Separation Logic Goes 

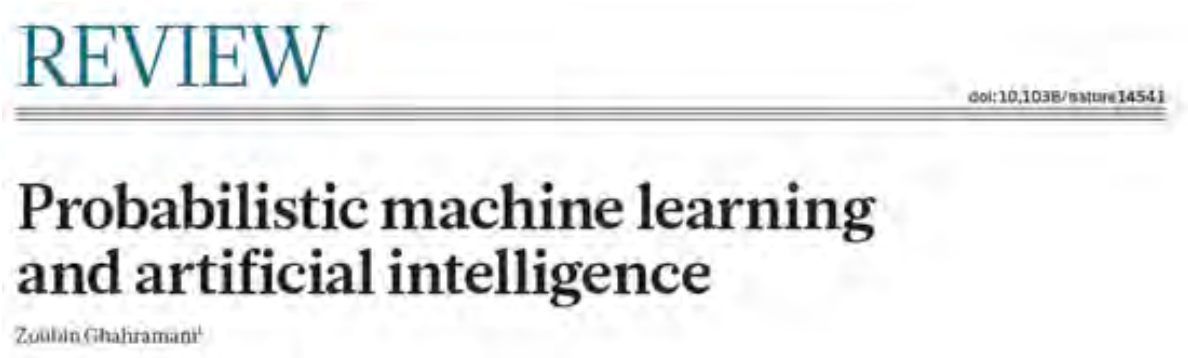
Joost-Pieter Katoen

Verified Software Workshop, Isaac Newton Institute, Cambridge 2019

Overview

Perspective in *Nature*

"There are several reasons why probabilistic programming could prove to be revolutionary for machine intelligence and scientific modelling."



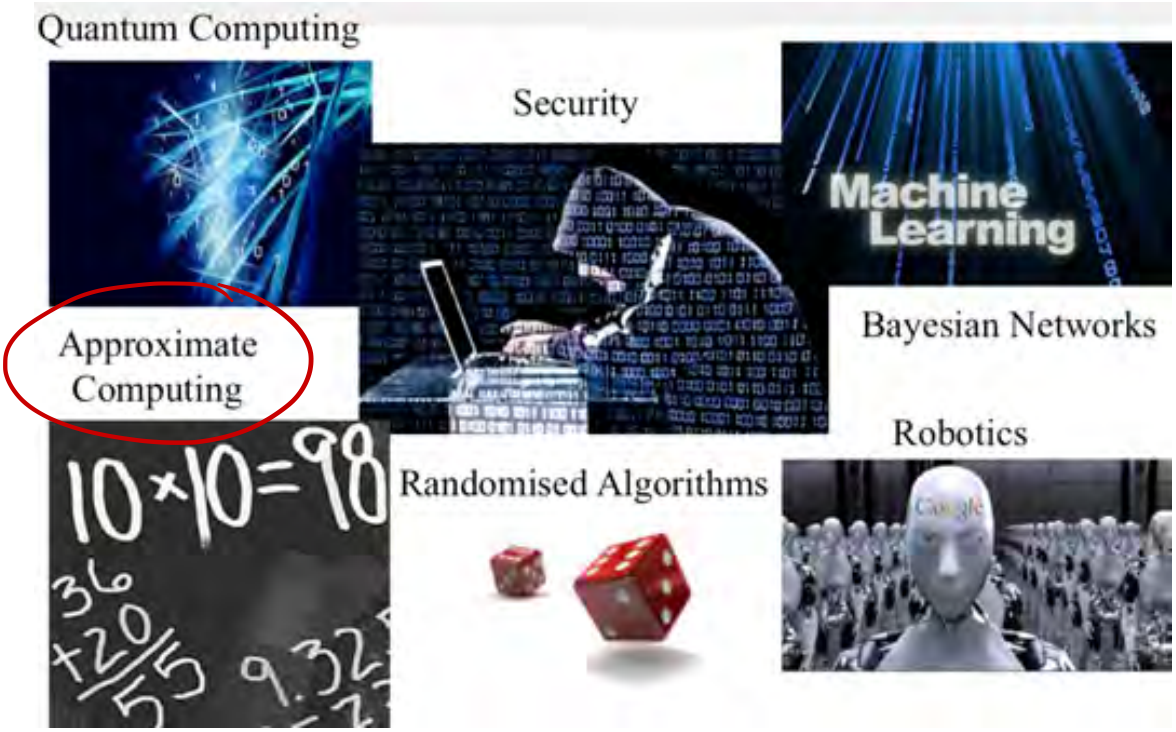
REVIEW

doi:10.1038/nature14541

Probabilistic machine learning and artificial intelligence

Zoubin Ghahramani[1]

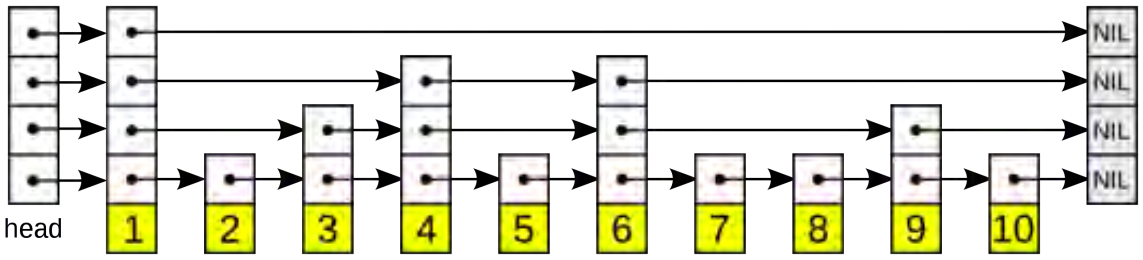
Probabilistic Programming



Scenic

Popular PPL: STAN (> 10,000 active users, 33 releases)
Almost every PL has a probabilistic version!
probabilistic-programming.org

Today's Focus



"Randomised skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster, and use less space."

[Pugh, 1989]

"The expected running time of randomised splay trees is smaller than deterministic variants"

[Albers and Karpinski, 2002]

Today's Focus

Can we formally prove programs
that flip coins and manipulate pointers?

At the source code level.
No "descend" in some operational model.
No ad-hoc arguments.
Enabling mechanised certification.

Practical Relevance

Algorithms and Data Structures

Jeffrey Vitter Editor

Skip Lists: A Probabilistic Alternative to Balanced Trees

Skip lists are data structures that use probabilistic balancing rather than strictly enforced balancing. As a result, the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees.

William Pugh

Binary trees can be used for representing abstract data types such as dictionaries and ordered lists. They work well when the elements are inserted in a random order. Some sequences of operations, such as inserting the elements in order, produce degenerate data structures that perform very poorly. If it were possible to randomly permute the list of items to be inserted, trees would work well with high probability for any input sequence. In most cases queries must be answered online, so randomly permuting the input is impractical.

self-adjusting tree algorithms. Skip lists are also very space efficient. They can easily be configured to require an average of 1 ⅓ pointers per element (or even less) and do not require balance or priority information to be stored with each node.

