

Amortised Resource Analysis for HUME

Kevin Hammond and Steffen Jost

ASTReNet Workshop, London, 21st March 2007

ASTReNet

Analysis, Slicing and Transformation
Research Network

The EmBounded Team

K.Hammond, R.Dyckhoff, S.Jost (St Andrews)

G.Michaelson, A.Wallace, R. Pointon, G.Grov (Heriot-Watt)

C.Ferdinand, R.Heckmann (AbsInt GmbH)

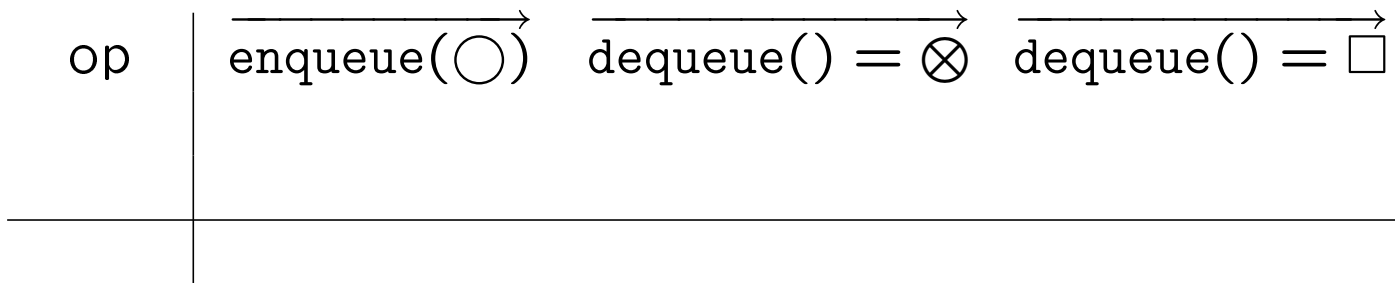
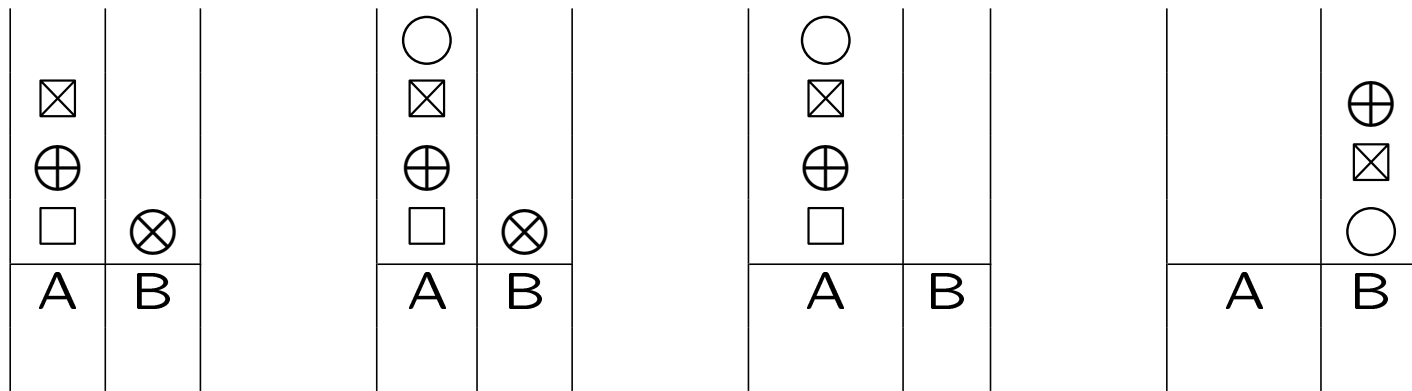
M.Hofmann, H-W.Loidl (LMU, München)

J.Sérot, N.Scaife (LASMEA, France)

Amortised Cost

Example: Simulating queue (FIFO) by two stacks (LIFO)

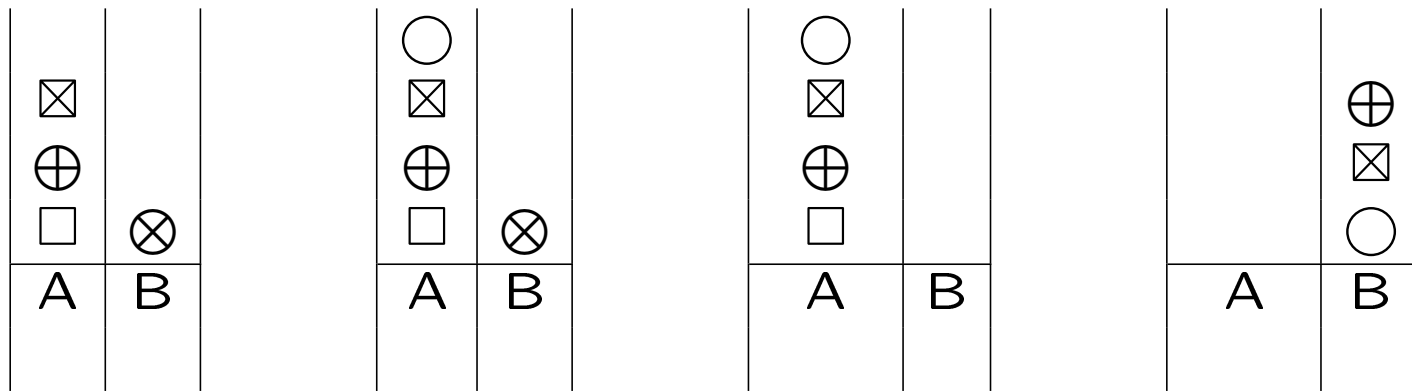
Always enqueue onto A and dequeue from B
 assume each basic push, pop or move costs 1



