Amortised Resource Analysis and Functional Correctness with Separation Logic (Part II)

Summer School on Formal Reasoning and Representation of Complex Systems

Robert Atkey
Robert.Atkey@cis.strath.ac.uk
University of Strathclyde

14th August 2010

Resource Specification and Verification

Programs execute.

But not for free.

How much will it cost?

How do we *say* how much it will cost?

Specifying Resource Usage

Maybe we could attach sizes to things:

$$\mathsf{list}^n(x)$$

And then state the resource consumption in these terms:

$$\{\mathsf{list}^n(x) \wedge r_c = r_1\}\ \texttt{iterateList}\ \{\mathsf{list}^n(x) \wedge r_c = r_1 + n\}$$

How well does this work?

Functional Queues

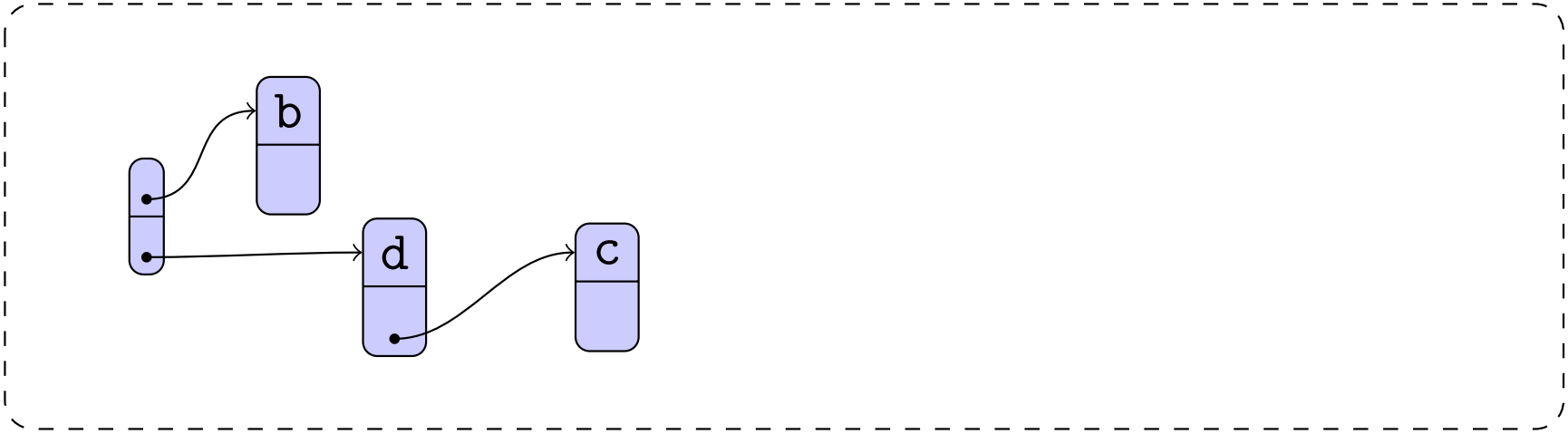
Functional Queues

- Dequeue (1 unit)

Functional Queues

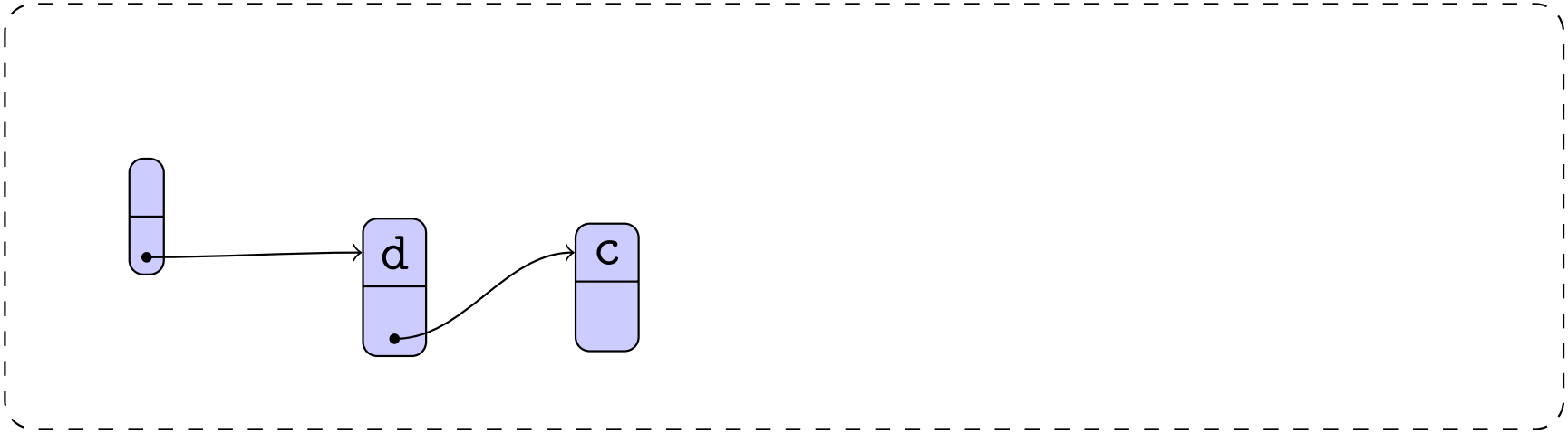
- Dequeue (1 unit)
- Enqueue (1 unit)

Functional Queues



- Dequeue (1 unit)
- Enqueue (1 unit)
- Enqueue (1 unit)

Functional Queues



- Dequeue (1 unit)
- Enqueue (1 unit)
- Enqueue (1 unit)
- Dequeue (1 unit)

Functional Queues

- Dequeue (1 unit)
- Enqueue (1 unit)
- Enqueue (1 unit)
- Dequeue (1 unit)
- Dequeue (3 units) (2 to reverse $[\mathtt{d}, \mathtt{c}]$, 1 to remove $\mathtt{c}$)

Functional Queues

- Dequeue (1 unit)
- Enqueue (1 unit)
- Enqueue (1 unit)
- Dequeue (1 unit)
- Dequeue (3 units) (2 to reverse $[\mathtt{d}, \mathtt{c}]$, 1 to remove $\mathtt{c}$)
- Total: 7 units

Specifying the Resource Behaviour

Using a ghost variable r_c for consumed resources.