SKIP LISTS

We might need to examine every node of the list when searching a linked list (Figure 1a). If the list is stored in sorted order and every other node of the list also has

[Pugh, CACM 1989]

Practical Relevance

Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware

Michael Carbin Sasa Misailovic Martin C. Rinard
MIT CSAIL
{mcarbin, misailo, rinard}@csail.mit.edu

Abstract

Emerging high-performance architectures are anticipated to contain unreliable components that may exhibit *soft errors*, which silently corrupt the results of computations. Full detection and masking of soft errors is challenging, expensive, and, for some applications, unnecessary. For example, approximate computing applications (such as multimedia processing, machine learning, and big data analytics) can often naturally tolerate soft errors.

We present Rely, a programming language that enables developers to reason about the quantitative reliability of an application – namely, the probability that it produces the correct result when executed on unreliable hardware. Rely allows developers to specify the reliability requirements for each value that a function produces.

We present a static quantitative reliability analysis that

1. Introduction

System reliability is a major challenge in the design of emerging architectures. Energy efficiency and circuit scaling are becoming major goals when designing new devices. However, aggressively pursuing these design goals can often increase the frequency of *soft errors* in small [67] and large systems [10] alike. Researchers have developed numerous techniques for detecting and masking soft errors in both hardware [23] and software [20, 53, 57, 64]. These techniques typically come at the price of increased execution time, increased energy consumption, or both.

Many computations, however, can tolerate occasional unmasked errors. An *approximate computation* (including many multimedia, financial, machine learning, and big data analytics applications) can often acceptably tolerate occasional errors in its execution and/or the data that it manip-

[Carbin et al., CACM 2016]

Practical Relevance

Decision–Making with Complex Data Structures using Probabilistic Programming

Brian E. Ruttenberg and **Avi Pfeffer**
Charles River Analytics
625 Mt Auburn St.
Cambridge MA 02140
{bruttenberg, apfeffer}@cra.com

Abstract

Existing decision–theoretic reasoning frameworks such as decision networks use simple data structures and processes. However, decisions are often made based on complex data structures, such as social networks and protein sequences, and rich processes involving those structures. We present a framework for representing decision problems with complex data structures using probabilistic programming, allowing probabilistic models to be created with programming language constructs such as data structures and control flow. We provide a way to use arbitrary data types with minimal effort from the user, and an approximate decision–making algorithm that is effective even when the information space is very large or infinite. Experimental results show our algorithm working on problems with very large information spaces.

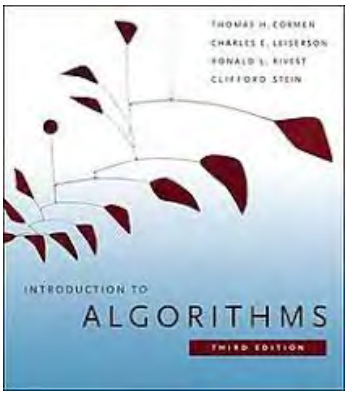
ditional probability tables. In our problem, however, the protein and DNA sequence data structures are complex, as are the processes by which proteins map to DNA and the rate of DNA mutation. This presents two challenges: first, how do we represent decision problems with complex data structures, and second, how do we reason with them to create a policy that recommends the best decisions?

We address these challenges using probabilistic programming, which provides the ability to create probabilistic models using programming language constructs such as data structures and control flow. Probabilistic programming languages contain general purpose reasoning algorithms that can reason on all models written in the language. Probabilistic programming languages can naturally be extended with constructs denoting decisions, similar to the way decision networks extend Bayesian networks. By providing a general–purpose decision–making algorithm, all the benefits

[Ruttenberg & Pfeffer, 2014]

Array Randomisation

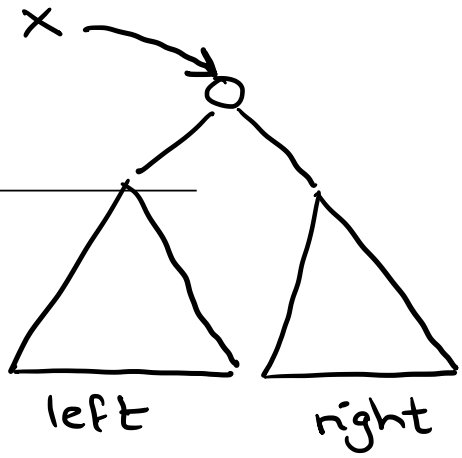
```
randomise(array,n) {
   i := 0;
   while (0 <= i < n) {
      j := uniform(i,n-1);
      swap(array,i,j);
      i++
  }
}
```



Is the probability of any fixed array configuration $\frac{1}{n!}$?

Faulty Garbage Collector

```
delete(x) {
   if (x != 0) {{
     skip // fails with probability p
   } [p] { // flip biased coin
     left := <x> ; right := <x+1>;
     delete(left) ;
     delete(right);
     free(x) ; free(x+1)
  }}
}
```

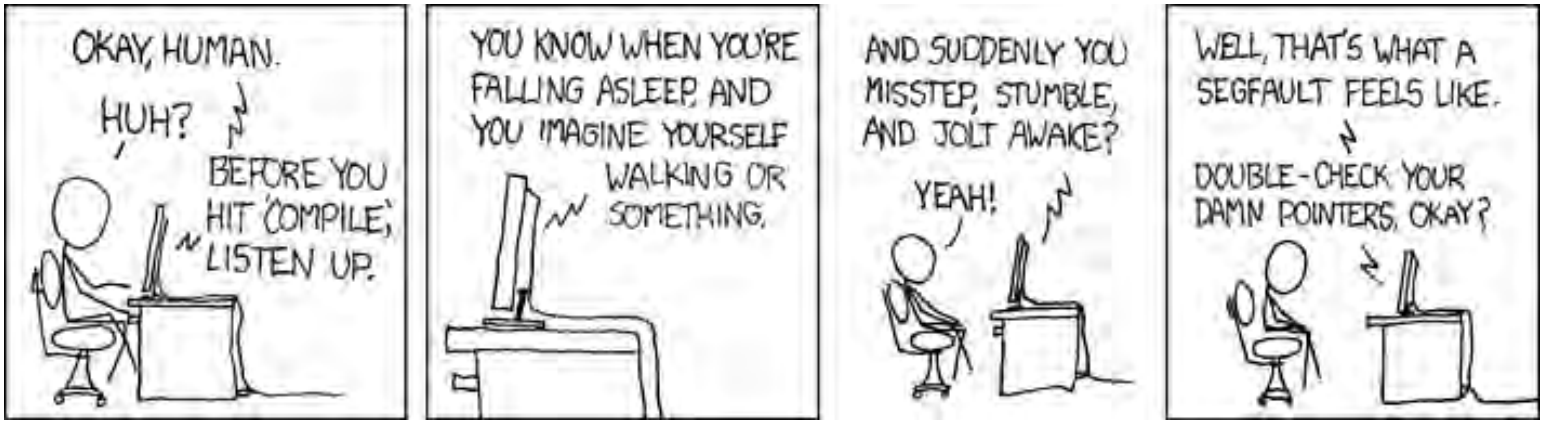


Faulty Garbage Collector

```
delete(x) {
   if (x != 0) {{
     skip // fails with probability p
   } [p] { // flip biased coin
     left := <x> ; right := <x+1>;
     delete(left) ;
     delete(right);
     free(x) ; free(x+1)
  }}
}
```

What is the probability that on termination the heap is empty?