Amortised Cost

Example: Simulating queue (FIFO) by two stacks (LIFO)

Always enqueue onto A and dequeue from B
assume each basic push, pop or move costs 1

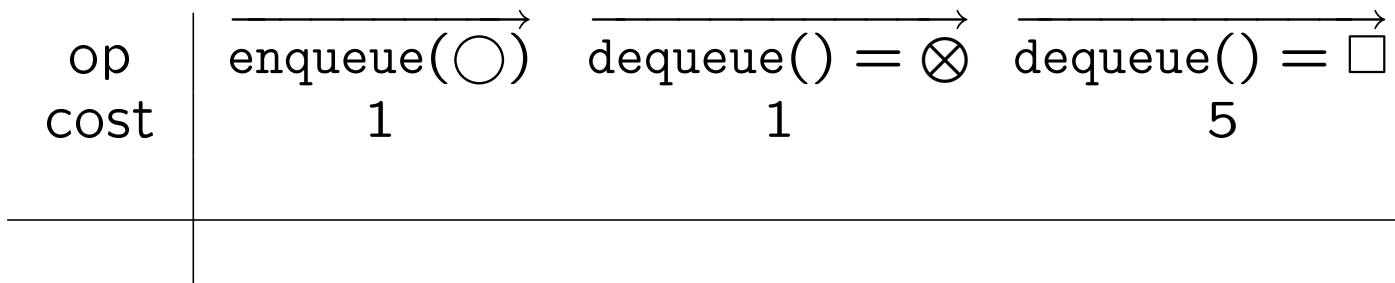
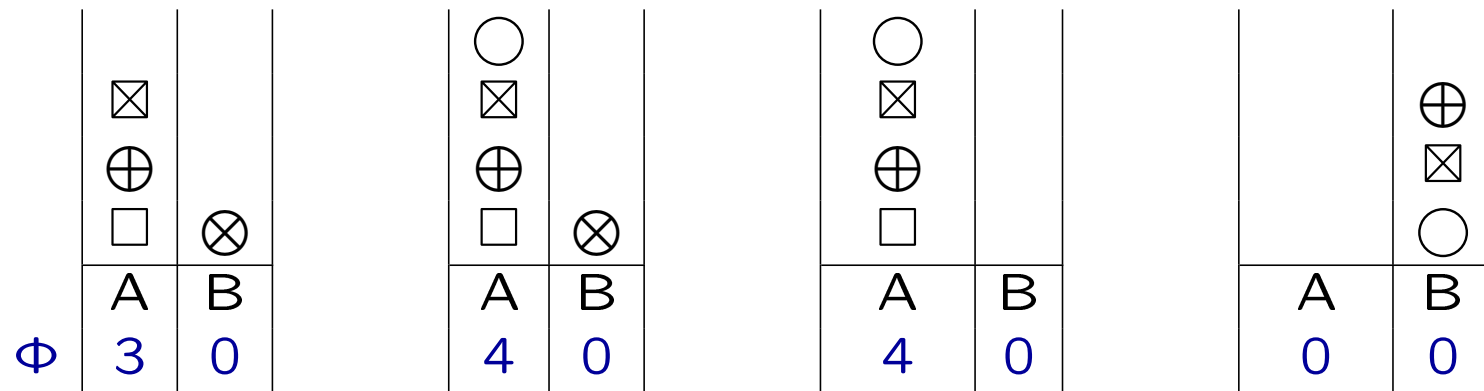


op cost	$\xrightarrow{\text{enqueue}(\bigcirc)}$	$\xrightarrow{\text{dequeue}() = \otimes}$	$\xrightarrow{\text{dequeue}() = \square}$
	1	1	5

Amortised Cost

Example: Simulating queue (FIFO) by two stacks (LIFO)

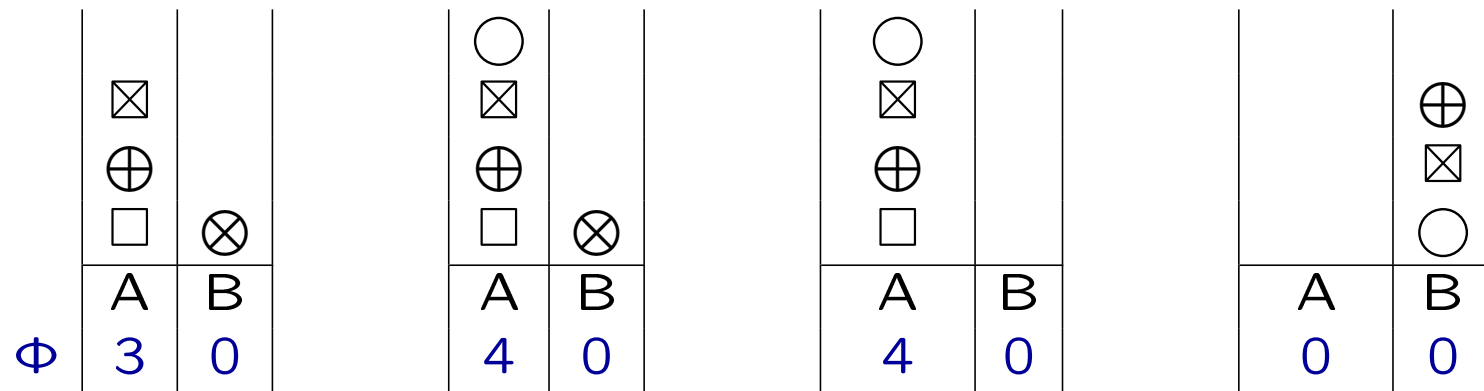
Always enqueue onto A and dequeue from B
assume each basic push, pop or move costs 1



