Type-based amortized resource prediction

Brian Campbell
Brian.Campbell@ed.ac.uk

Laboratory for Foundations of Computer Science
University of Edinburgh

March 17, 2008

Overview

Wouldn't we all like memory predictions for our programs that are

- given in terms of the input.
- actual bounds on the amounts required,
- automatically generated,
- certifiable?

Limitations

What are we prepared to give up?

- Only get linear bounds in terms of the input,
  (surprisingly common)
- no relying on fancy reasoning,
- especially no fancy invariants,
- start with first-order functional language and work up.

Outline

Idea for Hofmann-Jost

A type system which shows bounds are good:

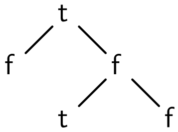
- Assigns ‘free’ memory to input (proportional to size);
- play game of pass the parcel / carbon credit trading / *your analogy here*;
- typing rules enforce no sneaky increases, and all allocations paid for.

Then we add some inference:

- All the side conditions on assignments are linear (in)equalities;
- reduce to linear programming!

Heap memory example

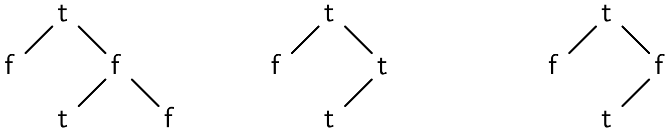
The `andtrees` function computes the pointwise ‘and’ of two boolean trees (up to the smaller tree):



booltree $\times$ booltree $\rightarrow$ booltree

Heap memory example

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

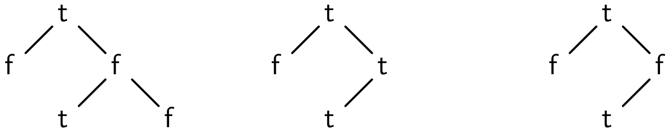


$$\text{booltree}\,[1] \quad \times \quad \text{booltree}\,[0], \quad 0 \quad \to \quad \text{booltree}\,[0], \quad 0$$

- means `andtrees t1 t2` uses no more than $|$`t1`$|$ units of space.

Heap memory example

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

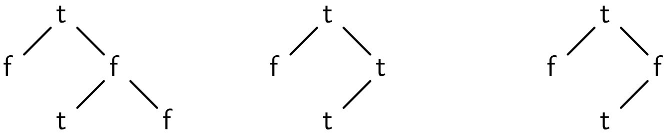


$$\begin{array}{lclcl}
\text{booltree }[1] & \times & \text{booltree }[0], \quad 0 & \to & \text{booltree }[0], \quad 0 \\
\text{booltree }[0] & \times & \text{booltree }[1], \quad 0 & \to & \text{booltree }[0], \quad 0
\end{array}$$

- 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):



$$\text{booltree}\,[2] \quad \times \quad \text{booltree}\,[0], \quad 4 \quad \to \quad \text{booltree}\,[1], \quad 4$$

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

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

Hofmann-Jost is an amortized analysis

$\texttt{t1}\!:\!\mathsf{booltree}\ [2], \texttt{t2}\!:\!\mathsf{booltree}\ [0], 4 \vdash \texttt{andtrees t1 t2} : \mathsf{booltree}\ [1], 4$

The type annotations define *potential* functions

$$\Upsilon_{\Gamma}(t1, t2) = |t1| \times 2 + |t2| \times 0 + 4$$

for the context, and for the result:

$$\Upsilon_{R}(r) = |r| \times 1 + 4.$$

Constraints ensures that the allocation is accounted for by a drop in potential. (See *Physicist's view* in Tarjan 1985)

Hofmann-Jost rules — construction

$$\frac{n \geq \text{size(booltree node)} + k + n'}{l : \text{booltree}[k], r : \text{booltree}[k], v : \text{bool}, n \vdash \text{node}(l, v, r) : \text{booltree}[k], n'} \quad (\textsc{Node})$$

means that if we have

$$|l| \times k + |r| \times k + n$$

units of free memory then we can allocate the node and end up with

$$|\text{node}(l, v, r)| \times k + n' = (1 + |l| + |r|) \times k + n'.$$

Hofmann-Jost rules — matching

$$\frac{\Gamma, n \vdash e_1 : T, n' \qquad n + k = n_2 \qquad \Gamma, l : \mathrm{booltree}[k], r : \mathrm{booltree}[k], v : \mathrm{bool}, n_2 \vdash e_2 : T, n'}{\Gamma, t : \mathrm{booltree}[k], n \vdash \text{match } t \text{ with leaf}' \to e_1 \mid \mathrm{node}(l, r, v)' \to e_2} \ (\textsc{TreeElim}')$$

We 'reserved' k units of memory when we constructed the node, so we get it back now.