Pointers = Problematic

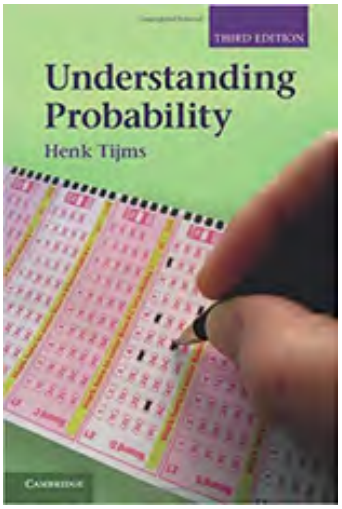


Dereferencing null pointers, aliasing, memory leaks, ...

Probabilities = Problematic Too

"In no other branch of mathematics
is it so easy to make mistakes
as in probability theory"

[Henk Tijms, Understanding Probability, 2004]



Mission Impossible? Not Quite!

We will develop a weakest precondition calculus à la Dijkstra that:

1. combines discrete probabilities with pointers

Mission Impossible? Not Quite!

We will develop a weakest precondition calculus à la Dijkstra that:

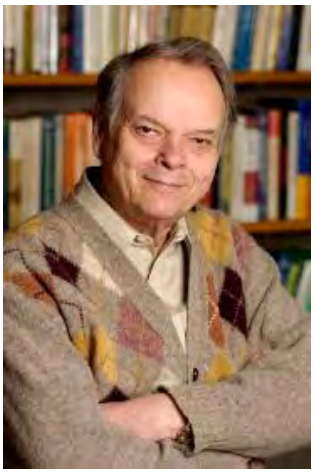
1. combines discrete probabilities with pointers
2. mixes probabilistic choices and unbounded nondeterminism
3. preserves virtually all properties of both:
   - separation logic, and
   - weakest pre-expectations (aka: quantitative preconditions)

   though things can easily break in both worlds ...
4. is applicable to reason about actual randomised algorithms

Overview

"The Hoare Logic for Pointers"

SEPARATION LOGIC

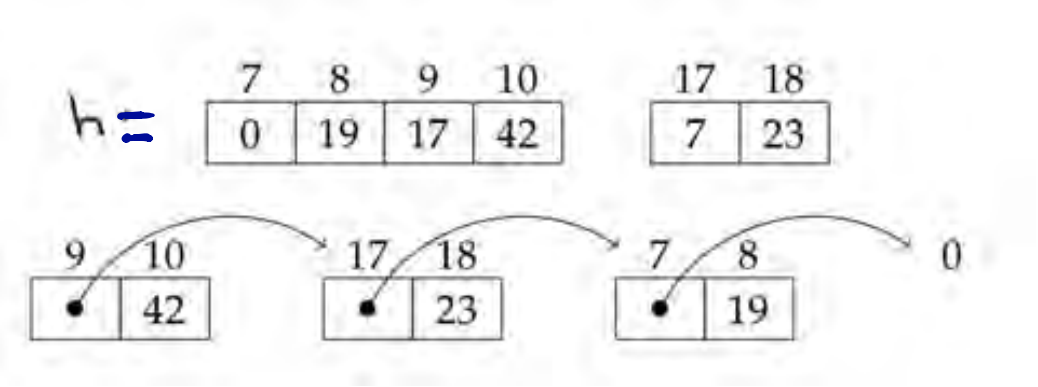


John Reynolds and Peter O'Hearn

Heaps

$$States = \mathbb{S} = \{(s, h) \mid \underbrace{s : Vars \to \mathbb{Z}}_{\text{valuation}}, \underbrace{h : \overbrace{\mathrm{dom}(h)}^{\subseteq \mathbb{N}_{>0}} \to \mathbb{Z}}_{\text{heap}}\}$$

h =



h =

Deficiency of Hoare Logic: Pointers

$$\frac{\{P\}\ C\ \{Q\} \quad \text{and} \quad \mathrm{Mod}(C) \cap \mathrm{Vars}(R) = \varnothing}{\{P \wedge R\}\ C\ \{Q \wedge R\}}$$

becomes unsound for pointers, e.g.,

$$\frac{\{x \mapsto 0\}\ \langle x \rangle := 1\ \{x \mapsto 1\}}{\{x \mapsto 0 \wedge y \mapsto 0\}\ \langle x \rangle := 1\ \{x \mapsto 1 \wedge y \mapsto 0\}}$$

is not valid as y could alias x

The Frame Rule

$$\frac{\{P\}\ C\ \{Q\} \quad \text{and} \quad \mathrm{Mod}(C) \cap \mathrm{Vars}(R) = \varnothing}{\{P \star R\}\ C\ \{Q \star R\}}$$

for any heap R that is unaffected by program C.

Then:

$$\frac{\{x \mapsto 0\}\ \langle x \rangle := 1\ \{x \mapsto 1\}}{\{x \mapsto 0 \star y \mapsto 0\}\ \langle x \rangle := 1\ \{x \mapsto 1 \star y \mapsto 0\}}$$

is valid as the separation conjunction excludes aliasing of x and y

The frame rule is the key to compositional reasoning.

Pointer Programs

Heap manipulation commands:

$x := \texttt{new}(E)$	allocation
$\texttt{free}(E)$	deallocation
$x := \langle E \rangle$	lookup
$\langle E \rangle := E'$	mutation

Operational semantics:

$$\frac{u \in \mathbb{N}_{>0} \setminus \mathrm{dom}(h) \quad \text{and} \quad E(s) = v}{\langle x := \texttt{new}(E), s, h \rangle \to \langle \mathrm{term}, s[x/u], h \uplus \{ u :: v \} \rangle}$$

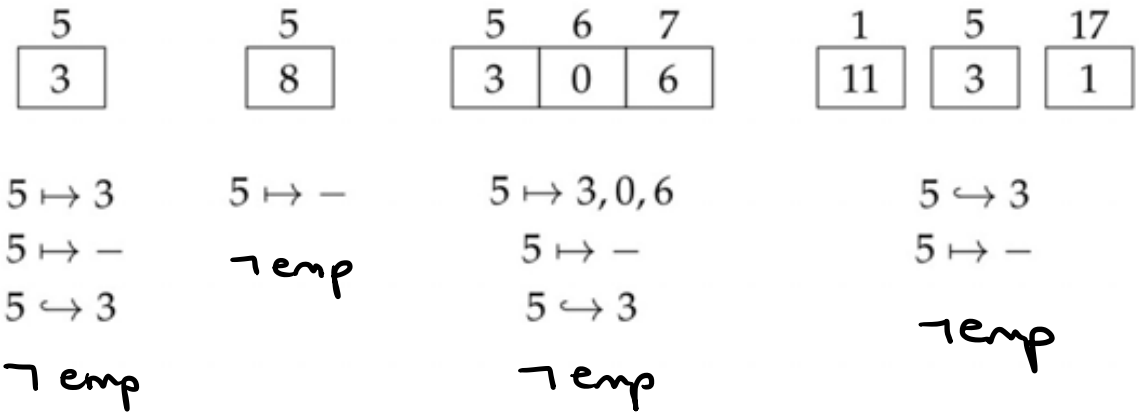
$$\frac{E(s) = u \in \mathrm{dom}(h) \quad \text{and} \quad h(u) = v}{\langle x := \langle E \rangle, s, h \rangle \to \langle \mathrm{term}, s[x/v], h \rangle}$$

$$\frac{E(s) = u \notin \mathrm{dom}(h) \quad \text{and} \quad h(u) = v}{\langle x := \langle E \rangle, s, h \rangle \to \langle \mathrm{fault}, s, h \rangle}$$