Amortised Cost

Example: Simulating queue (FIFO) by two stacks (LIFO)

Always enqueue onto A and dequeue from B
assume each basic push, pop or move costs 1



op	→ enqueue(○)	→ dequeue() = ⊗	→ dequeue() = □
cost	1	1	5
ΔΦ	1	0	-4
Σ	2	1	1

HUME Types

$$B ::= \text{unit} \mid \text{int} \mid \text{float} \mid \text{char} \mid \text{bool} \mid \text{string}$$
$$A ::= B \mid C \mid X \mid \mu X. \{c_1:q_1; A_{(1,1)}, \dots, A_{(1,j_1)} \mid \dots \mid c_k:q_k; A_{(k,1)}, \dots, A_{(k,j_k)}\}$$
$$C ::= \forall \alpha \in \psi. A_1, \dots, A_k \xrightarrow[m']{m} A$$

- usual range of base types
- arbitrary annotated recursive datatypes
- function types with constraints and bound constraint vars

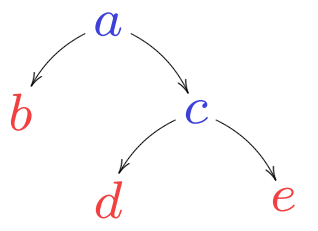
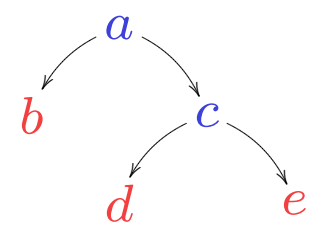
Automated Analysis of Functional Code: Idea

- Assign potential to data based on type
Type constructors receive weights $(\text{list}(\text{int}, 0), \text{list}(\text{int}, 1), \dots)$
- Abstract from actual values $(\text{list}(\text{int}, x), \text{list}(\text{int}, y), \dots)$
- Gather constraints from type derivation with amortised costs
- Feed constraints to LP solver

Successful heap-space analysis of first-order functional programs applied in EU FET-IST project [Mobile Resource Guarantees](#)

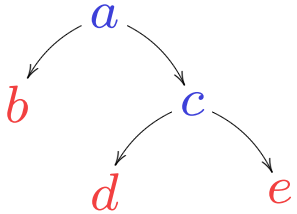
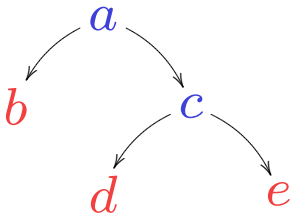
Calculation Example:

```
type tree= Leaf of char | Node of char*tree*tree
```

Examples of Enriched Type	Instance	Φ
<code>tree[char,0 char,#,#,3]</code>	 <pre> graph TD a((a)) --> b((b)) a --> c((c)) c --> d((d)) c --> e((e)) </pre>	$3 \cdot 0 + 2 \cdot 3 = 6$
<code>tree[char,4 char,#,#,1]</code>	 <pre> graph TD a((a)) --> b((b)) a --> c((c)) c --> d((d)) c --> e((e)) </pre>	$3 \cdot 4 + 2 \cdot 1 = 14$

Calculation Example:

```
type tree= Leaf of char | Node of char*tree*tree
```

Examples of Enriched Type	Instance	Φ
<code>tree[char,0 char,#,#,3]</code>	 <pre> graph TD a((a)) --> b((b)) a --> c((c)) c --> d((d)) c --> e((e)) style a fill:#add8e6 style b fill:#ff0000 style c fill:#add8e6 style d fill:#ff0000 style e fill:#ff0000 </pre>	$3 \cdot 0 + 2 \cdot 3 = 6$
<code>tree[char,4 char,#,#,1]</code>	 <pre> graph TD a((a)) --> b((b)) a --> c((c)) c --> d((d)) c --> e((e)) style a fill:#add8e6 style b fill:#ff0000 style c fill:#add8e6 style d fill:#ff0000 style e fill:#ff0000 </pre>	$3 \cdot 4 + 2 \cdot 1 = 14$

- Annotations are linearly distributed in aliased data

Automated Analysis: Result

$$f : \text{list}(\text{list}(\text{int}, 1), 2.3) \xrightarrow{\frac{4}{6}} \text{list}(\text{int}, 5)$$

Evaluating $f([l_1, \dots, l_m])$

- requires at most $4 + 2.3m + 1\sum|l_i|$ extra heap units and
- leaves at least $6 + 5|f(l)|$ unused memory units

The **potential** of the consumed input gives an upper bound on the overall runtime heap-consumption

Annotations are weight factors – *not* references to length/size as *opposed* to sized types [Hughes & Pareto '99,'02]

Informal calculation:

$\text{rev_aux} : \text{list}(\text{int }), \text{list}(\text{int }) \longrightarrow \text{list}(\text{int })$

$$\left. \begin{array}{l} \text{rev_aux}([], acc) = acc \\ \text{rev_aux}(h :: t, acc) = \text{rev_aux}(t, h :: acc) \end{array} \right|$$

- In ASCII we write $A, 3 \rightarrow B, 4$ for type $A \xrightarrow{3} B$

Informal calculation:

$\text{rev_aux} : \text{list}(\text{int}, u), \text{list}(\text{int}, v), x \longrightarrow \text{list}(\text{int}, w), y$

$$\begin{array}{l} \text{rev_aux}([], acc) = acc \\ \text{rev_aux}(h :: t, acc) = \text{rev_aux}(t, h :: acc) \end{array} \left| \begin{array}{l} \{x \geq y, v \geq w\} \\ \{x + u \geq k + v + p, \\ p \geq x, p \geq x - y + y\} \end{array} \right.$$