Predicate $\mathsf{queue}(x, h, t)$

- Queue pointed to by x;
- Head of length h, tail of length t

$$
\begin{array}{rlll}
\forall r_1.\{\mathsf{queue}(x,h,t) \wedge r_c = r_1\} & \texttt{enqueue} & \{\mathsf{queue}(x,h,t+1) \wedge r_c = r_1 + R\} \\
\forall r_1.\{\mathsf{queue}(x,0,t) \wedge r_c = r_1\} & \texttt{dequeue} & \{\mathsf{queue}(x,t-1,0) \wedge r_c = r_1 + (1+t)R\} \\
\forall r_1.\{\mathsf{queue}(x,h+1,t) \wedge r_c = r_1\} & \texttt{dequeue} & \{\mathsf{queue}(x,h,t) \wedge r_c = r_1 + R\}
\end{array}
$$

Exposes the internals of the queue abstraction.

Amortised Analysis (Tarjan 1985)

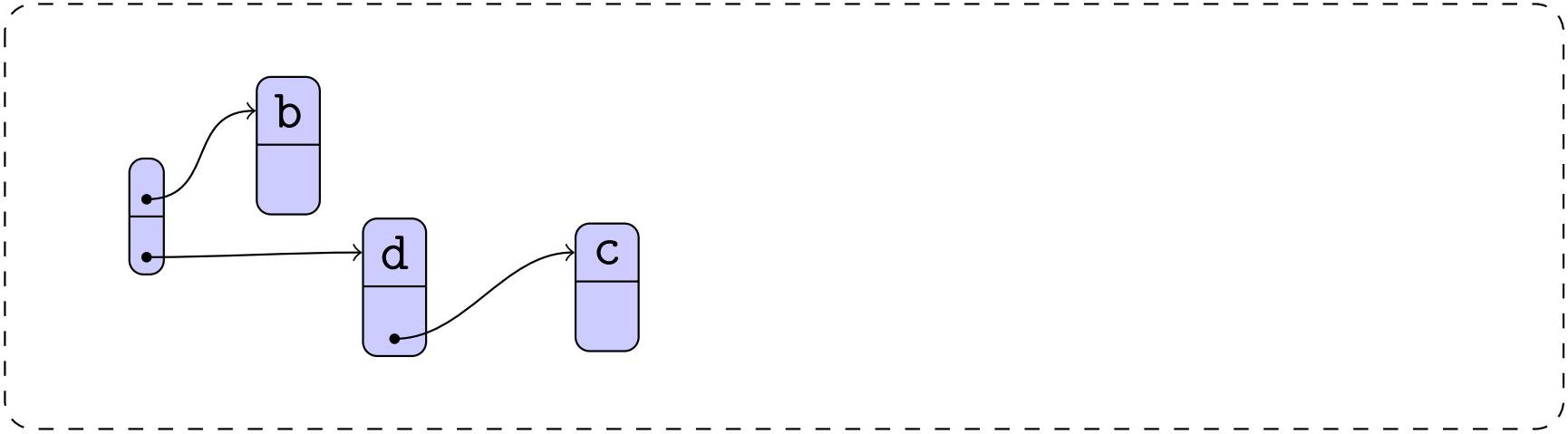
Amortised Analysis (Tarjan 1985)

- Dequeue (1 unit)

Amortised Analysis (Tarjan 1985)

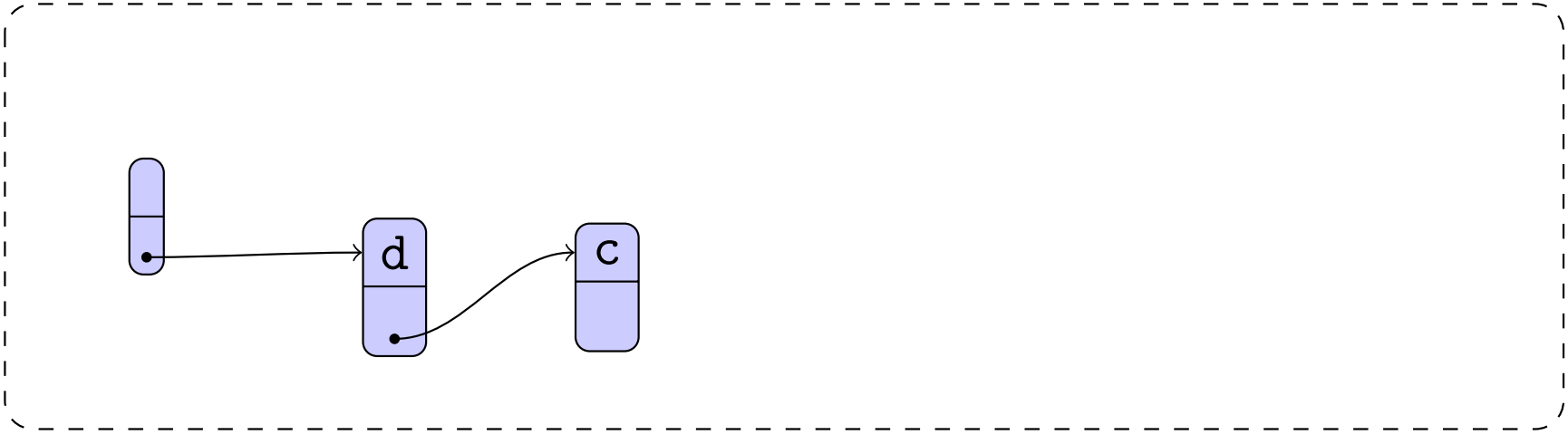
- Dequeue (1 unit)
- Enqueue (2 units)

Amortised Analysis (Tarjan 1985)



- Dequeue (1 unit)
- Enqueue (2 units)
- Enqueue (2 units)

Amortised Analysis (Tarjan 1985)



- Dequeue (1 unit)
- Enqueue (2 units)
- Enqueue (2 units)
- Dequeue (1 unit)

Amortised Analysis (Tarjan 1985)

- Dequeue (1 unit)
- Enqueue (2 units)
- Enqueue (2 units)
- Dequeue (1 unit)
- Dequeue (1 unit)

Amortised Analysis (Tarjan 1985)

- Dequeue (1 unit)
- Enqueue (2 units)
- Enqueue (2 units)
- Dequeue (1 unit)
- Dequeue (1 unit)
- Total: 7 units

Where Do Resources Live?