Elementary SL Formulas

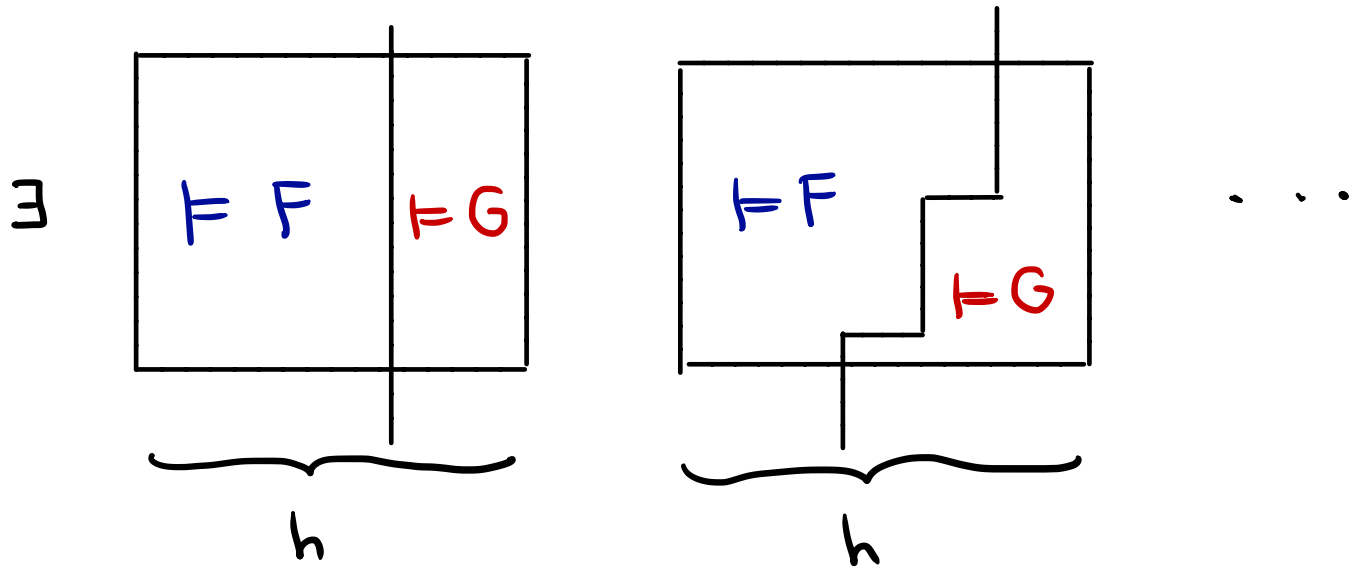
$(s, h) \vDash \text{emp}$ iff $\text{dom}(h) = \varnothing$

$(s, h) \vDash [x \mapsto y]$ iff $\text{dom}(h) = \{s(x)\}$ and $h(s(x)) = s(y)$



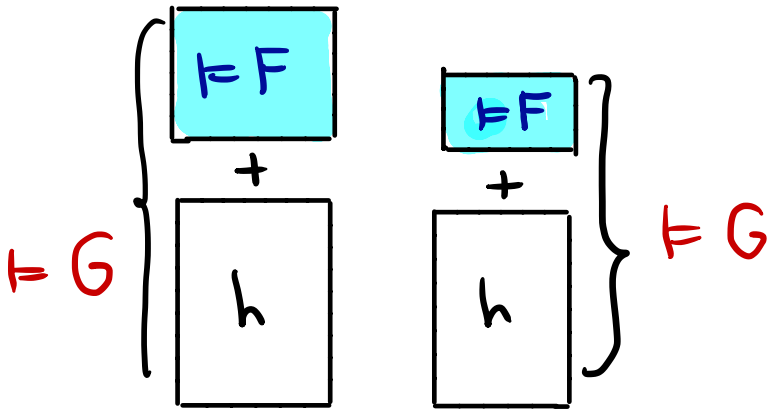
Separation Conjunction

$s, h \vDash F \star G$ iff $\exists h_1, h_2 : h = h_1 \uplus h_2$ and $s, h_1 \vDash F$ and $s, h_2 \vDash G$



Separation Implication

$$s, h \vDash F \mathrel{-\!\!\star} G \quad \text{iff} \quad \forall h' : \big(h \# h' \text{ and } s, h' \vDash F\big) \text{ implies } s, h \uplus h' \vDash G$$



Separation Implication

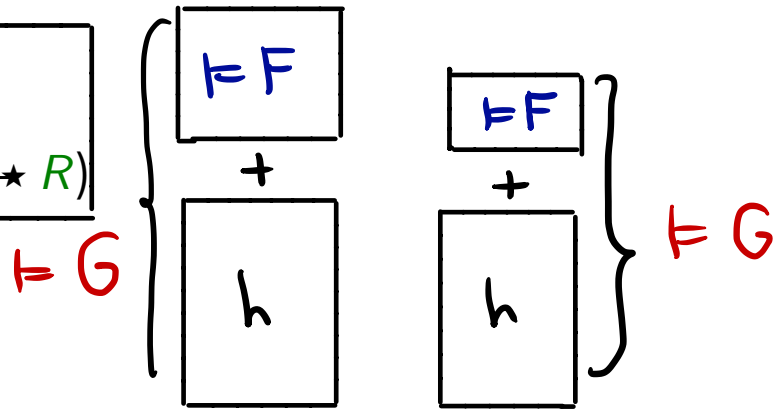
$$s, h \vDash F \mathrel{-\!\!\star} G \quad \text{iff} \quad \forall h' : \big(h \# h' \text{ and } s, h' \vDash F\big) \text{ implies } s, h \uplus h' \vDash G$$

Adjointness of $\star$ and $\mathrel{-\!\!\star}$:

$$(F \star G) \Rightarrow R \quad \text{iff} \quad G \Rightarrow (F \mathrel{-\!\!\star} R)$$

Modus ponens:

$$F \star (F \mathrel{-\!\!\star} G) \Rightarrow G$$



Example SL Proof

```
{tree(x)}
delete(*x) {
  if x = null then return  {emp}
  else {  {∃y, z : x ↦ (y, z) ∗ tree(y) ∗ tree(z)}
    l := x.left; r := x.right;
    {x ↦ (l, r) ∗ tree(l) ∗ tree(r)}
    delete(l);
    {x ↦ (l, r) ∗ emp ∗ tree(r)}
    delete(r);
    {x ↦ (l, r) ∗ emp ∗ emp}
    free(x); free(x + 1)
    {emp ∗ emp ∗ emp}
  }  {emp}
}  {emp}
```

Overview

"Dijkstra's Weakest Preconditions Go Random"