- In ASCII we write $A, 3 \rightarrow B, 4$ for type $A \xrightarrow{3} B$
- Enrich type system with resource variables

Informal calculation:

$\text{rev_aux} : \text{list}(\text{int}, u), \text{list}(\text{int}, v), x \longrightarrow \text{list}(\text{int}, w), y$

$$\begin{array}{l} \text{rev_aux}([], acc) = acc \\ \text{rev_aux}(h :: t, acc) = \text{rev_aux}(t, h :: acc) \end{array} \left| \begin{array}{l} \{x \geq y, v \geq w\} \\ \{x + u - k - v \geq x\} \end{array} \right.$$

- In ASCII we write $A, 3 \rightarrow B, 4$ for type $A \xrightarrow{\frac{3}{4}} B$
- Enrich type system with resource variables
- Gather linear constraints

Informal calculation:

$\text{rev_aux} : \text{list}(\text{int}, u), \text{list}(\text{int}, v), x \longrightarrow \text{list}(\text{int}, w), y$

$$\begin{array}{l} \text{rev_aux}([], acc) = acc \\ \text{rev_aux}(h :: t, acc) = \text{rev_aux}(t, h :: acc) \end{array} \left| \begin{array}{l} \{x \geq y, v \geq w\} \\ \{x + u - k - v \geq x\} \end{array} \right.$$

$$x = 0, y = 0, u = 1, v = 0, w = 0; \text{ for } k = 1$$

- In ASCII we write $A, 3 \rightarrow B, 4$ for type $A \xrightarrow{\frac{3}{4}} B$
- Enrich type system with resource variables
- Gather linear constraints
- Solve constraints by standard methods

Informal calculation:

$\text{rev_aux} : \text{list}(\text{int}, 1), \text{list}(\text{int}, 0), 0 \longrightarrow \text{list}(\text{int}, 0), 0$

$$\begin{array}{l} \text{rev_aux}([], acc) = acc \\ \text{rev_aux}(h :: t, acc) = \text{rev_aux}(t, h :: acc) \end{array} \left| \begin{array}{l} \{0 \geq 0, 0 \geq 0\} \\ \{0 + 1 - 1 \geq 0, \\ 0 + 1 - 1 \geq 0 - 0 + 0\} \end{array} \right.$$

$x = 0, y = 0, u = 1, v = 0, w = 0; \text{ for } k = 1$

- In ASCII we write $A, 3 \rightarrow B, 4$ for type $A \xrightarrow{\frac{3}{4}} B$
- Enrich type system with resource variables
- Gather linear constraints
- Solve constraints by standard methods

Informal calculation:

$\text{rev_aux} : \text{list}(\text{int}, u), \text{list}(\text{int}, v), x \longrightarrow \text{list}(\text{int}, w), y$

$$\begin{array}{l} \text{rev_aux}([], acc) = acc \\ \text{rev_aux}(h :: t, acc) = \text{rev_aux}(t, h :: acc) \end{array} \left| \begin{array}{l} \{x \geq y, v \geq w\} \\ \{u \geq k + w\} \end{array} \right.$$

$x = 0, y = 0, u = 1, v = 0, w = 0; \text{ for } k = 1$

- In ASCII we write $A, 3 \rightarrow B, 4$ for type $A \xrightarrow{\frac{3}{4}} B$
- Enrich type system with resource variables
- Gather linear constraints
- OR: project polyhedron to relevant dimensions

Key Features of HUME

- Purely functional expression language
- (Recursive, Higher-Order, Automatic memory management)
- Asynchronous rules embedded into “boxes”
- Boxes wired into static process network
- Single-buffering on wires
- Heap/Stack deallocation on box exit
- Explicit handling of time
- Exception handling

Example: Accumulating Parameter Factorial

```
fac_acc :: int -> int -> int;
fac_acc n a =
  case n of
    0 -> a
  | x -> fac_acc (x - 1) (x * a)
;
```

Example: Box Version of Factorial

```
box fac_box
in  (n :: int, ri :: int, ai :: int)
out (f :: int, ro :: int, ao :: int)
match
  ( i, *, *) -> ( *, i      , 1      )
| ( *, 0, a) -> ( a, *      , *      )
| ( *, r, b) -> ( *, r - 1, r * b)
;

wire stdin      to fac_box.n;
wire fac_box.ro to fac_box.ri;
wire fac_box.ao to fac_box.ai;
wire fac_box.f  to stdout;
```

Analysing HUME

Required extensions over previous work

- Higher-order types
- Polymorphism
- Arbitrary recursive datatypes
- Bounding non-terminating computation
- Bound arbitrary resource kinds
(Heap-space, Stack-space, Worst-Case Execution Time, . . .)

carried out within EU FET-IST project [EmBounded](#)

but only *a part* of EmBounded

Issues Encountered

Higher-order

Determining closure sizes in presence of under application

Solution: Incremental closures

Resource Polymorphism

Resource annotations must be polymorphic

Solution: Constraint sets become part of types

Exceptions and loss of let-normal form

Causes many cumbersome redundancies

Solution: Alternative syntax to let having different cost

$$\text{fid } e_1 \dots e_n \quad \rightsquigarrow \quad \text{LET } x_1 = e_1 \text{ in LET } \dots \text{ in fid } x_1 \dots x_n$$