Tarjan: associate the extra resources for enqueue with the nodes.

Banker's method.

When accessing that node we get to use the resources.

Applied to functional languages by Hofmann and Jost (2003).

Where Do Resources Live?

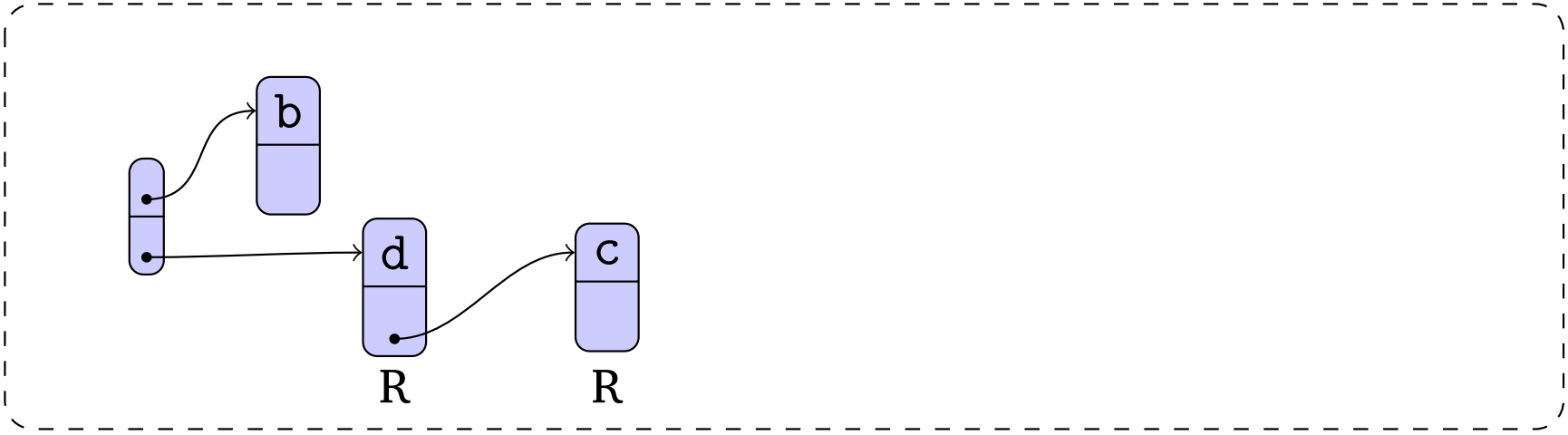
Where Do Resources Live?

- Dequeue (1 unit) (1 real unit)

Where Do Resources Live?

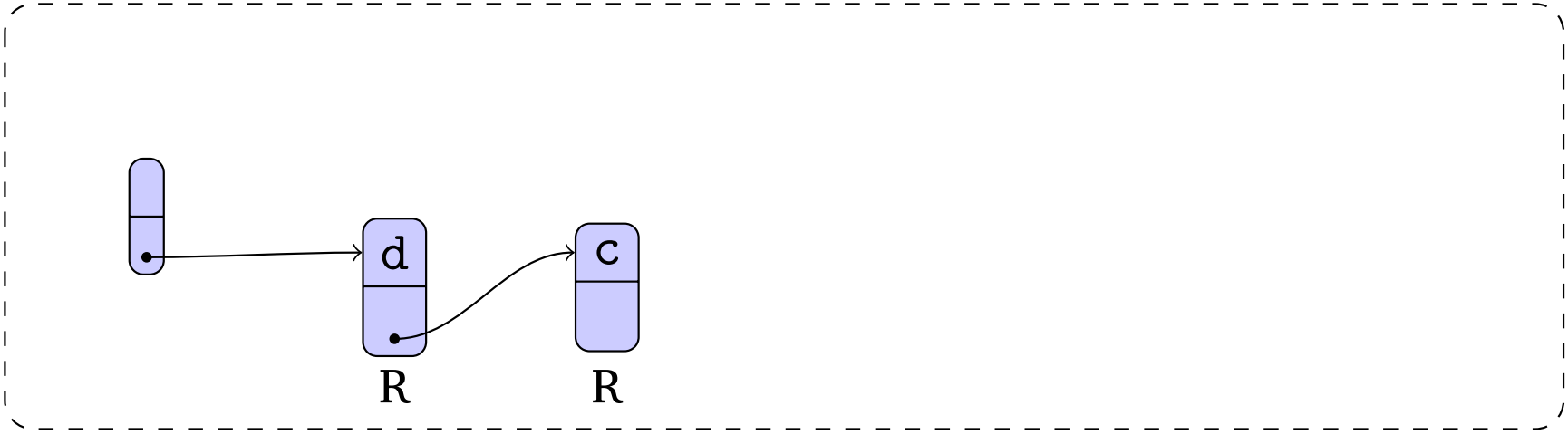
- Dequeue (1 unit) (1 real unit)
- Enqueue (2 units) (1 real unit)

Where Do Resources Live?



- Dequeue (1 unit) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Enqueue (2 units) (1 real unit)

Where Do Resources Live?