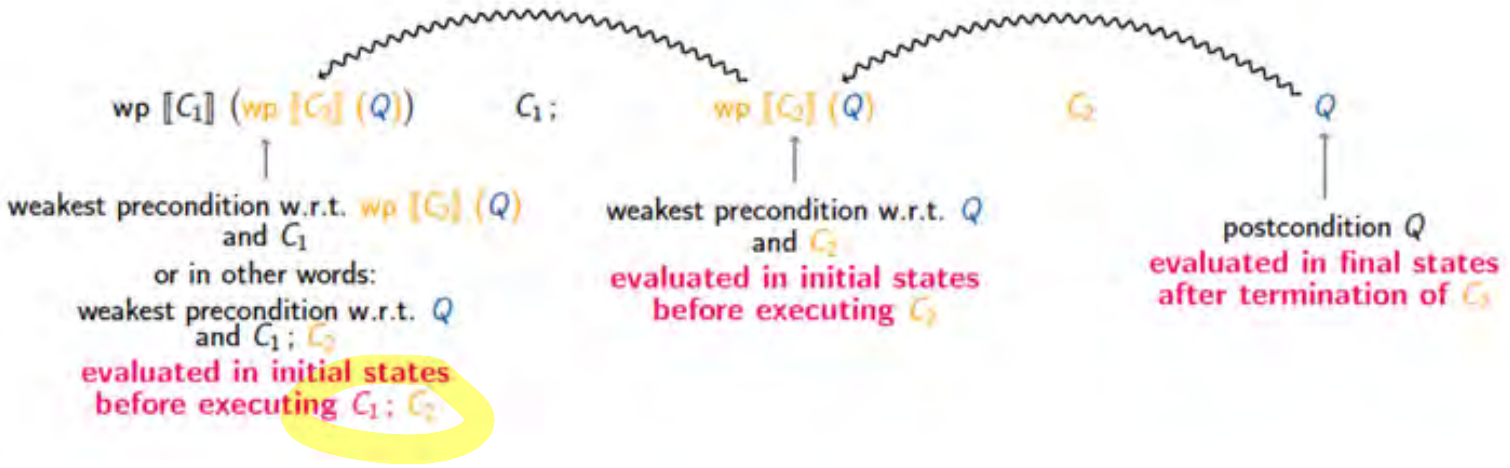
WEAKEST PRE-EXPECTATIONS



Dexter Kozen, Annabelle McIver, and Carroll Morgan

Weakest Precondition Reasoning

Use an inductively defined backwards moving predicate transformer $wp(C) : 2^{\mathbb{S}} \to 2^{\mathbb{S}}$:



From Predicates to Quantities

1. Let program C be:

   `x := 5 [4/5] x := 10`

   The expected value of `x` on C's termination is: $\frac{4}{5}\cdot 5 + \frac{1}{5}\cdot 10 = 6$

2. Let program C' be:

   `x := x+5 [4/5] x := 10`

   The expected value of `x` on C''s termination is: $\frac{4}{5}\cdot(x+5) + \frac{1}{5}\cdot 10 = \frac{4x}{5} + 6$

3. The probability that `x` = 10 on C''s termination is:

$$\frac{4}{5}\cdot[x+5 = 10] + \frac{1}{5}\cdot 1 = \frac{4\cdot[x = 5]+1}{5}$$

Expectations

Classical predicates:

A predicate F maps program states onto Booleans.

Let $F \leq G$ if and only if $F \Rightarrow G$.

Quantitative predicates: expectations:

An expectation[1] (aka: factor) f maps program states onto the non-negative reals extended with infinity.

Let $f \leq g$ if and only if $f(s) \leq g(s)$ for all states s.

The set of expectations under $\leq$ is a complete lattice.

[1] $\neq$ expectations in probability theory.

Expectation Transformers

Classical predicate transformer:

Maps predicates onto predicates.
Dijkstra's weakest preconditions are an instance of this.

$\frac{4}{5} \cdot [x=5] + \frac{1}{2} \cdot \ldots$

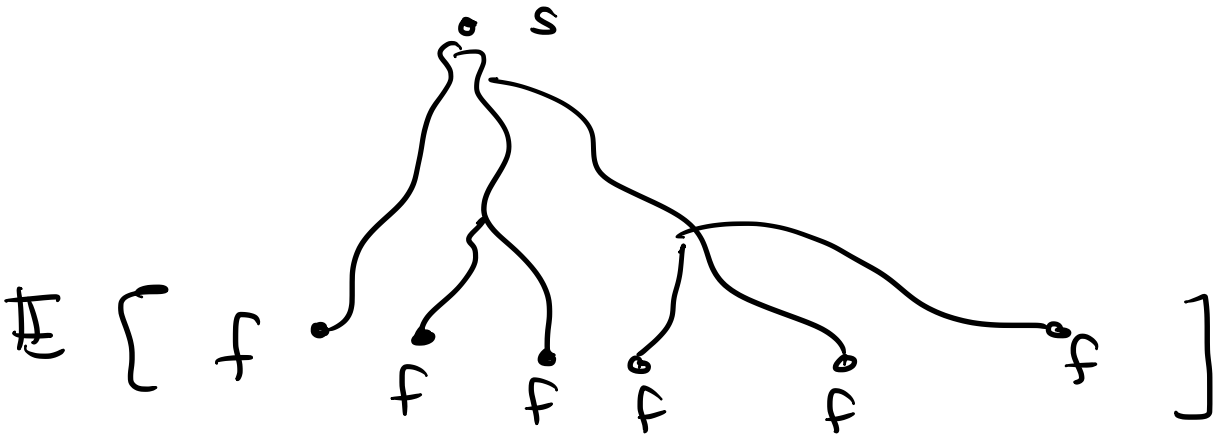
Expectation transformer:

Maps expectations onto expectations.
Characterising equation of a Kozen's weakest pre-expectation:

$$wp(C)(f) \;=\; \lambda s.\, \int_{\mathbb{S}} f \, dC_s$$

with C_s the distribution over C's final states when running C on state s

Pictorially



Quantitative Weakest Pre-condition Semantics

For expectation f, programs C, the $wp(C)(f)$ is defined by:

`skip`	f
$x := E$	$f[x/E]$
$C_1; C_2$	$wp(C_1)(wp(C_2)(f))$
`if` B $\{C_1\}$ `else` $\{C_2\}$	$[B] \cdot wp(C_1)(f) + [\neg B] \cdot wp(C_2)(f)$
$C_1 \, [p] \, C_2$	$p \cdot wp(C_1)(f) + (1-p) \cdot wp(C_2)(f)$
`while`$(B)\{C'\}$	$lfp\ X.\ \underbrace{[\neg B] \cdot f + [B] \cdot wp(C')(X)}_{\text{loop unrolling}}$

lfp is the least fixed point operator wrt. the ordering $\leq$ on expectations.

Extensions with recursion, conditioning, liberal wp, negative expectations.