Core-HUME

Programs automatically translated to Core-HUME for analysis

- Let-normal form via “Ghost”-Let
 - much simpler type rules
- All variables become unique
- Box-wires transformed into datatypes (Bundles & Wires)
 - boxes can be treated like functions
- Over-applications translated into successive applications
 - no need for extra mechanism
- Resource cost behaviour is preserved

Reading Rules

$$\overline{\mathcal{V}, \eta \mid \frac{p'+1}{p'} \mid \frac{m}{m-2}} e \rightsquigarrow \ell, \eta'$$

is an abbreviation for

$$\frac{\begin{array}{l} p = p' + 1 \\ m = m' + 2 \end{array} \quad \begin{array}{l} p \in \mathbb{N} \\ m \in \mathbb{N} \end{array} \quad \begin{array}{l} p' \in \mathbb{N} \\ m' \in \mathbb{N} \end{array}}{\mathcal{V}, \eta \mid \frac{p}{p'} \mid \frac{m}{m'}} e \rightsquigarrow \ell, \eta'$$

Annotated Type Rule

$$\frac{\Gamma \frac{p}{p'} \frac{m}{m'} e_t : A \mid \Phi \quad \Gamma \frac{p}{p'} \frac{m}{m'} e_f : A \mid \Psi}{\Gamma, x:\text{bool} \frac{p-1}{p'} \frac{m}{m'} \text{if } x \text{ then } e_t \text{ else } e_f : A \mid \Phi \cup \Psi} \text{ (Conditional)}$$

Annotated Operational Semantics

$$\frac{\eta(\mathcal{V}(x)) = (\text{bool}, \mathbb{t}) \quad \mathcal{V}, \eta \frac{p+1}{p'} \frac{m}{m'} e_1 \rightsquigarrow l', \eta'}{\mathcal{V}, \eta \frac{p}{p'} \frac{m}{m'} \text{if } x \text{ then } e_1 \text{ else } e_2 \rightsquigarrow l', \eta'} \text{ (Conditional True)}$$

$$\frac{\eta(\mathcal{V}(x)) = (\text{bool}, \mathbb{f}) \quad \mathcal{V}, \eta \frac{p+1}{p'} \frac{m}{m'} e_2 \rightsquigarrow l', \eta'}{\mathcal{V}, \eta \frac{p}{p'} \frac{m}{m'} \text{if } x \text{ then } e_1 \text{ else } e_2 \rightsquigarrow l', \eta'} \text{ (Conditional False)}$$

Building an Inference Algorithm

$$\frac{\Gamma \frac{p}{p'} \frac{m}{m'} e_t : A \mid \Phi \quad \Gamma \frac{p}{p'} \frac{m}{m'} e_f : A \mid \Psi}{\text{(Conditional)}}$$

$$\Gamma, x:\text{bool} \frac{p-1}{p'} \frac{m}{m'} \text{ if } x \text{ then } e_t \text{ else } e_f : A \mid \Phi \cup \Psi$$

- Generalise over resource kinds (polymorphism)
- Merge substructural rules (Weakening, Relax)
- Insert resource variables and constants in each step

$$\frac{\Gamma \frac{q_1}{q'_1} e_t : A \mid \Phi \quad \Gamma \frac{q_2}{q'_2} e_f : A \mid \Psi}{\text{(Conditional*)}}$$

$$\Gamma, x:\text{bool} \frac{q_0}{q'_0} \text{ if } x \text{ then } e_t \text{ else } e_f : A \mid \Phi \cup \Psi \cup \left\{ \begin{array}{l} q_0 \geq q_1 + k_{tb}, \quad q'_1 + k_{ta} \geq q'_0, \\ q_0 \geq q_2 + k_{fb}, \quad q'_2 + k_{fa} \geq q'_0 \end{array} \right\}$$

Application

$$\frac{D = \forall \alpha \in \psi. A_1, \dots, A_a \xrightarrow[p', m']{p, m} C \quad k \geq 1 \quad k = a}{z:D, y_1:A_1, \dots, y_k:A_k \mid \frac{p + k_{frame}}{p' + k_{frame} + k} \mid \frac{m}{m'} \quad z \ y_1 \cdots y_k : C \mid \psi} \text{ (App Var)}$$

Under Application

$$\begin{array}{c}
 D = \forall \alpha \in \psi. A_1, \dots, A_a \xrightarrow[p' m']{p m} C \quad k \geq 1 \quad k < a \\
 \forall i \leq k. (\phi_i = \forall (A_i | A_i, A_i)) \quad \beta = \alpha \setminus \mathbf{FV}(A_1, \dots, A_k) \\
 B = \forall \beta \in \psi. A_{k+1}, \dots, A_a \xrightarrow[p' + k m']{p + k m} C \\
 \hline
 z:D, y_1:A_1, \dots, y_k:A_k \mid \frac{0}{k-1} \mid \frac{k_{\text{clos}} + k}{0} \quad z \ y_1 \cdots y_k : B \mid \bigcup_i \phi_i \\
 \text{(Under App Var)}
 \end{array}$$

Constructor and Case

Binding of potential:

$$\begin{array}{c}
 c \in \text{Con} \quad C = \mu X. \{ \dots \mid c : q; B_1, \dots, B_k \mid \dots \} \quad k \geq 1 \\
 A_i = B_i \vee (A_i = C \wedge B_i = X) \quad (\text{for } i = 1, \dots, k) \\
 \hline
 x_1 : A_1, \dots, x_k : A_k \quad \left| \frac{q}{k-1} \right| \frac{q + \text{Size}(c)}{0} \quad c \ x_1 \dots x_k : C \mid \emptyset
 \end{array} \quad (\text{Constr})$$

Constructor and Case (contd.)

