Story Outline

Programs often produce unwanted *"emissions"*, such as littering the memory with garbage. We a priori limit the amount of "emissions" a program can produce by issuing "carbon credits", which are paired with a program's input at compile time.

As opposed to the real world, our "bureaucracy" imposes zero overhead at runtime and we can rigorously prove that each and every "emission" is amortised by expending a "carbon credit".

"Carbon credits" are linearly paired with a program's input. An environmental-friendly program will require less "carbon credits" than an inefficient, polluting one.

Abstract

Bounding resource usage is important for a number of areas, notably real-time embedded systems and safety-critical systems. In this paper, we present a fully automatic static type-based analysis for inferring upper bounds on resource usage for programs involving general algebraic datatypes and full recursion. Our method can easily be used to bound any countable resource, without needing to revisit proofs.

We apply the analysis to the important metrics of worst-case execution time, stack- and heap-space usage. Our results from several realistic embedded control applications demonstrate good matches between our inferred bounds and measured worst-case costs for heap and stack usage. For time usage we infer good bounds for one application. Where we obtain less tight bounds, this is due to the use of software floating-point libraries.

"Carbon credits" for Resource-Bounded Computations using Amortised Analysis

Steffen Jost – St Andrews University, UK
Hans-Wolfgang Loidl – Heriot-Watt University, UK
Kevin Hammond – St Andrews University, UK
Norman Scaife – Université Blaise-Pascal, France
Martin Hofmann – Ludwig-Maximilians University, Germany

Eindhoven, the Netherlands

November 5, 2009

Program Analysis

Automatic	No programmer annotations required
Static	No change of runtime behaviour
Efficient	Programmer can analyse on the fly
Modular	Analyse libraries only once
Transparent	Programmer can understand derivation of bounds
Reliable	Delivers formally guaranteed upper-bound functions
Generic	Bounding usage of heap, stack, time, calls, ...

Good Mutual recursive (higher-order) functional language
Implementation at `http://www.embounded.org`

Bad Only linear bounds inferred ... thus far!

Program Analysis

Automatic	No programmer annotations required
Static	No change of runtime behaviour
Efficient	Programmer can analyse on the fly
Modular	Analyse libraries only once
Transparent	Programmer can understand derivation of bounds
Reliable	Delivers formally guaranteed upper-bound functions
Generic	Bounding usage of heap, stack, time, calls, ...

Good	Mutual recursive (higher-order) functional language Implementation at `http://www.embounded.org`
Bad	Only linear bounds inferred ... thus far!

Program Analysis

Automatic	No programmer annotations required
Static	No change of runtime behaviour
Efficient	Programmer can analyse on the fly
Modular	Analyse libraries only once
Transparent	Programmer can understand derivation of bounds
Reliable	Delivers formally guaranteed upper-bound functions
Generic	Bounding usage of heap, stack, time, calls, ...

Good Mutual recursive (higher-order) functional language
Implementation at `http://www.embounded.org`

Bad Only linear bounds inferred ... thus far!

Idea Programs become "emission free"
by amortising emissions with "carbon credits"
received bundled with their input (Mindgame!)

Gain Pollution bounded up-front
by number of carbon credits issued

How much carbon credits do we need?

Automatic Amortised Analysis:

Simple Small addition to standard type system

Guaranteed Predicted credits proven to suffice

No Bureaucracy Static type based analysis

Idea Programs become "emission free"
by amortising emissions with "carbon credits"
received bundled with their input (Mindgame!)

Gain Pollution bounded up-front
by number of carbon credits issued

How much carbon credits do we need?

Automatic Amortised Analysis:

Simple Small addition to standard type system

Guaranteed Predicted credits proven to suffice

No Bureaucracy Static type based analysis

Idea Programs become "emission free"
by amortising emissions with "carbon credits"
received bundled with their input (Mindgame!)

Gain Pollution bounded up-front
by number of carbon credits issued

How much carbon credits do we need?