Examples

1. Let program C be:

   ```
   x := 5 [4/5] x := 10
   ```

   For $f = x$, we have

   $$wp(C, x) = \frac{4}{5} \cdot wp(x := 5)(x) + \frac{1}{5} \cdot wp(x := 10)(x) = \frac{4}{5} \cdot 5 + \frac{1}{5} \cdot 10 = 6$$

2. Let program C' be:

   ```
   x := x+5 [4/5] x := 10
   ```

   For $f = x$, we have:

   $$wp(C')(x) = \frac{4}{5} \cdot wp(x := x+5)(x) + \frac{1}{5} \cdot wp(x := 10)(x) = \frac{4}{5} \cdot (x+5) + \frac{1}{5} \cdot 10 = \frac{4x}{5} + 6$$

3. For program C' (again) and $f = [x = 10]$, we have:

   $$\begin{aligned} wp(C')([x{=}10]) &= \frac{4}{5} \cdot wp(x := x+5)([x{=}10]) + \frac{1}{5} \cdot wp(x := 10)(x{=}10]) \\ &= \frac{4}{5} \cdot [x+5 = 10] + \frac{1}{5} \cdot [10 = 10] \\ &= \frac{4 \cdot [x{=}5] + 1}{5} \end{aligned}$$

Properties

For all programs C and expectations f, g it holds:

- Continuity: $wp(C)(\cdot)$ is continuous.
- Monotonicity: $f \leq g$ implies $wp(C)(f) \leq wp(P)(g)$
- Feasibility: $f \leq \mathbf{k}$ implies $wp(C)(f) \leq \mathbf{k}$
- Linearity: $wp(C)(r \cdot f + g) = r \cdot wp(C)(f) + wp(C)(g)$ for every $r \in \mathbb{R}_{\geq 0}$
- Strictness: $wp(C)(\mathbf{0}) = \mathbf{0}$

Good to know: $wp(C)(\mathbf{1})$ = termination probability of program C

Practical Relevance

- Formal verification of randomised algorithms
- Exact inference for Bayesian networks
- Deciding program equivalence
- Proving program transformations
- Expected resource consumption
- Proving almost-sure termination

Overview

- Assertion language
- wp-Calculus
- Theorems
- Case studies

Assertion Language: States and Expectations

$$States = \{(s,h) \mid s\colon Vars \to \mathbb{Z},\quad h\colon \underbrace{\mathrm{dom}\,(h)}_{\subseteq \mathbb{N}\setminus\{0\}\ \text{finite}} \to \mathbb{Z}\}$$

Expectations: $f : States \to \mathbb{R}^{\infty}_{\geq 0}$

Examples: $x^2 = \lambda(s,h).\ s(x)^2$ $\qquad$ $\mathsf{size} = \lambda(s,h).\ |\mathrm{dom}\,(h)|$

$$[\mathsf{emp}] = \lambda(s,h).\ \begin{cases} 1 & \text{if } \mathrm{dom}\,(h) = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

$$[x \mapsto y] = \lambda(s,h).\ \begin{cases} 1 & \text{if } \mathrm{dom}\,(h) = \{s(x)\}, h(s(x)) = s(y)\} \\ 0 & \text{otherwise} \end{cases}$$

Example QSL Specifications

Postexpectation f	Weakest pre-expectation $wp(C)(f)$
$\lambda(s, h).\, 1$	Probability of memory-safe termination
$\lvert x \rvert$	Expected absolute value of x
$[\mathbf{emp}]$	Probability of termination with an empty heap
$\mathtt{len}(x, y)$	Expected length of list segment from x to y

Quantitative Conjunction

Classical conjunction

$$F \wedge G$$

Quantitative conjunction

$$f \cdot g \;=\; \lambda(s,h).\, f(s,h) \cdot g(s,h)$$

Note:

$$[F \wedge G] \;=\; [F] \cdot [G]$$

Quantitative ~~Conjunction~~

Dis

dis

Classical ~~conjunction~~

$F \not\wedge G$

$\vee$

Quantitative conjunction

$$f \cdot g \;=\; \lambda(s,h).\; \cancel{f(s,h) \cdot g(s,h)}$$

$$\max\{\, f(s,h),\, g(s,h) \,\}$$

Note:

$$[F \not\wedge G] \;=\; \cancel{[F]\ [G]}$$

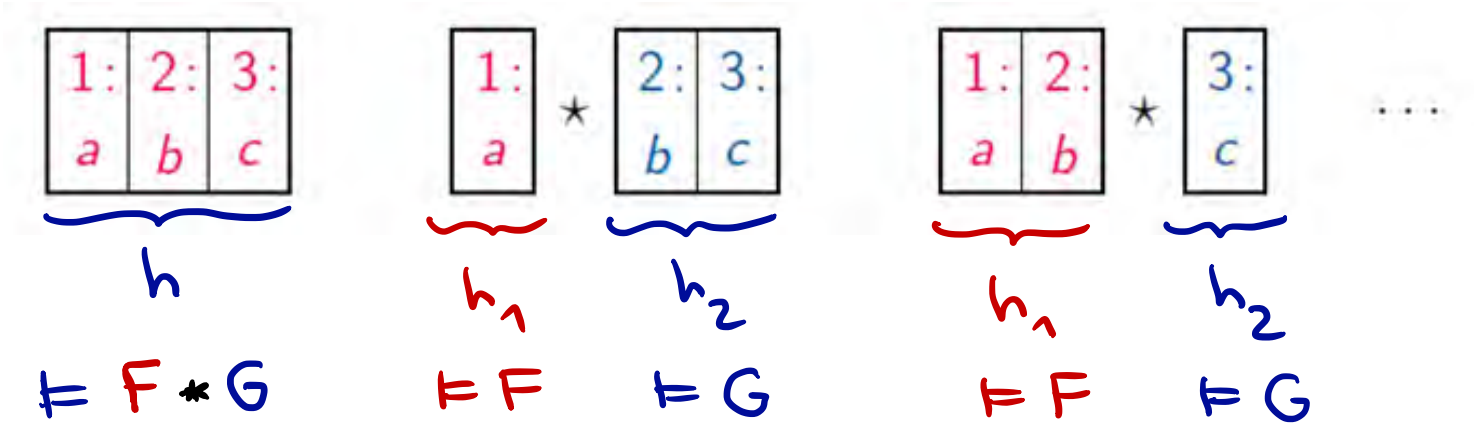
$\vee$

$$\max\{\, [F],\, [G] \,\}$$

Separating Conjunction

Classical separation conjunction:

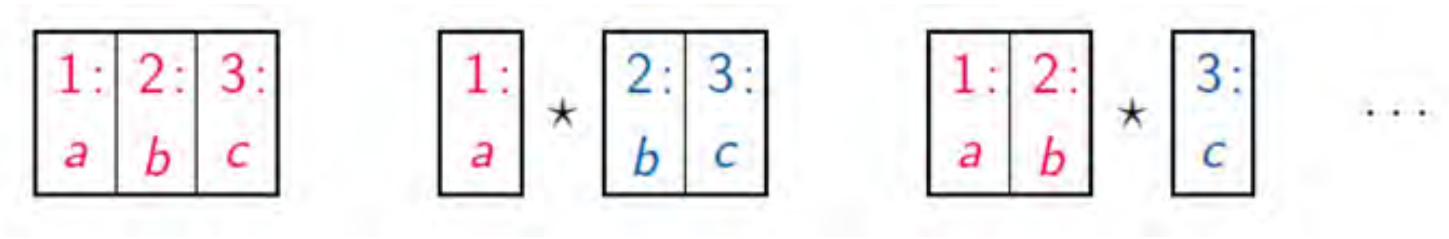
$(s, h) \models F \star G$ iff $\exists h_1, h_2.\ h = h_1 \star h_2$ and $(s, h_1) \models F$ and $(s, h_2) \models G$



Separating Conjunction

Classical separation conjunction:

$(s, h) \models F \star G$ iff $\exists h_1, h_2.\, h = h_1 \star h_2$ and $(s, h_1) \models F$ and $(s, h_2) \models G$



