } [1], \quad 0 \quad \to \quad \text{bool tree } [0], \quad 0$$

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

Heap memory example

The `andtrees` function computes the pointwise ‘and’ of two boolean trees (up to the smaller tree):

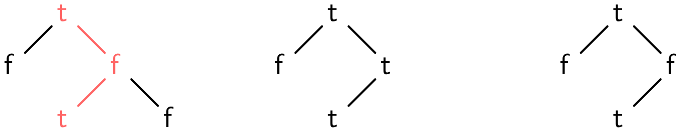


$$\text{bool tree } [2] \quad \times \quad \text{bool tree } [0], \quad 4 \quad \rightarrow \quad \text{bool tree } [1], \quad 4$$

```
let x = andtrees y z in ...
```

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

Example with stack space



$$\text{bool tree } [1] \times \text{bool tree } [0],\ 0 \to \text{bool tree } [1],\ 0$$
$$\text{bool tree } [0] \times \text{bool tree } [1],\ 0 \to \text{bool tree } [1],\ 0$$

- 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:

$$\begin{array}{ccccccc} l : \text{bool tree } [k], & & r : \text{bool tree } [k], & & v : \text{bool}, & & n \vdash e : \ldots \\ |l| \times k & + & |r| \times k & + & 0 & + & n \end{array}$$

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

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

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

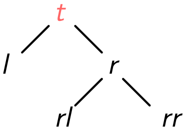
Inspired by O'Hearn's Bunched Typing.

Unfolding trees in the context

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

$$\frac{\Gamma(\cdot) \vdash e_1 : T, k' \qquad \Gamma((l : \text{bool tree}[k]; r : \text{bool tree}[k]; v : \text{bool}), k) \vdash e_2 : T, k'}{\Gamma(t : \text{bool tree}[k]) \vdash \text{match } t \text{ with } \begin{array}{l} \ \text{leaf} \to e_1 \\ \mid \text{node}(l, v, r) \to e_2 : T, k' \end{array}} \text{(TreeMatch)}$$

$\longrightarrow$



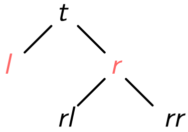
$t : \text{bool tree}[k]$

Unfolding trees in the context

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

$$\frac{\Gamma(\cdot) \vdash e_1 : T, k' \qquad \Gamma((l : \text{bool tree}[k]; r : \text{bool tree}[k]; v : \text{bool}), k) \vdash e_2 : T, k'}{\Gamma(t : \text{bool tree}[k]) \vdash \text{match } t \text{ with } \begin{array}{l} \text{leaf} \longrightarrow e_1 \\ \mid \text{node}(l, v, r) \longrightarrow e_2 : T, k' \end{array}} \; (\textsc{TreeMatch})$$

$\longrightarrow$



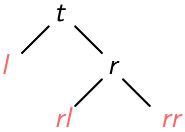
$(l : \text{bool tree}[k]; r : \text{bool tree}[k]; v : \text{bool}), k$

Unfolding trees in the context

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

$$\frac{\Gamma(\cdot) \vdash e_1 : T, k' \qquad \Gamma((l : \text{bool tree}[k]; r : \text{bool tree}[k]; v : \text{bool}), k) \vdash e_2 : T, k'}{\Gamma(t : \text{bool tree}[k]) \vdash \text{match } t \text{ with } \begin{array}{l} \ \text{leaf} \to e_1 \\ \mid \text{node}(l, v, r) \to e_2 : T, k' \end{array}} \ (\textsc{TreeMatch})$$

$\longrightarrow$



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

New rules

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

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

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

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

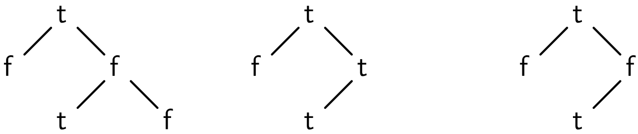
$$\Gamma \cong q\Gamma, (1-q)\Gamma \quad q \in [0,1] \quad \text{(plus-contraction)}$$

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’.

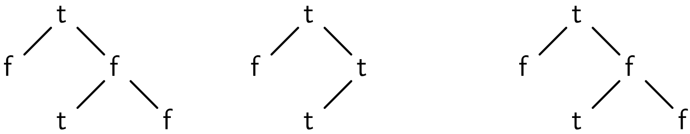


$$\text{bool tree}[1] \;;\; \text{bool tree}[0] \rightarrow \text{bool tree}[1]$$
$$\text{bool tree}[0] \;;\; \text{bool tree}[1] \rightarrow \text{bool tree}[1]$$

- 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’.

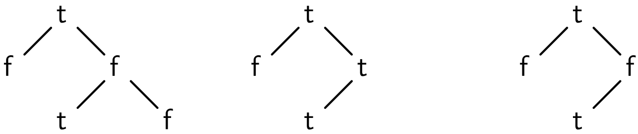


bool tree[1] ; bool tree[1] $\rightarrow$ bool tree[1]

- 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'.



bool tree[1] ; bool tree[1] $\rightarrow$ bool tree[1]

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

Extra benefit from maxima

```
let maybetail(l,b) =
  match l with cons(h,t)' ->
    if b then t else l
```

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 $|l| \Rightarrow \max\{|l|, |l|\} \Rightarrow \max\{|t|, |l|\}$.
- Context goes l : list $\Rightarrow$ l : list; l : list $\Rightarrow$ t : list; l : list

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, Γ_1:

$$\frac{\Gamma_1 \vdash e_1 : T_1, n_1 \qquad \Gamma(x : T_1, n_1) \vdash e_2 : T, n'}{\Gamma(\Gamma_1) \vdash \text{let } x = e_1 \text{ in } e_2 : T, n'} \qquad (\textsc{Let})$$

(Only sound for stack discipline.)

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.

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

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.

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.

Further work

- Nested types don't behave that well. Have done some work on separating contents and structure
- Inferring the structure of function signatures.
- Reduce complexity of inference.
- Deal with construction of log-depth trees.
- Try heap space version.