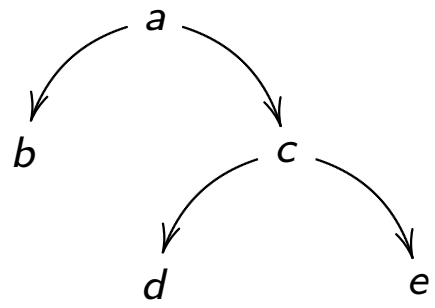
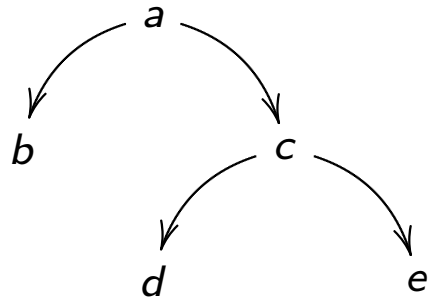
Automatic Amortised Analysis:

Simple Small addition to standard type system

Guaranteed Predicted credits proven to suffice

No Bureaucracy Static type based analysis

Tree Processing Function

Analysis' Result	Input Instance	Carbon Credits Φ
$\mu X.\{\ \text{Leaf} : (3.5; \text{char})$ $\mid \text{Node} : (7; \text{char}, X, X)\}$		$3 \cdot 3.5 + 2 \cdot 7 \ = 24.5$
$\mu X.\{\ \text{Leaf} : (6; \text{char})$ $\mid \text{Node} : (0; \text{char}, X, X)\}$		$3 \cdot 6 + 2 \cdot 0 \ = 18$

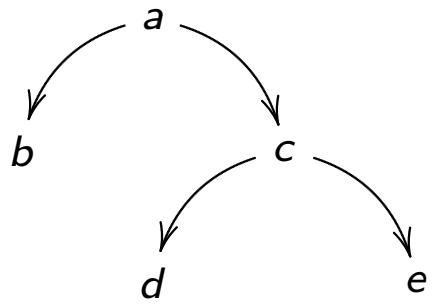
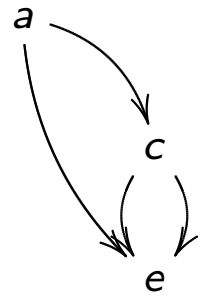
- Each constructor carries its own credits

  $\rightsquigarrow$ Upper bounds presented as linear functions of input sizes

Annotations are weight factors with *no* reference to size/length

as *opposed* to Sized Types [Hughes & Pareto '99,'02]

Tree Processing Function

Analysis' Result	Input Instance	Carbon Credits Φ
$\mu X.\{\ \text{Leaf} : (3.5; \text{char})$ $\mid \text{Node} : (7; \text{char}, X, X)\}$		$3 \cdot 3.5 + 2 \cdot 7 \ = 24.5$
$\mu X.\{\ \text{Leaf} : (6; \text{char})$ $\mid \text{Node} : (0; \text{char}, X, X)\}$		$3 \cdot 6 + 2 \cdot 0 \ = 18$

- Each constructor carries its own credits
  $\rightsquigarrow$ Upper bounds presented as linear functions of input sizes
- Credit is linearly distributed in aliased data
  $\rightsquigarrow$ Consumer needs not care about possible aliasing

Annotations are weight factors with *no* reference to size/length

as *opposed* to Sized Types [Hughes & Pareto '99,'02]

Dealing with Aliasing

- Explicit contraction rule — Pointer duplication
- Explicit weakening rule — Pointer discarding

$x : \mathsf{list}(3; \mathsf{list}(4; \mathsf{list}(5; \mathsf{int})))$ may share to $x_1 : \mathsf{list}(0; \mathsf{list}(2; \mathsf{list}(1; \mathsf{int})))$, $x_2 : \mathsf{list}(3; \mathsf{list}(2; \mathsf{list}(4; \mathsf{int})))$

Contraction rule shares carbon credits linearly between aliases
$\rightsquigarrow$ safe deallocations are accounted for

[Bunched Implication Logic, Ishtiaq & O'Hearn '01]

[Alias Types, Walker & Morrisett '01]

We employ *Storeless Semantic*, capturing quantities and aliasing, but excluding random reanimation of stale pointers

[Jonkers '81; Rinetzky, Wilhelm et al. '05]

Underlying principle

Based on "Amortised Analysis" R.E.Tarjan'85

- Abstract state into *single number*: $\Phi(\text{state})$
- Ensure for transitions: $\mathit{actual\ cost} - \Phi(\mathit{before}) + \Phi(\mathit{after}) = 0$
- Overall cost easy to determine: $\Phi(\mathit{initial\ state})$

Problems:

- Ingenuity required for application!