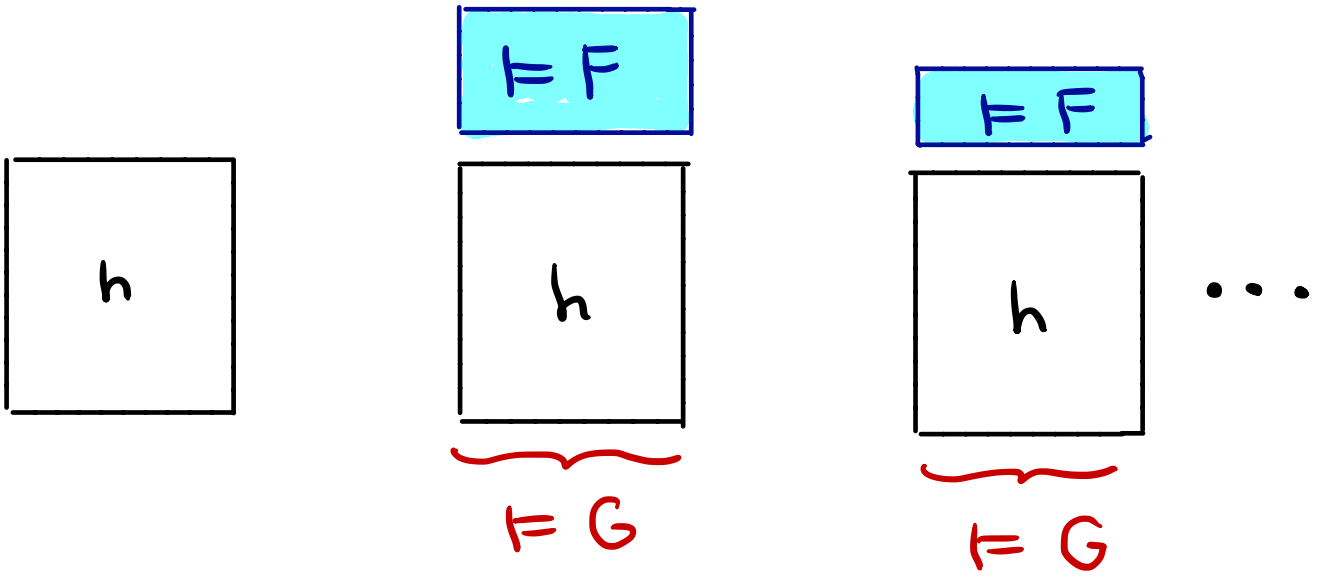
Quantitative separation conjunction:

$$f \star g \quad = \quad \lambda(s, h).\ \sup \{\, f(s, h_1) \star g(s, h_2) \mid h = h_1 \star h_2 \,\}$$

Separating Implication

Classical separation implication:

$(s, h) \models F \mathrel{-\!\!\star} G$ iff $\forall h'$ with $h' \# h$ and $(s, h') \models F$: $(s, h \star h') \models G$



[2] and not $f(s, h') = \infty = g(s, h \star h')$

Separating Implication

Classical separation implication:

$$(s,h) \models F \mathrel{-\!\!\star} G \quad \text{iff} \quad \forall h' \text{ with } h' \# h \text{ and } (s,h') \models F : \quad (s, h \star h') \models G$$

Quantitative separation implication:

$$f \mathrel{-\!\!\star} g \quad = \quad \lambda(s,h).\ \inf \left\{ \frac{g(s,h \star h')}{f(s,h')} \mid h' \# h \text{ and } f(s,h') > 0 \right\}^2$$

Note that:

$$[F] \mathrel{-\!\!\star} g \quad = \quad \lambda(s,h).\ \inf \left\{ g(s,h \star h') \mid h' \# h \text{ and } (s,h') \models F \right\}$$

[2] and not $f(s,h') = \infty = g(s,h \star h')$

Adjointness

Classical adjointness:

$$(F \star G) \Rightarrow J \quad \text{iff} \quad (F \Rightarrow (G \mathrel{-\!\!\star} J))$$

Quantitative adjointness:

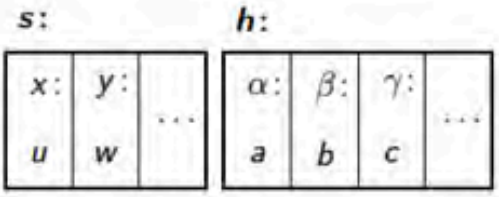
$$(f \star g) \leq j \quad \text{iff} \quad (f \leq (g \mathrel{-\!\!\star} j))$$

Calculus:

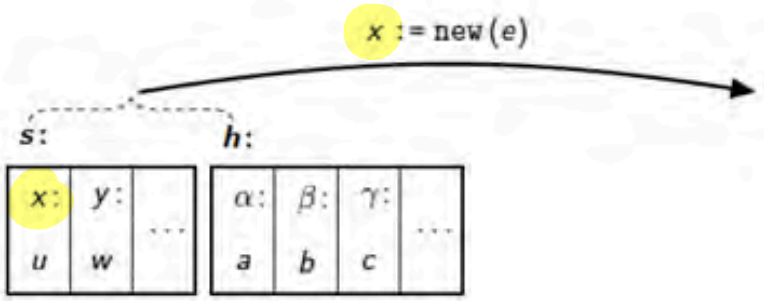
$$a - b \leq c \quad \text{iff} \quad a \leq b + c$$

+ virtually all properties in [Ishtiaq & O'Hearn 2001, Reynolds 2002]