Release of potential:

$$q = \text{maxVars}(pat_1, \dots, pat_k)$$

$$\forall i. \left\{ \begin{array}{l} A \mid \frac{p_{(i-1)} \mid m_{(i-1)}}{p_i \mid m_i} \quad pat_i \ ?\triangleright \ \Delta_i; \pi_i; \psi_i \\ \Gamma, \Delta_i \mid \frac{p'_i \mid m'_i}{p'' \mid m''} \quad e_i : B \mid \phi_i \\ \chi_i = \{\pi_i + p_i \geq p'_i\} \{\pi_i + m_i \geq m'_i\} \end{array} \right. \quad \text{(Case)}$$

$$\Gamma, x:A$$

$$\mid \frac{p_0 + k_{frame} + q \mid m_0}{p'' + k_{frame} + 1 + q \mid m''}$$

$$\text{case } x \text{ of } pat_1 \rightarrow e_1 \mid \dots \mid pat_k \rightarrow e_k : B \mid \cup_i \psi_i \cup \phi_i \cup \chi_i$$

Substructural Rules

$$\frac{\Gamma \frac{q}{q'} \frac{n}{n'} e : A \mid \psi}{\Gamma \frac{p}{p'} \frac{m}{m'} e : A \mid \psi \cup \{p \geq q, p - q \geq p' - q'\}}$$

(Relax Stack)

$$\frac{\Gamma \frac{p}{p'} \frac{m}{m'} e : C \mid \psi}{\Gamma, x:A \frac{p}{p'} \frac{m}{m'} e : C \mid \psi}$$

(Weak)

$$\frac{\Gamma, x:B \frac{p}{p'} \frac{m}{m'} e : C \mid \psi}{\Gamma, x:A \frac{p}{p'} \frac{m}{m'} e : C \mid \psi \cup A <: B}$$

(Supertype)

$$\frac{\Gamma \frac{p}{p'} \frac{m}{m'} e : C \mid \psi}{\Gamma \frac{p}{p'} \frac{m}{m'} e : D \mid \psi \cup C <: D}$$

(Subtype)

$$\frac{\Gamma, x:A_1, y:A_2 \frac{p}{p'} \frac{m}{m'} e : C \mid \phi}{\Gamma, z:A \frac{p}{p'} \frac{m}{m'} e[z/x, z/y] : C \mid \phi \cup \forall(A \mid A_1, A_2)}$$

(Share)

where $C <: D := \exists E . \forall(C \mid D, E)$

Main Soundness Theorem

Theorem 1 (Soundness). Choose a well-typed HUME program. Let $r \in \mathbb{Q}^+$ be fixed, but arbitrary. If the following four statements hold

$$\Gamma \mid_{\frac{q}{q'}} e:A \mid \phi \quad (1.A)$$

$$\mathcal{V}, \eta \vdash e \rightsquigarrow \ell, \eta' \quad (1.B)$$

$$\eta \models \mathcal{V} : \Gamma \quad (1.C)$$

$$v : CV \rightarrow \mathbb{Q}^+, \text{ satisfying } \phi \quad (1.D)$$

then for all $m \in \mathbb{N}$ such that

$$m \geq v(q) + \Phi_{\eta}(\mathcal{V} : v(\Gamma)) + r \quad (1.E)$$

there exists $m' \in \mathbb{N}$ satisfying

$$\mathcal{V}, \eta \mid_{\frac{m}{m'}} e \rightsquigarrow \ell, \eta' \quad (1.I)$$

$$m' \geq v(q') + \phi_{\eta'}(\ell : v(A)) + r \quad (1.II)$$

$$\eta' \models \ell : A \quad (1.III)$$

Outline Proof

Proof. The proof is by induction on the lengths of the derivations of (1.B) and (1.A) ordered lexicographically with the derivation of the evaluation taking priority over the typing derivation.

The ordering is required since an induction on the length of the typing derivation alone would fail for the case of function application, which increases the length of the typing derivation.

While the length of the derivation for the evaluation of terms never increases, it may remain unchanged in the cases where the last step of the typing derivation was obtained by a substructural rule. In such cases the length of the typing derivation does decrease, allowing an induction over lexicographically ordered lengths of both derivations. \square

Example 1: factorial function over floats

```
program
```

```
type _float = float 32;
```

```
fac n = if (n==0.0)  
        then 1.0  
        else n * (fac (n - 1.0));
```

```
expression (fac);
```

Example 1: factorial function over floats

In the first stage of the analysis this code is translated into an intermediate format:

```
program

-- type of main
val main :: float, ->float
-- Functions
{fac :: float, ->float (n :: float) =
  glet ?z_1    = 0.0
  in glet ?z_2    = n==?z_1
    in if ?z_2
      then 1.0
      else glet ?z_3    = 1.0
        in glet ?z_4    = n-?.?z_3
          in glet ?z_5    = (fac <> ?z_4)
            in n*?.?z_5}

-- Boxes
-- Expression:
(fac)
```

Analytical Results: Heap Consumption

ARTHUR3 typing for resource "Heap":

```
0, (float<10>) -6/0-> float<0> ,0
```

For some input n , execution of the main function will require $10n + 6$ heap units.

An interval analysis is used to deduce ranges for the possible values of the floating-point variable. This generates linear (in-)equality constraints, which are then solved by a separate LP-solver.

Analytical Results: Stack Consumption

ARTHUR3 typing for resource "Stack":

```
1, (float<3>) -1/0-> float<1> ,0
```

For some input n , execution of the main function will require $3n + 1$ stack units, plus 1 stack unit to set-up the main expression.