Our solution:

Underlying principle

Based on "Amortised Analysis" R.E.Tarjan'85

- Abstract state into *single number*: $\Phi(\text{state})$
- Ensure for transitions: $\mathit{actual\ cost} - \Phi(\mathit{before}) + \Phi(\mathit{after}) = 0$
- Overall cost easy to determine: $\Phi(\mathit{initial\ state})$

Problems:

- Ingenuity required for application!

C. Okasaki, 1999, "Purely Functional Data Structures":

"[AA] is often unnecessarily efficient in practice. Unfortunately, [AA] breaks in the presence of persistence."

Our solution:

- Linear bounds 'allow' automatic inference

Underlying principle

Based on "Amortised Analysis" R.E.Tarjan'85

- Abstract state into *single number*: $\Phi(\text{state})$
- Ensure for transitions: $\mathit{actual\ cost} - \Phi(\mathit{before}) + \Phi(\mathit{after}) = 0$
- Overall cost easy to determine: $\Phi(\mathit{initial\ state})$

Problems:

- Ingenuity required for application!
- C. Okasaki, 1989,"Purely Functional Data Structures":

  [AA] is often tremendously effective in practice. Unfortunately, [AA] breaks in the presence of persistence.

Our solution: Use type and effect system!

- Linear bounds allow automatic inference
- Credits assigned per reference allows persistence

Underlying principle

Based on "Amortised Analysis" R.E.Tarjan'85

- Abstract state into *single number*: $\Phi(\text{state})$
- Ensure for transitions: $\mathit{actual\ cost} - \Phi(\mathit{before}) + \Phi(\mathit{after}) = 0$
- Overall cost easy to determine: $\Phi(\mathit{initial\ state})$

Problems:

- Ingenuity required for application!
- C. Okasaki, 1989, "Purely Functional Data Structures":

[AA] is often tremendously effective in practice.
Unfortunately, [AA] breaks in the presence of persistence.

Our solution: Use type-based system!

- Linear bounds allow automatic inference
- Credits assigned per reference allows persistence

Underlying principle

Based on "Amortised Analysis" R.E.Tarjan'85

- Abstract state into *single number*: $\Phi(\text{state})$
- Ensure for transitions: $\mathit{actual\ cost} - \Phi(\mathit{before}) + \Phi(\mathit{after}) = 0$
- Overall cost easy to determine: $\Phi(\mathit{initial\ state})$

Problems:

- Ingenuity required for application!
- C. Okasaki, 1989, "Purely Functional Data Structures":

[AA] is often tremendously effective in practice.
Unfortunately, [AA] breaks in the presence of persistence.

Our solution: Use type-based system!

- Linear bounds allow automatic inference
- Credits assigned per reference allows persistence

Underlying principle

Based on "Amortised Analysis" R.E.Tarjan'85

- Abstract state into *single number*: $\Phi(\text{state})$
- Ensure for transitions: $\textit{actual cost} - \Phi(\textit{before}) + \Phi(\textit{after}) = 0$
- Overall cost easy to determine: $\Phi(\textit{initial state})$

Problems:

- Ingenuity required for application!
- C. Okasaki, 1989, "Purely Functional Data Structures":

$$[AA] \textit{ is often tremendously effective in practice.}$$
$$\textit{Unfortunately, } [AA] \textit{ breaks in the presence of persistence.}$$

Our solution: Use type-based system!

- Linear bounds allow automatic inference
- Credits assigned per reference allows persistence

Type Rule Example

$$\frac{\Gamma \vdash e_t : A \qquad \Gamma \vdash e_f : A}{\Gamma, x{:}\mathsf{bool} \vdash \texttt{if } x \texttt{ then } e_t \texttt{ else } e_f : A} \text{ (Conditional)}$$

$\Gamma \vdash e : A \qquad \approx$ *term* e is of *type* A within *type-context* Γ

- Inference amounts to gathering constraints along typing
- Linear constraints allow efficient solving by LP-solvers
- Exchangeable cost parameters independent of soundness proof

Type Rule Example

$$\frac{\Gamma \vdash^{q_t}_{q'_t} e_t : A \qquad \Gamma \vdash^{q_f}_{q'_f} e_f : A}{\Gamma, x{:}\mathsf{bool} \vdash^{q}_{q'} \texttt{if } x \texttt{ then } e_t \texttt{ else } e_f : A} \text{(Conditional)}$$