The $'$ means the data might still be live.

Hofmann-Jost rules — matching

$$\frac{\begin{array}{c}\Gamma, n \vdash e_1 : T, n' \\ n + k + \mathrm{size}(\text{booltree node}) = n_2 \\ \Gamma, l : \text{booltree}[k], r : \text{booltree}[k], v : \text{bool}, n_2 \vdash e_2 : T, n'\end{array}}{\begin{array}{l}\Gamma, t : \text{booltree}[k], n \vdash \text{match } t \text{ with leaf} \to e_1 \\ \qquad\qquad\qquad\qquad\qquad\qquad \mid \text{node}(l, r, v) \to e_2\end{array}} \;(\textsc{TreeElim})$$

We 'reserved' k units of memory when we constructed the node, so we get it back plus the memory freed up.

No $'$, so we get the memory back.
(Assumes external safety checker.)

Hofmann-Jost rules — contraction

$$\frac{k = k_1 + k_2 \qquad \Gamma, x_1 : \mathsf{booltree}[k_1], x_2 : \mathsf{booltree}[k_2], n \vdash e : T', n'}{\Gamma, x : \mathsf{booltree}[k], n \vdash e[x/x_1, x/x_2] : T', n'} \ (\textsc{Share})$$

If $|x_1| \times k_1 + |x_2| \times k_2$ is sufficient for e, then $|x| \times k$ is sufficient for $e[x/x_1, x/x_2]$.

Weakening of variables is admissible. Together with Share, we get weakening of annotations.

Example revisited

`andtrees t1 t2`

$$\text{booltree}\ [k_1] \quad \times \quad \text{booltree}\ [k_2]\ ,\ \ n \quad \rightarrow \quad \text{booltree}\ [k']\ ,\ \ n'$$

$$|t1| \times k_1 \quad + \quad |t2| \times k_2 \quad + n \quad \rightarrow \quad |r| \times k' \quad + n'$$

$n \geq n'$	for leaf cases
$n + k_1 = n_1$	`t1` node match
$n_1 + k_2 = n_2$	`t2` node match

Example revisited

`andtrees t1 t2`

$$\begin{array}{ccccccc} \text{booltree}\,[k_1] & \times & \text{booltree}\,[k_2]\,,\ n & \rightarrow & \text{booltree}\,[k']\,,\ n' \\ |t1| \times k_1 & + & |t2| \times k_2 \ + n & \rightarrow & |r| \times k' \ + n' \end{array}$$

$$\begin{array}{rl} n \geq n' & \text{for leaf cases} \\ n + k_1 = n_1 & \texttt{t1}\ \text{node match} \\ n_1 + k_2 = n_2 & \texttt{t2}\ \text{node match} \\ n_2 \geq n,\ n_2 - n + n' \geq n_3 & \text{left recursive call} \\ n_3 \geq n,\ n_3 - n + n' \geq n_4 & \text{right recursive call} \end{array}$$

Example revisited

```
andtrees t1 t2
```

$$\text{booltree } [k_1] \quad \times \quad \text{booltree } [k_2]\,,\ n \quad \to \quad \text{booltree } [k']\,,\ n'$$

$$|t1| \times k_1 \quad + \quad |t2| \times k_2 \quad + n \quad \to \quad |r| \times k' \quad + n'$$

$n \geq n'$	for leaf cases
$n + k_1 = n_1$	`t1` node match
$n_1 + k_2 = n_2$	`t2` node match
$n_2 \geq n,\ \ n_2 - n + n' \geq n_3$	left recursive call
$n_3 \geq n,\ \ n_3 - n + n' \geq n_4$	right recursive call
$k_1 \geq k'$	weakening
$k_2 \geq k'$	weakening
$n_4 \geq \mathrm{size}(\text{node}) + k' + n'$	for constructing the result