Analytical Results: Worst-Case Execution Time

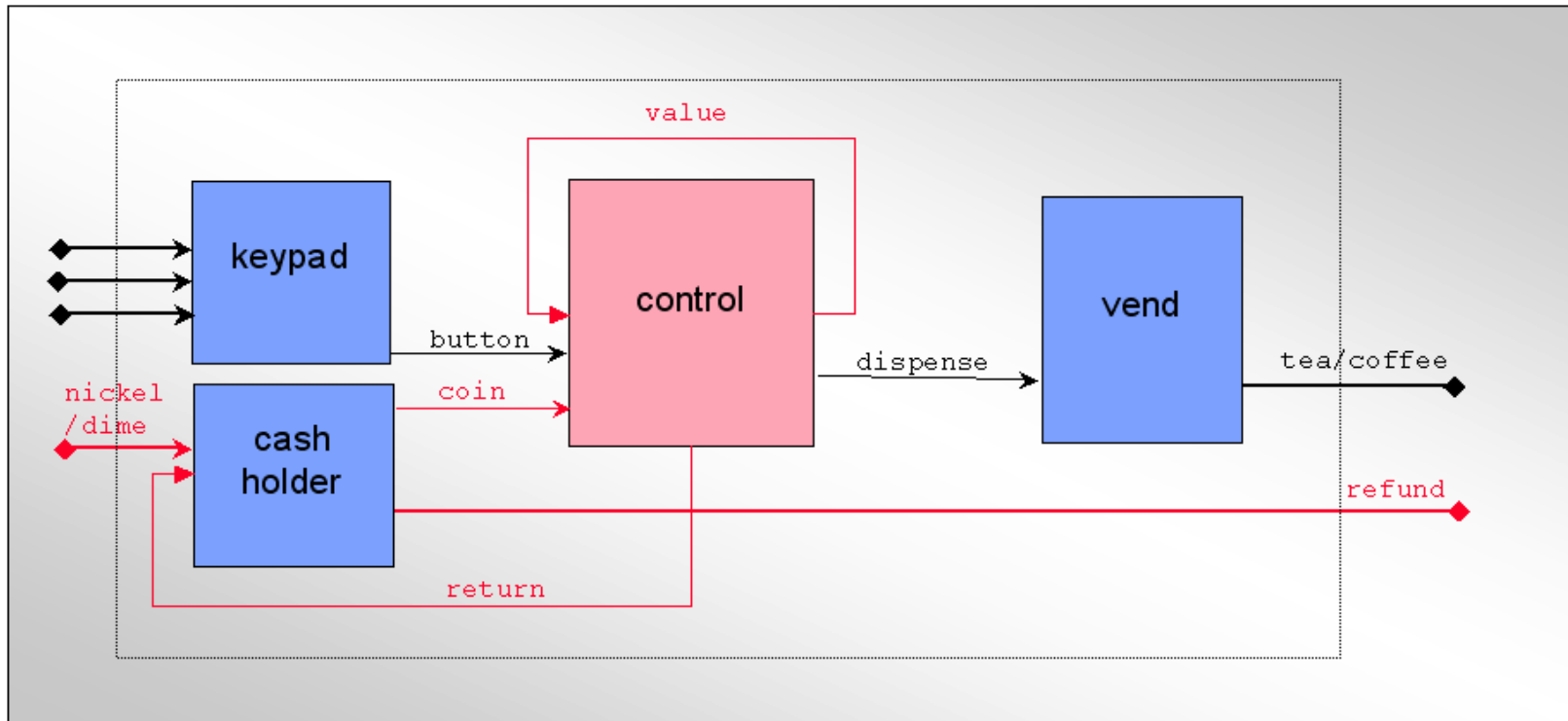
Worst-case timing figures have been obtained for a Renesas M32C/85 based on costs of individual AM instructions obtained using AbsInt's **aiT** tool. These are reflected in the operational semantics, and then the corresponding type rules.

ARTHUR3 typing for resource "Time":

```
30, (float<1043>) -501/0-> float<59> ,0
```

For some input n , execution of the main function will require $1043n + 501$ machine cycles, plus 30 cycles set-up time.

Example 2: drinks vending machine



Analytical Results: Heap Usage

ARTHUR3 typing for resource "Heap":

Box: inp

?v_6: wire1char[17;char|*]

---2/0--->

?v_2: wire1coins[8;coins[0|0]|*], ?v_1: wire1buttons[16;buttons[0|0|0]|*]

Box: coffee

?v_2: wire1coins[8;coins[0|0]|*], ?v_1: wire1buttons[16;buttons[0|0|0]|*],

?v_5: wire1float[0;float<0>|*]

---0/0--->

?v_3: wire1drinks[5;drinks[0|0]|*], ?v_5: wire1float[0;float<0>|*],

?v_4: wire1float[9;float<0>|*]

Box: outp

?v_3: wire1drinks[5;drinks[0|0]|*], ?v_4: wire1float[9;float<0>|*]

---0/0--->

?v_7: wire1int[0;int|*], ?v_8: wire1float[6;float<0>|*]

Analytical Results: Heap Usage (contd.)

We observe that all boxes can operate in constant heap space, since all potentials attached to the data structures used here have a factor of 0.

In total 512 inequalities in 398 variables were generated; solving them takes about 0.16s on a 1.7GHz Pentium laptop, with 1GB main memory and 2MB cache.