$\Gamma \vdash^{q}_{q'} e : A \qquad \approx$ *term* e is of *type* A within *type-context* Γ

Evaluating e costs *at most* $q + \Phi(\Gamma)$ credits,
afterwards *at least* $\Phi(A) + q'$ left over

- Inference amounts to gathering constraints along typing
- Linear constraints allow efficient solving by LP-solvers
- Exchangeable cost parameters independent of soundness proof

Type Rule Example

$$\frac{\Gamma \vdash^{q_t}_{q'_t} e_t : A \qquad \Gamma \vdash^{q_f}_{q'_f} e_f : A}{\Gamma, x{:}\mathsf{bool} \vdash^{q}_{q'} \texttt{if } x \texttt{ then } e_t \texttt{ else } e_f : A} \text{(Conditional)}$$

$\Gamma \vdash^{q}_{q'} e : A \qquad \approx$ *term* e is of *type* A within *type-context* Γ

Evaluating e costs *at most* $q + \Phi(\Gamma)$ credits,
afterwards *at least* $\Phi(A) + q'$ left over

- Inference amounts to gathering constraints along typing
- Linear constraints allow efficient solving by LP-solvers
- Exchangeable cost parameters independent of soundness proof

Type Rule Example

$$\psi_0 = \left\{ \begin{array}{ll} q_t \leq q - \texttt{KifT} & q'_t \geq q' + \texttt{KifT}' \\ q_f \leq q - \texttt{KifF} & q'_f \geq q' + \texttt{KifF}' \end{array} \right\}$$

$$\frac{\Gamma \vdash^{q_t}_{q'_t} e_t : A \mid \psi_1 \qquad \Gamma \vdash^{q_f}_{q'_f} e_f : A \mid \psi_2}{\Gamma, x{:}\mathsf{bool} \vdash^{q}_{q'} \texttt{if } x \texttt{ then } e_t \texttt{ else } e_f : A \mid \psi_0 \cup \psi_1 \cup \psi_2} \text{(Conditional)}$$

$\Gamma \vdash^{q}_{q'} e : A \mid \psi \approx$ *term* e is of *type* A within *type-context* Γ

Evaluating e costs *at most* $q + \Phi(\Gamma)$ credits,
afterwards *at least* $\Phi(A) + q'$ left over — if ψ is satisifed

- Inference amounts to gathering constraints along typing
- Linear constraints allow efficient solving by LP-solvers
- Exchangeable cost parameters independent of soundness proof

Type Rule Example

$$\psi_0 = \left\{ \begin{array}{ll} q_t \leq q - \mathtt{KifT} & q'_t \geq q' + \mathtt{KifT}' \\ q_f \leq q - \mathtt{KifF} & q'_f \geq q' + \mathtt{KifF}' \end{array} \right\}$$

Constant depending on cost model:

	KifT	KifT′	KifF	KifF′
Heap	0	0	0	0
Stack	−1	0	−1	0
Time	30	0	30	3

$$\frac{\Gamma \vdash^{q_t}_{q'_t} e_t : A \mid \psi_1 \qquad \Gamma \vdash^{q_f}_{q'_f} e_f : A \mid \psi_2}{\Gamma, x{:}\mathsf{bool} \vdash^{q}_{q'} \mathtt{if}\ x\ \mathtt{then}\ e_t\ \mathtt{else}\ e_f : A \mid \psi_0 \cup \psi_1 \cup \psi_2}\ (\textsc{Conditional})$$

$\Gamma \vdash^{q}_{q'} e : A \mid \psi$ ≈ *term* e is of *type* A within *type-context* Γ

Evaluating e costs *at most* $q + \Phi(\Gamma)$ credits,
afterwards *at least* $\Phi(A) + q'$ left over — if ψ is satisifed

- Inference amounts to gathering constraints along typing
- Linear constraints allow efficient solving by LP-solvers
- Exchangeable cost parameters independent of soundness proof

Soundness Theorem (simplified)

IF term e is well-typed $\quad \Gamma \vdash^{q}_{q'} e : A \mid \psi$

and evaluates $\quad \mathcal{V}, \mathcal{H} \vdash e \rightsquigarrow \ell, \mathcal{H}'$

under well-formed initial state $\quad \mathcal{H} \vDash \mathcal{V} : \Gamma$

THEN

for all $p \in \mathbb{N}$ such that $\quad p \geq q + \Phi_{\mathcal{H}}(\mathcal{V}:\Gamma)$

there exists $p' \in \mathbb{N}$ satisfying $\quad p' \geq q' + \Phi_{\mathcal{H}}(\ell:A)$