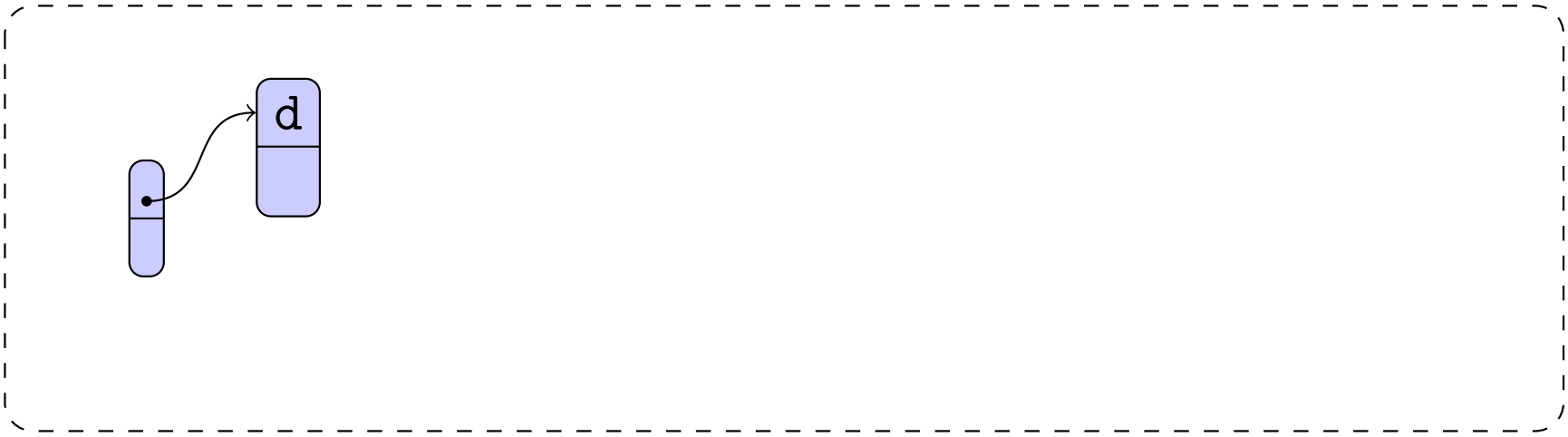


- Dequeue (1 unit) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Dequeue (1 unit) (1 real unit)

Where Do Resources Live?

- Dequeue (1 unit) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Dequeue (1 unit) (1 real unit)
- Dequeue (1 unit) (3 real units)

Where Do Resources Live?



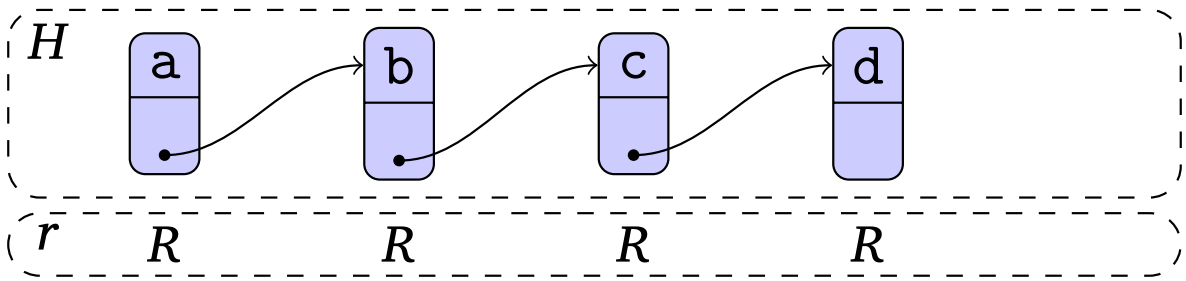
- Dequeue (1 unit) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Enqueue (2 units) (1 real unit)
- Dequeue (1 unit) (1 real unit)
- Dequeue (1 unit) (3 real units)
- Total: 7 units

Consumable Resources

Let's assume that resources are a commutative, ordered monoid.

Separation Logic with Consumable Resources

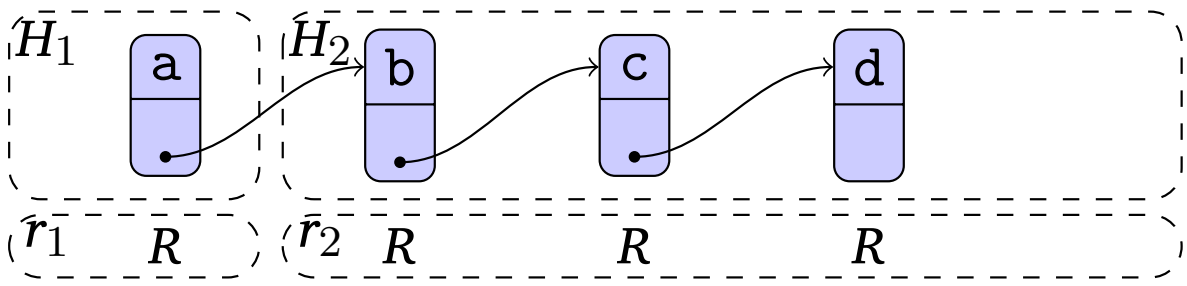
$$\mathsf{list}_R(x) \equiv \quad x = \mathsf{null} \wedge \mathsf{emp}$$
$$\vee\, \exists yz.\, [x \overset{\mathsf{data}}{\mapsto} y] * [x \overset{\mathsf{next}}{\mapsto} z] * R * \mathsf{list}_R(z)$$



$r, H \models \mathsf{list}_R(x)$

Separation Logic with Consumable Resources

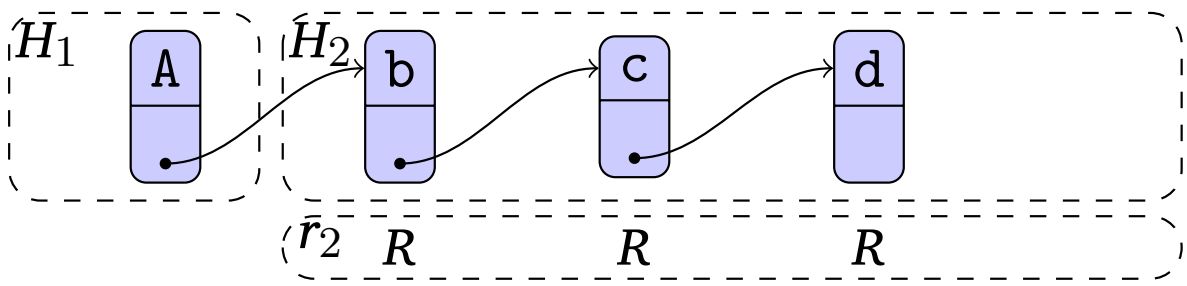
$$\mathsf{list}_R(x) \equiv \quad x = \mathsf{null} \wedge \mathsf{emp}$$
$$\vee\, \exists yz.\, [x \overset{\mathsf{data}}{\mapsto} y] * [x \overset{\mathsf{next}}{\mapsto} z] * R * \mathsf{list}_R(z)$$



$$r_1 \cdot r_2, H_1 \uplus H_2 \models [x \overset{\mathsf{data}}{\mapsto} \mathsf{a}] * [x \overset{\mathsf{next}}{\mapsto} y] * R * \mathsf{list}_R(y)$$

Separation Logic with Consumable Resources

$$\mathsf{list}_R(x) \equiv \quad x = \mathsf{null} \wedge \mathsf{emp}$$
$$\vee\, \exists yz.\, [x \overset{\mathsf{data}}{\mapsto} y] * [x \overset{\mathsf{next}}{\mapsto} z] * R * \mathsf{list}_R(z)$$



After some mutation and resource consumption.