Analytical Results: Stack Usage

ARTHUR3 typing for resource "Stack":

Box: inp

?v_6: wire1char[2;char|*]

---2/1--->

?v_2: wire1coins[0;coins[0|0]|*], ?v_1: wire1buttons[2;buttons[0|0|0]|*]

Box: coffee

?v_2: wire1coins[0;coins[0|0]|*], ?v_1: wire1buttons[2;buttons[0|0|0]|*],

?v_5: wire1float[0;float<0>|*]

---4/2--->

?v_3: wire1drinks[2;drinks[0|0]|*], ?v_5: wire1float[0;float<0>|*],

?v_4: wire1float[4;float<0>|*]

Box: outp

?v_3: wire1drinks[2;drinks[0|0]|*], ?v_4: wire1float[4;float<0>|*]

---3/1--->

?v_7: wire1int[4;int|*], ?v_8: wire1float[6;float<0>|*]



Analytical Results: Time Usage

ARTHUR3 typing for resource "Time":

Box: inp

?v_6: wire1char[1809;char|*]

---2603/0--->

?v_2: wire1coins[0;coins[0|0]|*], ?v_1: wire1buttons[1741;buttons[0|402|0]|*]

Box: coffee

?v_2: wire1coins[0;coins[0|0]|*], ?v_1: wire1buttons[1741;buttons[0|402|0]|*],

?v_5: wire1float[0;float<0>|*]

---4790/0--->

?v_3: wire1drinks[1104;drinks[0|207]|*], ?v_5: wire1float[0;float<0>|*],

?v_4: wire1float[626;float<0>|*]

Box: outp

?v_3: wire1drinks[1104;drinks[0|207]|*], ?v_4: wire1float[626;float<0>|*]

---1180/0--->

?v_7: wire1int[687;int|*], ?v_8: wire1float[0;float<0>|*]

Conclusions

- Amortised analysis for HUME expressions/boxes generalises previous work.
- Deals with higher-orderness, general datatypes, generalises to heap, stack **and** time.
- Correlated against implementation of HUME– formal operational semantics.
- Results verified against test implementation.
- Soundness proof under construction, but no major problems encountered so far.

Current Work

- Completing soundness proof
- Studying larger testbed examples
 - Lasmae, France; Heriot-Watt, Edinburgh
- Experimenting with annotated base-types
 - benefits from Sized-Type analysis
- Determining usage factors for closures to allow capture of potential
- Costs of non-strict evaluation
- Abstract interpretation using *dependent types*



Project documents and online Amortised Analysis at
<http://www.embounded.org>

HUME Abstract Syntax

<i>program</i>	$::=$	$decl_1 ; \dots decl_n ;$	$n \geq 1$
<i>decl</i>	$::=$	$box \mid id = expr \mid id \langle match_1 \mid \dots \mid match_n \rangle$	$n \geq 1$
<i>box</i>	$::=$	$box \ id \ (ins) \ (outs) \ fairness \ bmatches \ [handle \ cmatches]$	
<i>ins,outs</i>	$::=$	$\langle id_1, \dots, id_n \rangle$	$n \geq 0$
<i>fairness</i>	$::=$	$fair \mid unfair$	
<i>bmatches</i>	$::=$	$\langle bmatch_1 \mid \dots \mid bmatch_n \rangle$	$n \geq 1$
<i>bmatch</i>	$::=$	$\langle bpat_1, \dots, bpat_n \rangle \rightarrow expr$	$n \geq 1$
<i>cmatches</i>	$::=$	$\langle cmatch_1 \mid \dots \mid cmatch_n \rangle$	$n \geq 1$
<i>cmatch</i>	$::=$	$exn \ pat \rightarrow exnexpr$	
<i>bpat</i>	$::=$	$pat \mid * \mid _*$	
<i>pat</i>	$::=$	$int \mid float \mid char \mid bool \mid string \mid _ \mid var$ $\mid con \ pat_1 \ \dots \ pat_n$ $\mid (\ pat_1 \ , \ \dots \ , \ pat_n \)$	$n \geq 0$ $n \geq 2$

HUME Abstract Syntax (cont'd)

```

expr ::= int | float | char | bool | string | *
        | var expr1 ... exprn           n > 0
        | id expr1 ... exprn           n ≥ 0
        | con expr1 ... exprn         n ≥ 0
        | ( expr1 , ... , exprn )     n ≥ 2
        | if expr1 then expr2 else expr3
        | case expr of < match1 | ... | matchn > n ≥ 1
        | let < vdecl1 , ... , vdecln > in expr n ≥ 1
        | expr within int time raise expr
        | expr within int stack raise expr
        | expr within int heap raise expr
        | raise expr

match ::= pat -> expr
vdecl ::= var = expr

```

Ghost-Let

$$\begin{array}{c}
 \forall i. \left\{ \begin{array}{l}
 \Delta_i = \{x_1:A_1^i, \dots, x_{(i-1)}:A_{(i-1)}^i\} \upharpoonright \text{FV}(e_i) \\
 \mathcal{A}_i = \bigsqcup_j \text{ran}(\Delta_j \upharpoonright \{x_i\}) \\
 \psi_i = \Downarrow(A_i | \mathcal{A}_i) \\
 \Delta_i, \Gamma_i \mid \frac{p_{(i-1)}}{p_i} \mid \frac{m_{(i-1)}}{m_i} \quad e_i : A_i \mid \phi_i
 \end{array} \right. \\
 \hline
 \Gamma_1, \dots, \Gamma_{k+1} \mid \frac{p_0}{p_{k+1}} \mid \frac{m_0}{m_{k+1}} \quad \text{LET } x_1 = e_1, \dots, x_k = e_k \text{ IN } e : A \mid \bigcup_i \phi_1 \cup \psi_i \\
 \text{(Ghost let)}
 \end{array}$$