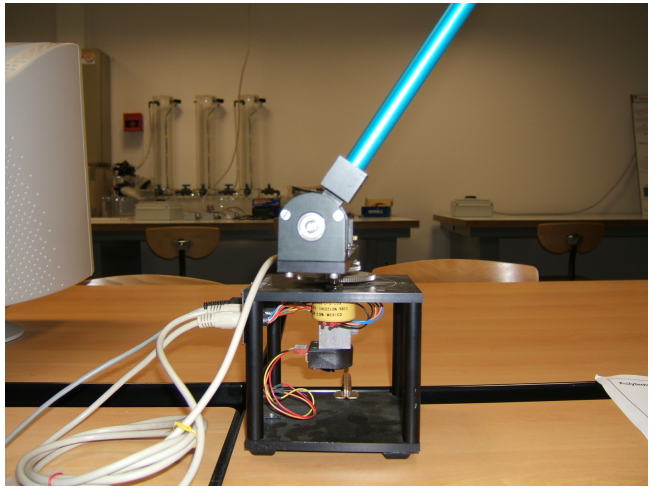
SUCH THAT

e evaluates with these restricted resources $\quad \mathcal{V}, \mathcal{H} \vdash^{p}_{p'} e \rightsquigarrow \ell, \mathcal{H}'$

- Proof independent of resource metric
- Separate theorem dealing with non-termination
- Note $p, p' \in \mathbb{N}$, but $q, q' \in \mathbb{Q}^{+}$

Inverted Pendulum

- Real-time control engineering problem (∼180 lines)
- Performed and measured using Renesas M32C/85U test board



- Results:
  - 36118 – 47635 clock cycles observed on typical iterations
  - 63678 clock cycles inferred upper bound on *WCET* (33.7%)
  - *Stack Space:* Exact Prediction!
  - *Dynamic Memory:* Exact Prediction!
- Analysis generates 1115 linear constraints over 2214 variables, solved in 0.67s on 1.73Ghz Pentium M, 2MB cache

Communicating understandable results

Example: `RBinsert: int, rbtree -> rbtree`

Initially, the user received:

```
ARTHUR3 typing for HumeHeapBoxed:
(int,rbtree[Leaf|Node<10>:colour[Red|Black<18>],#,int,#])
   -(20/0)->   rbtree[Leaf|Node:colour[Red|Black],#,int,#]
```

Nowadays:

```
Worst-case Heap-units required to call RBinsert:
  20 + 10*X1 + 18*X2
    where
        X1 = number of "Node" nodes at 1. position
        X2 = number of "Black" nodes at 1. position
```

Note: *data-dependend* bounds rather than size-dependent

Communicating understandable results

Example: `RBinsert:  int, rbtree -> rbtree`

Initially, the user received:

```
ARTHUR3 typing for HumeHeapBoxed:
(int,rbtree[Leaf|Node<10>:colour[Red|Black<18>],#,int,#])
   -(20/0)->   rbtree[Leaf|Node:colour[Red|Black],#,int,#]
```

Nowadays:

```
Worst-case Heap-units required to call RBinsert:
  20 + 10*X1 + 18*X2
    where
        X1 = number of "Node" nodes at 1. position
        X2 = number of "Black" nodes at 1. position
```

Note: *data-dependend* bounds rather than size-dependent

Past & Future Research on Amortised Analysis

Past:

Heap usage for first-order language	Hofmann & Jost, POPL'03
Featherweight Java	Hofmann et al., ESOP'06, EACSL'09
Stack space usage & Depth	Campbell, ESOP'09
WCET, Algebraic Datatypes & Cost Genericity	Jost et al., FM'09
Higher-order & Polymorphism	Jost et al., POPL'10

Future:

Non-linear bounds	Hofmann & Hoffmann, Munich
Lazy evaluation	Simões & Vasconcelos, Porto
Combination with Sized Types	Jost et al., St Andrews
Negative credits for montone resources	
Assigning credits to numeric types	

Summary

Program analysis for fully recursive eager functional language

Automatic	Just push the button!
Static	Compile-time analysis!
Efficient	Examples required $< 2s$ at 1.73GHz Pentium M!
Modular	Just provide annotated types!
Transparent	Constraints/Costs linked to source code!
Reliable	Formal Proof!
Generic	Heap- & Stack-space, WCET, Call Count,... BYOB

Limitations

- Analysis may fail
- Only linear bounds inferred ...thus far!
- Eager evaluation model ...thus far!