Specifying Resources and Heap Shape

$$\mathsf{queue}(x) \equiv \exists yz.[x \overset{\mathsf{front}}{\mapsto} y] * [x \overset{\mathsf{back}}{\mapsto} z] * \mathsf{list}(0, y) * \mathsf{list}(1, z)$$

$$\{\mathsf{queue}(x) * R * R\}\texttt{enqueue}\{\mathsf{queue}(x)\}$$

$$\{\mathsf{queue}(x) * R\}\texttt{dequeue}\{\mathsf{queue}(x)\}$$

Precondition specifies:

$$\left.\begin{array}{l}\text{Heap shape required}\\ \text{Resources required}\end{array}\right\} \text{Two may be intertwined}$$

It's all intertwingly

Reasoning follows Innumerate Shepherd model

- Numbers and arithmetic are abstract nonsense!
- *Structure* matters

Resources are made available as they are needed to process the data they are attached to.

Integrates well with the local reasoning of Separation Logic.

Assertion Language

$$
\begin{aligned}
\phi \quad ::= \quad & t_1 \bowtie t_2 \mid \top \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \to \phi_2 \\
& \mid \mathsf{emp} \mid \phi_1 * \phi_2 \mid \phi_1 \mathrel{-\!\!*} \phi_2 \mid \forall x.\phi \mid \exists x.\phi \\
& \mid [t_1 \overset{\mathsf{f}}{\mapsto} t_2] \mid R_r \mid \ldots
\end{aligned}
$$

Semantic Domains

Logic is defined over pairs $x = (H, r)$.

Use a ternary relation to define how resources and heaps are combined:

$$Rxyz \;\Leftrightarrow\; H_1 \# H_2 \wedge H_1 \uplus H_2 = H_3 \wedge r_1 \cdot r_2 \sqsubseteq r_3$$
$$\text{where } x = (H_1, r_1), y = (H_2, r_2), z = (H_3, r_3)$$

Extend the order on resources to pairs of heaps and resources by $(H_1, r_1) \sqsubseteq (H_2, r_2)$ iff $H_1 = H_2$ and $r_1 \sqsubseteq r_2$.

Semantics of Assertions

$$x = (H, r)$$

η, x	$\models$	$\top$	iff	always
η, x	$\models$	$t_1 \ \{=, \neq\} \ t_2$	iff	$[\![t_1]\!]_\eta \ \{=, \neq\} \ [\![t_2]\!]_\eta$
η, x	$\models$	emp	iff	$x = (H, r)$ and $H = \{\}$
η, x	$\models$	$[t_1 \overset{f}{\mapsto} t_2]$	iff	$x = (H, r)$ and $H = \{([\![t_1]\!]_\eta, f) \mapsto [\![t_2]\!]_\eta\}$
η, x	$\models$	R_{r_i}	iff	$x = (H, r)$ and $r_i \sqsubseteq r$ and $H = \{\}$
η, x	$\models$	$\phi_1 \wedge \phi_2$	iff	$\eta, x \models \phi_1$ and $\eta, x \models \phi_2$
η, x	$\models$	$\phi_1 \vee \phi_2$	iff	$\eta, x \models \phi_1$ or $\eta, x \models \phi_2$
η, x	$\models$	$\phi_1 * \phi_2$	iff	exists y, z. st. $Ryzx$ and $\eta, y \models \phi_1$ and $\eta, z \models \phi_2$
η, x	$\models$	$\phi_1 \rightarrow \phi_2$	iff	for all y. if $x \sqsubseteq y$ and $\eta, y \models \phi_1$ then $\eta, y \models \phi_2$
η, x	$\models$	$\phi_1 \mathrel{-\!\!*} \phi_2$	iff	for all y, z. if $Rxyz$ and $\eta, y \models \phi_1$ then $\eta, z \models \phi_2$
η, x	$\models$	$\forall v.\phi$	iff	for all a, $\eta[v \mapsto a], x \models \phi$
η, x	$\models$	$\exists v.\phi$	iff	exists a, $\eta[v \mapsto a], x \models \phi$

Examples of Inductive Predicates

List segments:

$$\begin{aligned}\mathsf{lseg}(r,x,y) \equiv\ & (x = y \wedge \mathsf{emp}) \\ & \vee\ (\exists d, z.\ [x \overset{\mathsf{data}}{\mapsto} d] * [x \overset{\mathsf{next}}{\mapsto} z] * R_r * \mathsf{lseg}(r,z,y))\end{aligned}$$

Doubly-linked lists:

$$\begin{aligned}\mathsf{dlseg}(r,p,x,y) \equiv\ & (x = y \wedge \mathsf{emp}) \\ & \vee\ (\exists z.\ [x \overset{\mathsf{next}}{\mapsto} z] * [x \overset{\mathsf{prev}}{\mapsto} p] * R_r * \mathsf{dlseg}(r,x,z,y))\end{aligned}$$

Trees:

$$\begin{aligned}\mathsf{tree}(r,x) \equiv\ & (x = \mathit{null} \wedge \mathsf{emp}) \\ & \vee\ (\exists y, z.\ [x \overset{\mathsf{left}}{\mapsto} y] * [x \overset{\mathsf{right}}{\mapsto} z] * R_r * \mathsf{tree}(y) * \mathsf{tree}(z))\end{aligned}$$

Examples of Inductive Predicates

Data-dependency (resources only available for non-0 elements):

$$\begin{aligned}&\mathsf{lseg}^{\neq 0}(r, x, y) \equiv \\ &\quad (x = y \wedge \mathsf{emp}) \\ &\quad \vee (\exists d, z.\ [x \overset{\mathsf{data}}{\mapsto} d] * [x \overset{\mathsf{next}}{\mapsto} z] * (d \neq 0 \to R_r) * \mathsf{lseg}^{\neq 0}(r, z, y))\end{aligned}$$

Compare to explicit resource counting version:

$$\mathsf{lseg}(l, x, y) \wedge r = \mathsf{length}(\mathsf{filter}(\lambda x.x \neq 0, l))$$

Numerical Bounds not Excluded

Can still write:

$$\mathsf{list}^n(x) * R^n$$

Where

$$R^n \equiv (n = 0 \wedge \mathsf{emp}) \vee (R * R^{n-1})$$

But lose the advantages of locality.