Example revisited

`andtrees t1 t2`

$$\text{booltree } [k_1] \quad \times \quad \text{booltree } [k_2]\,,\ n \quad \to \quad \text{booltree } [k']\,,\ n'$$

$$|t1| \times k_1 \quad + \quad |t2| \times k_2 \ + n \quad \to \quad |r| \times k' \quad + n'$$

. . . which is really just . . .

$$n \geq n'$$

$$k_1 + k_2 \geq \mathrm{size}(\text{node}) + k'$$

Hofmann-Jost inference

$$\frac{n \geq \mathsf{size}(\mathsf{booltree\ node}) + k + n'}{l : \mathsf{booltree}[k], r : \mathsf{booltree}[k], v : \mathsf{bool}, n \vdash \mathsf{node}(l, v, r) : \mathsf{booltree}[k], n'} \ (\textsc{Node})$$

- Construct typing with constraint variables k, n, ...;
- collect constraints from typing rules;
- solve linear program, minimising the bound.

Hofmann-Jost inference

$$\frac{n \geq \mathsf{size}(\mathsf{booltree\ node}) + k + n'}{l : \mathsf{booltree}[k], r : \mathsf{booltree}[k], v : \mathsf{bool}, n \vdash \mathsf{node}(l, v, r) : \mathsf{booltree}[k], n'} \quad (\textsc{Node})$$

- Construct typing with constraint variables k, n, ...;
- collect constraints from typing rules;
- solve linear program, minimising the bound.

Note: can use the solutions to the linear program as a checkable *certificate* of memory requirements.

[Mobile Resource Guarantees project.]

Complexity of the inference

- Number of constraints is (roughly) linear in the program size;
- LP solving is polynomial time.

However, often want different uses of a function to have different signatures (*resource polymorphism*):

- Easy: Pretend they are different functions by collecting and duplicating all the constraints.
- Hard: Worst case now exponential (but contrived?)

Extensions — stack space

$$\frac{\begin{array}{c}\Sigma(f) = T_1, \ldots, T_p, k \to T', k' \\ n \geq k \qquad\qquad\qquad\qquad n - k + k' \geq n'\end{array}}{x : T_1, \ldots, x : T_p, n \vdash f(x_1, \ldots, x_p) : T', n'} \qquad (\textsc{Fun})$$

Extensions — stack space

'First adaption' attempt quite easy:

$$\frac{\Sigma(f) = T_1, \ldots, T_p, k \to T', k' \qquad n \geq k + \mathrm{frame}(f) \qquad n - k + k' \geq n'}{x\colon T_1, \ldots, x\colon T_p, n \vdash f(x_1, \ldots, x_p)\colon T', n'} \quad (\textsc{Fun})$$

- For tail call optimisation add tail position flag to judgements

Stack space problem 1

```
let andtrees2 x y z =
  let r1 = andtrees x y in
  let r2 = andtrees x z in
    (r1,r2)
```

- Overall stack usage is bounded by $|\mathtt{x}|$.
- But we only ‘pass the parcel’ to `r1`, so infer $2 \times |\mathtt{x}|$ instead.

Stack space solution 1

```
let andtrees2 x y z =
  let r1 = andtrees x y in
  let r2 = andtrees x z in
    (r1,r2)
```

- Allow *after evaluation* bounds to refer to arguments as well as results.
- Types now have two annotations: $\mathsf{booltree}[k \rightsquigarrow k']$ — we are given k units of space, but we should *give back* k'.

$$\texttt{andtrees} : \mathsf{booltree}[1 \rightsquigarrow 1] \times \mathsf{booltree}[0 \rightsquigarrow 0], 0 \to \mathsf{booltree}[0 \rightsquigarrow 0], 0$$

'Overlapping' potential

```
let id x = x
```

$$\texttt{id} : \mathsf{booltree}[1 \rightsquigarrow 1], 0 \to \mathsf{booltree}[1 \rightsquigarrow 1], 0$$

This looks OK, but what does it mean?

- Given one unit of space per node, we will have one unit per node of space w.r.t. the result; and
- we promise to give back that one unit per node for the result, and **then** we will have one unit per node of the argument.

Use a separation condition from the memory safety analysis to approximate data flow and spot where the **then** occurs.