Color-Testslide

This TEXT is normal. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is red. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is green. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is blue. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is yellow. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is darkred. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is darkgreen. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is darkblue. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is darkyellow. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is purple. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is black. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$
This TEXT is grey. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$

This TEXT is normal. $1 + 2 = \vec{v}$ $\bullet \bullet \bullet$

Informal Cost Calculation

$$\texttt{rev\_acc} :: \mathsf{list}(\ \ \mathsf{int}), \mathsf{list}(\ \ \mathsf{int}) \ \longrightarrow\ \mathsf{list}(\ \ \mathsf{int})$$
$$\texttt{rev\_acc}([3,4],[2,1]) = \ \ [4,3,2,1]$$

$$\left.\begin{array}{lcl} \texttt{rev\_acc}([\,], acc) & = & acc \\ \texttt{rev\_acc}(h\!:\!t, acc) & = & \texttt{rev\_acc}(t, h\!:\!acc) \end{array}\right|$$

Informal Cost Calculation

$\texttt{rev\_acc} :: \text{list(u;int)}, \text{list(v;int)} \xrightarrow[y]{x} \text{list(w;int)}$

$\texttt{rev\_acc}([3,4],[2,1]) = [4,3,2,1]$

$$\left.\begin{array}{lcl} \texttt{rev\_acc}([\,], acc) & = & acc \\ \texttt{rev\_acc}(h:t, acc) & = & \texttt{rev\_acc}(t, h:acc) \end{array}\right|$$

- Add unique resource variables to type to denote credits
  $\texttt{rev\_acc}(a,b)=c \rightsquigarrow u|a| + v|b| + x \geq actual\_cost + w|c| + y$

Informal Cost Calculation

$\texttt{rev\_acc} :: \text{list(u;int)}, \text{list(v;int)} \xrightarrow[y]{x} \text{list(w;int)}$

$\texttt{rev\_acc}([3,4],[2,1]) = [4,3,2,1]$

$$\begin{array}{lcl} \texttt{rev\_acc}([\,], acc) & = & acc \\ \texttt{rev\_acc}(h\!:\!t, acc) & = & \texttt{rev\_acc}(t, h\!:\!acc) \end{array} \;\Big|\; \{x \geq y\,,\; v \geq w\}$$

- Add unique resource variables to type to denote credits
  $\texttt{rev\_acc}(a,b)=c \;\rightsquigarrow\; u|a| + v|b| + x \geq actual\_cost + w|c| + y$
- Consider each path of computation

Informal Cost Calculation

$\texttt{rev\_acc} :: \text{list(u;int)}, \text{list(v;int)} \xrightarrow[y]{x} \text{list(w;int)}$

$\texttt{rev\_acc}([3,4],[2,1]) = [4,3,2,1]$

$$\begin{array}{lcl} \texttt{rev\_acc}([\,], acc) & = & acc \\ \texttt{rev\_acc}(h{:}t, acc) & = & \texttt{rev\_acc}(t, h{:}acc) \end{array} \;\Bigg|\; \begin{array}{l} \{x \geq y\,,\ v \geq w\} \\ \{x + u \geq \texttt{Kcons} + v + p, \\ p \geq x, p \geq x - y + y\} \end{array}$$

- Add unique resource variables to type to denote credits
  $\texttt{rev\_acc}(a,b) = c \ \leadsto\ u|a| + v|b| + x \geq actual\_cost + w|c| + y$
- Consider each path of computation

Informal Cost Calculation

$\texttt{rev\_acc} :: \text{list(u;int)}, \text{list(v;int)} \xrightarrow[y]{x} \text{list(w;int)}$

$\texttt{rev\_acc}([3,4],[2,1]) = [4,3,2,1]$

$$\begin{array}{lcl|l} \texttt{rev\_acc}([\,], acc) & = & acc & \{x \geq y\,,\ v \geq w\} \\ \texttt{rev\_acc}(h\!:\!t, acc) & = & \texttt{rev\_acc}(t, h\!:\!acc) & \{u \geq \texttt{Kcons} + v\} \end{array}$$