Formal Development in Coq

- Subset of Java bytecode (w/o virtual methods or exceptions)
- Standard semantics + abstract costs
- Resources consumed by `consume` instruction

Layered approach to program logic:

- Shallow embedding (uses Coq as assertion logic)
- Deep embedding on top (formalised assertion logic in Coq)
- Verification condition generator

Proved (formally!):

- Soundness of the logic
- Certified verification condition generator

Semantics and Soundness (simplified)

Given program P

- State is a triple $\langle r, H, frm \rangle$.
- Small step semantics: $P \vdash \langle r, H, frm \rangle \rightarrow \langle r', H', frm' \rangle$.

Given certificate $C : offsets(P) \rightarrow \mathsf{Assn}$:

$$\mathsf{safeState}(\langle r, H, frm \rangle, r_{total}) \equiv \exists r_{future}. \quad r * r_{future} \sqsubseteq r_{total} \wedge (r_{future}, H) \models C(\mathsf{pc}(frm))$$

Proved (if certificate is OK):

- safeState is preserved by steps;
- On termination, consumed resources are less than r_{total}.

Automated Verification

Two possible ways:

- Generate Verification Conditions, try to prove them
- Symbolic execution approach

Did the first for expository reasons;
second is more popular for Separation Logic.

Method and implementation are proof of concept; basic idea can be reused with other automated verification techniques.

Generating Verification Conditions

Verification conditions can be generated for methods by weakest precondition.

$$
\begin{aligned}
\mathsf{wp}(\texttt{consume(R)}, pc) &= R * C(pc+1) \\
\mathsf{wp}(\texttt{return}, pc) &= Q \\
\mathsf{wp}(\texttt{goto n}, pc) &= C(n) \\
\mathsf{wp}(\texttt{ifnull n}, pc) &= (s_0 = \mathsf{null} \rightarrow C(n)) \wedge (s_0 \neq \mathsf{null} \rightarrow C(pc+1)) \\
\mathsf{wp}(\texttt{call pname}, pc) &= P' * (Q' \mathrel{-\!\!*} C(pc+1))
\end{aligned}
$$

where:

- Q is post condition of current method;
- (P', Q') is the specification of `pname`.
- C is pre-seeded with loop invariants

Frame rule is implicit in procedure call rule.

Solving Verification Conditions

Restricted form of assertion (closed under VC generation):

Restriction: only allows *linear* resource bounds.

Data: $P := t_1 = t_2 \mid t_1 \neq t_2 \mid \top$

Heap: $X := [t_1 \overset{\mathsf{f}}{\mapsto} t_2] \mid \mathsf{lseg}(\Theta, t_1, t_2) \mid \mathsf{emp}$

Resource: $R := R_r \mid \top$

Data: $\Pi := P_1, \ldots, P_n \quad (P_1 \wedge \ldots \wedge P_n)$

Heap: $\Sigma := X_1, \ldots, X_n \quad (X_1 * \ldots * X_n)$

Resource: $\Theta := R_1, \ldots, R_n \quad (R_1 * \ldots * R_n)$

$$S := \bigvee_i (\Pi \wedge (\Sigma * \Theta))$$

$$G \quad := \quad S * G \mid S \mathrel{-\!\!*} G \mid S \mid G_1 \wedge G_2 \mid P \rightarrow G \mid \forall x.G \mid \exists x.G$$

Solving Verification Conditions

Proof search, using I/O model of Cervesato, Hodas and Pfenning.

Main judgement: $\Pi|\Sigma|\Theta \vdash G$.

Auxilliary judgements:

$\Pi\|\Sigma\|\Theta \vdash \Sigma_1 \backslash \Sigma_2, \Theta'$	Heap assertion matching
$\Theta \vdash \Theta_1 \backslash \Theta_2$	Resource matching
$\Pi \vdash \bot$	Contradiction spotting
$\Pi \vdash \Pi'$	Data entailment

Proof Search

Three phases, repeat until done:

- Goal driven, switching on the shape of the goal;
- Saturate the context
  - Infer facts from context members
  - Unfold list predicates according to heuristics
- Look for contradictions

With possible backtracking.

Example goal-directed rule:

$$\frac{\text{exists } i. \quad \Pi|\Sigma|\Theta \vdash \Sigma_i \backslash \Sigma', \Theta' \qquad \Pi \vdash \Pi_i \qquad \Theta' \vdash \Theta_i \backslash \Theta'' \qquad \Pi|\Sigma'|\Theta'' \vdash G}{\Pi|\Sigma|\Theta \vdash \bigvee_i (\Pi_i \wedge (\Sigma_i * \Theta_i)) * G}$$

Resource Annotation Inference

This approach requires that we know the resource amounts to annotate our lists with beforehand.

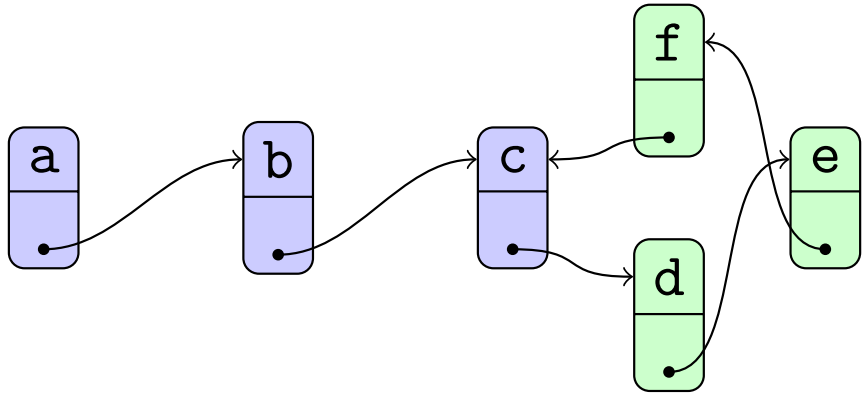
Better to infer them (as with Hofmann and Jost's system).

1. Replace all resource elements with linear expressions.
2. Resource matching rule generates linear constraints:

$$e_1 \vdash e_2 \backslash e_1 - e_2, e_2 \leq e_1$$

3. Solve using standard LP solver (GLPK).

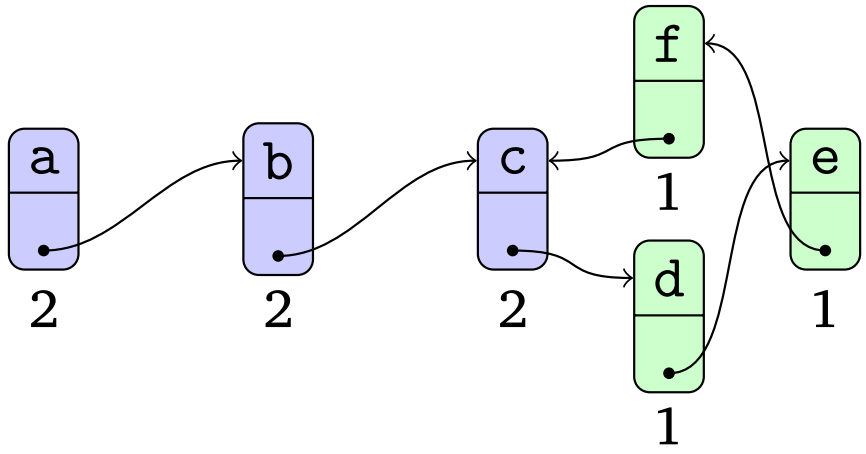
Frying Pan List Reversal



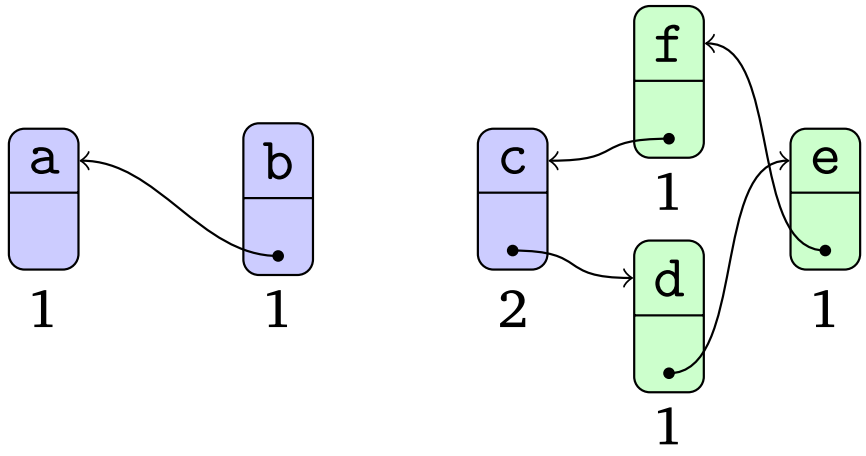
- Handle: a, b, c
- Pan: d, e, f.

$$\exists k.\mathsf{lseg}(p_1, v_0, v_1) * [v_1 \overset{\mathsf{next}}{\mapsto} k] * \mathsf{lseg}(p_2, k, v_1) * R^{p_3}$$