- Add unique resource variables to type to denote credits
  $\texttt{rev\_acc}(a,b)=c \ \rightsquigarrow\ u|a| + v|b| + x \geq actual\_cost + w|c| + y$
- Consider each path of computation
- Solve constraints using a standard LP-solver
  if $\texttt{Kcons} = 4$ then $x = 0,\ y = 0,\ u = 4,\ v = 0,\ w = 0$

Informal Cost Calculation

$\texttt{rev\_acc} :: \text{list}(4;\text{int}), \text{list}(0;\text{int}) \xrightarrow[0]{0} \text{list}(0;\text{int})$

$\texttt{rev\_acc}([3,4],[2,1]) = [4,3,2,1]$

$$\begin{array}{lcl} \texttt{rev\_acc}([\,], acc) & = & acc \\ \texttt{rev\_acc}(h:t, acc) & = & \texttt{rev\_acc}(t, h:acc) \end{array} \;\Bigg|\; \begin{array}{l} \{0 \geq 0\,,\, 0 \geq 0\} \\ \{4 \geq \;\; 4 \;\; + 0\} \end{array}$$

- Add unique resource variables to type to denote credits
  $\texttt{rev\_acc}(a,b)=c \;\leadsto\; u|a| + v|b| + x \geq \mathit{actual\_cost} + w|c| + y$
- Consider each path of computation
- Solve constraints using a standard LP-solver
  if $\texttt{Kcons} = 4$ then $x = 0,\ y = 0,\ u = 4,\ v = 0,\ w = 0$
- **Hence the call** $\texttt{rev\_acc}(\texttt{a},\texttt{b})=\texttt{c}$ **requires** $\texttt{Kcons} \cdot |a|$ **units**

Big-step semantics vs. Small-step semantics

Problem Big-step semantics fit structure of type-rules,
but ill-suited for measuring time and stack cost

Solution Normal terminal rules account for pushing,
non-terminal only account for popping!

Example

$$\texttt{let}\ x = f\ 5\ \texttt{in}\ g\ x\ x$$

translates automatically into let-normal form according to policy to

$$\texttt{let}\ x = (\texttt{LET}\ a = 5\ \texttt{IN}\ f\ a)\ \texttt{in}\ \texttt{LET}\ c = x\ \texttt{IN}\ \texttt{LET}\ b = x\ \texttt{IN}\ g\ b\ c$$

with "LET ... IN ..." having zero costs

(as opposed to "let ... in ...")

Efficiency

Program	Code Lines	Constraints		Run-time FFI	
		Number	Variables	Total	LP-solve
biquad	400	2956	5756	1.43s	0.418s
cycab	270	3043	6029	2.75s	1.385s
gravdragdemo	190	2692	5591	2.14s	1.065s
matmult	100	21485	36638	84.17s	21.878s
meanshift	350	8110	15005	11.01s	6.414s
pendulum	190	1115	2214	0.67s	0.260s

biquad	Biquadratic filter application and communication
cycab	Messenging module for autonomous vehicle
gravdragdemo	Satellite tracking demo, using Kalman filer
matmult	Synthesized Hume from C-Code, heavy higher-order
meanshift	Lane tracking computer vision algorithm
pendulum	Real-time inverted pendulum controller

Example Results

Program	Cost Model	Analysis (N=10) absolute	ratio	Cost Model	Analysis (N=10) absolute	ratio
	Call Count			*Heap Space*		
`sum/fold`	22	22	1.00	88	88	1.00
`zipWith`	21	21	1.00	190	192	1.01
`repmin`	60	60	1.00	179	179	1.00
`rbInsert`	10	20	2.00	208	294	1.41
	Time			*Stack Space*		
`sum/fold`	16926	21711	1.28	34	34	1.00
`zipWith`	27812	32212	1.16	139	140	1.01
`repmin`	47512	58759	1.24	81	222	2.74
`rbInsert`	27425	43087	1.57	82	155	1.89

sum: Standard list-fold with higer order argument “add”

zipWith: Combining two lists with a higher-order argument

repmin: Replace leaves in binary tree with minimal element

rbInsert: Red-Black tree balanced insertion