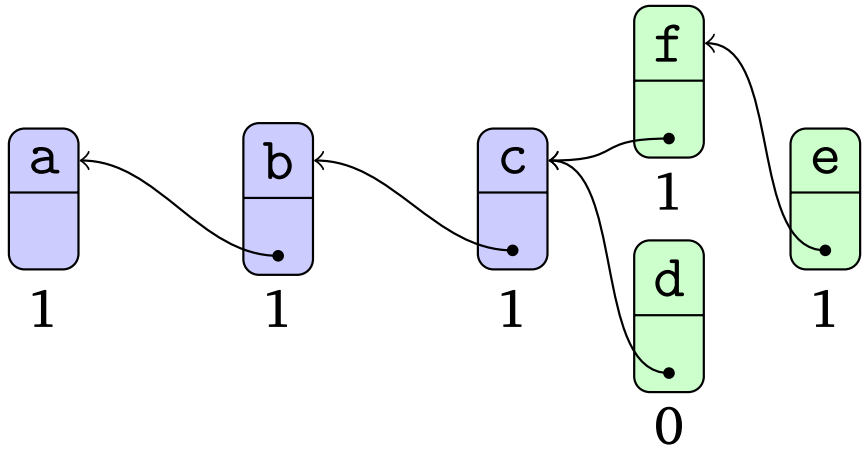
Frying Pan List Reversal



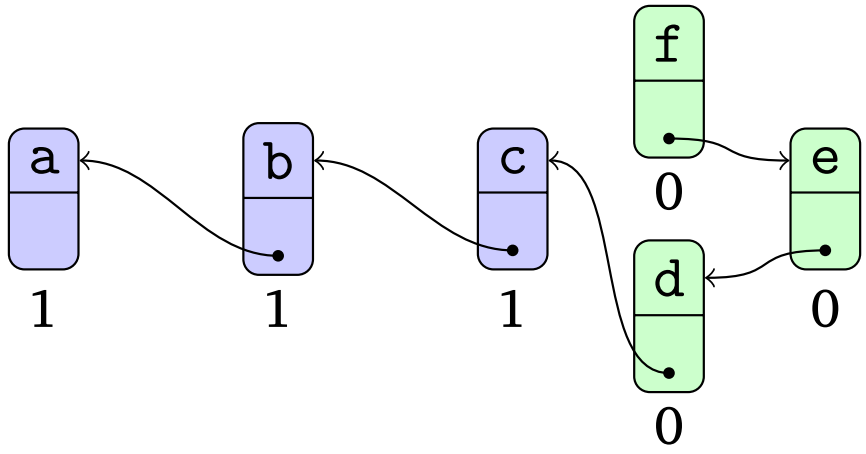
Frying Pan List Reversal



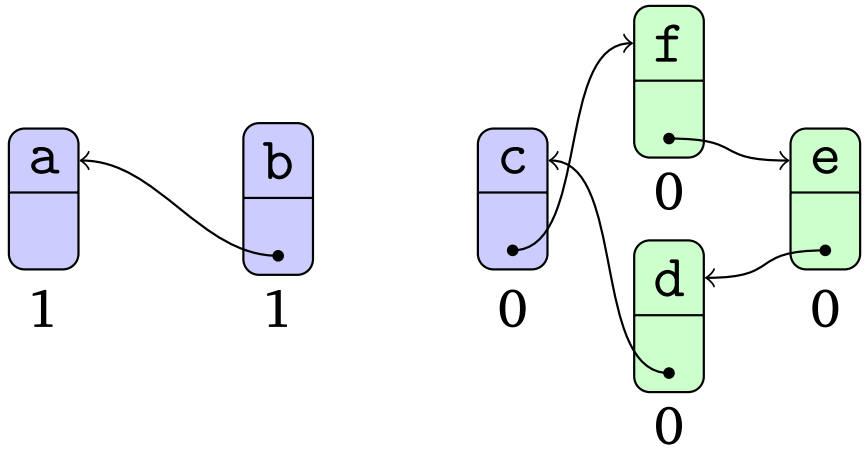
Frying Pan List Reversal



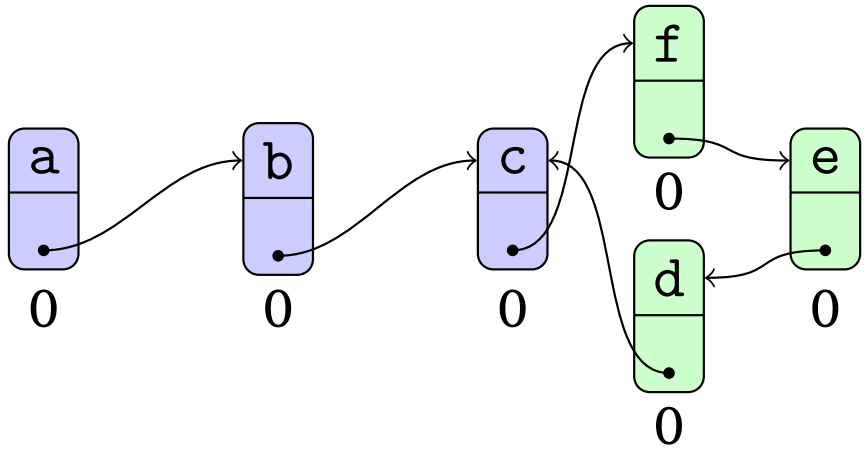
Frying Pan List Reversal



Frying Pan List Reversal



Frying Pan List Reversal



Frying Pan List Reversal

Loop invariant (Brotherston, Bornat and Calcagno):

$$
\begin{aligned}
&(\exists k.\mathsf{lseg}(a_1, l_0, v_1) * \mathsf{lseg}(a_2, l_1, \mathtt{null}) * [v_1 \overset{\mathsf{next}}{\mapsto} k] * \mathsf{lseg}(a_3, k, v_1) * R^{a_4}) \\
\vee\ &(\exists k.\mathsf{lseg}(b_1, k, \mathtt{null}) * [j \overset{\mathsf{next}}{\mapsto} k] * \mathsf{lseg}(b_2, l_0, v_1) * \mathsf{lseg}(b_3, l_1, j) * R^{b_4}) \\
\vee\ &(\exists k.\mathsf{lseg}(c_1, l_0, \mathtt{null}) * \mathsf{lseg}(c_2, l_1, v_1) * [v_1 \overset{\mathsf{next}}{\mapsto} k] * \mathsf{lseg}(c_3, k, v_1) * R^{c_4})
\end{aligned}
$$

Corresponds to the three phases.

Annotated with variables for resources, to be filled in by linear program solver.

Frying Pan List Reversal

After constraint solving:

Pre-condition	$p_1 = 2$	$p_2 = 1$	$p_3 = 2$	
Loop invariant, phase 1	$a_1 = 2$	$a_2 = 1$	$a_3 = 1$	$a_4 = 2$
Loop invariant, phase 2	$b_1 = 1$	$b_2 = 1$	$b_3 = 0$	$b_4 = 1$
Loop invariant, phase 3	$c_1 = 1$	$c_2 = 0$	$c_3 = 0$	$c_4 = 0$
Post-condition	$x'_1 = 0$	$x'_2 = 0$	$x'_3 = 0$	

Further Examples

- List iteration
- (Non-cyclic) list reversal
- List copying
- Tree copying/mirroring
- Inner loop of in-place merge sort

More connectives?

The current logic has separating conjunction:

$$X * Y$$

meaning that X and Y use separate heap *and* separate resources.

What about another two connectives:

- Separate heap, but not resources
- Separate resources, but not heap

Need for "separate resource, but not heap" has shown up in small examples already.

Related Work

- Inspired by work of Hofmann and Jost
  - Amortised analysis for (linear) functional programs
  - Use of linear programming solvers
- Work continued at Edinburgh, St. Andrews and Munich
  - Stack depth (Campbell)
  - Polymorphism, higher order (Jost, Loidl, Hammond, Hofmann)
  - RAJA (Hofmann, Jost, Rodriguez)
  - Polynomial bounds (Hoffmann and Hofmann)
- Other resource accounting techniques:
  - COSTA (Albert, Arenas, Genaim, Puebla, Zanardini)
  - SPEED (Gulwani, Mehra, Chilimbi)

Conclusions

Amortised resource analysis and Separation Logic are practically made for each other.

Attaching resources to data structures:

- eases specification;
- eases verification.

Seems easy to extend:

- Resources that depend on data;
- Resource aquisition;
- Polynomial bounds?