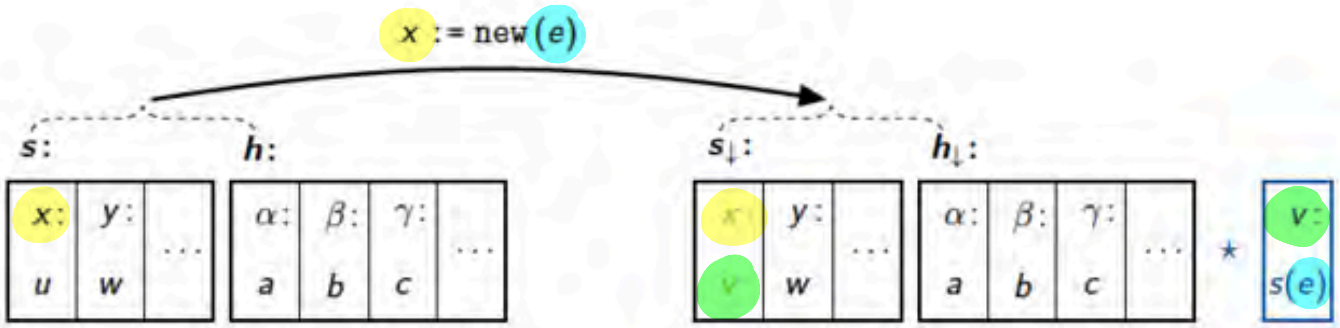
Probabilistic wp for Memory Allocation



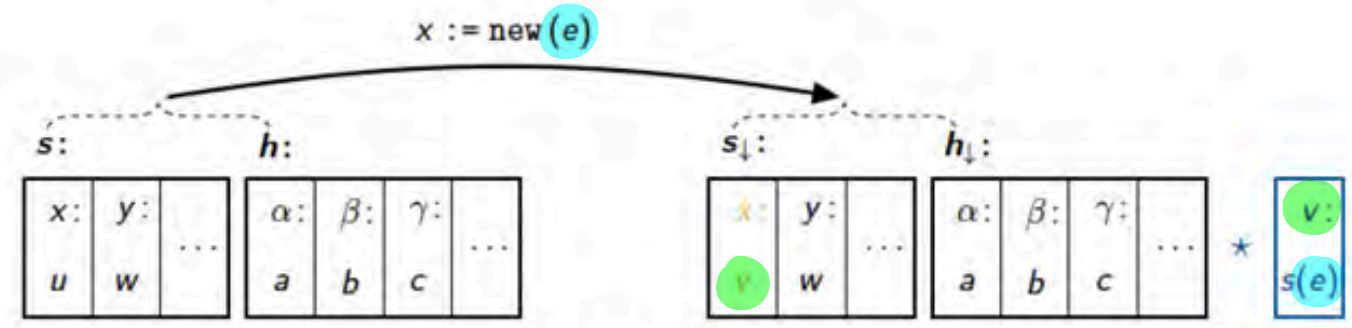
Probabilistic wp for Memory Allocation

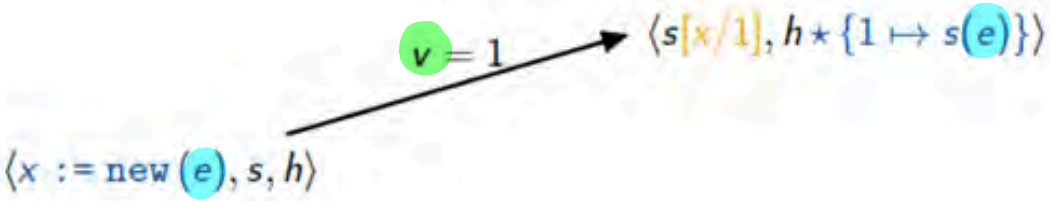


Probabilistic wp for Memory Allocation

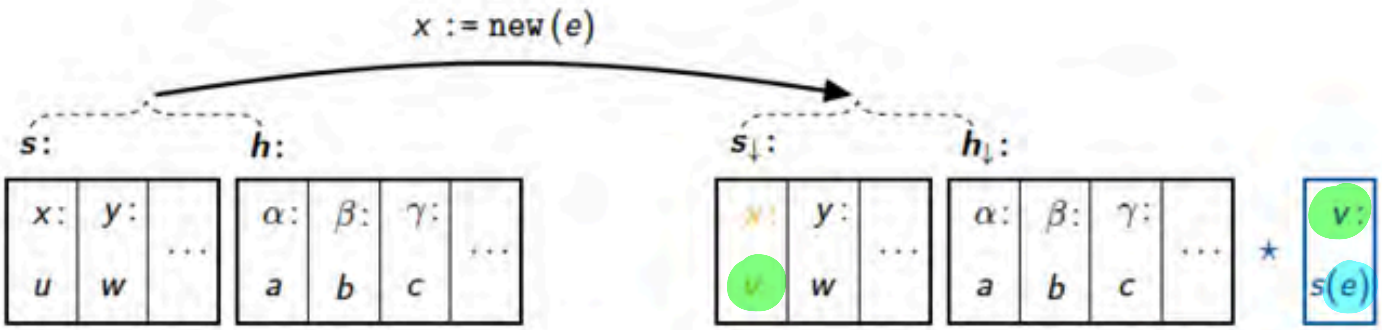


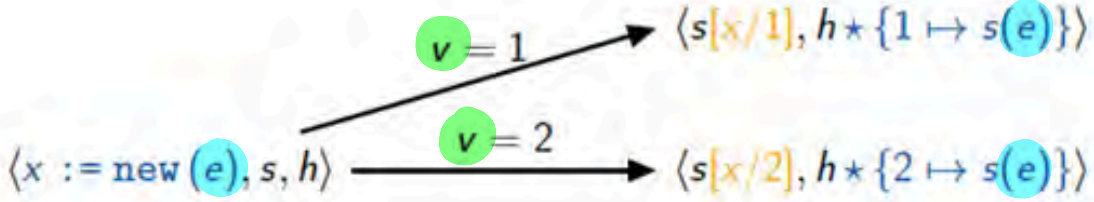
Probabilistic wp for Memory Allocation



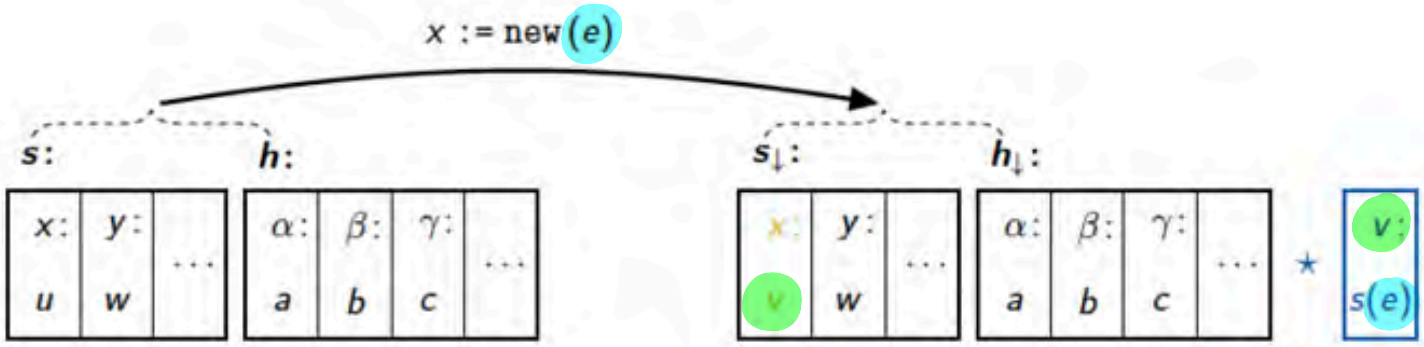


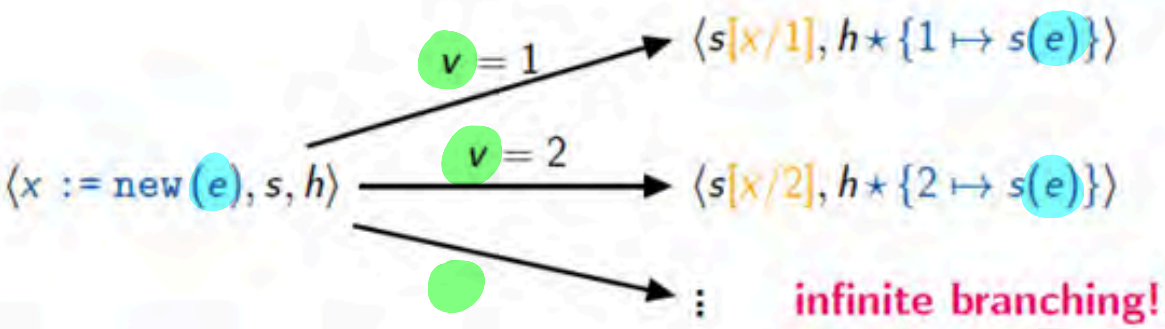
Probabilistic wp for Memory Allocation



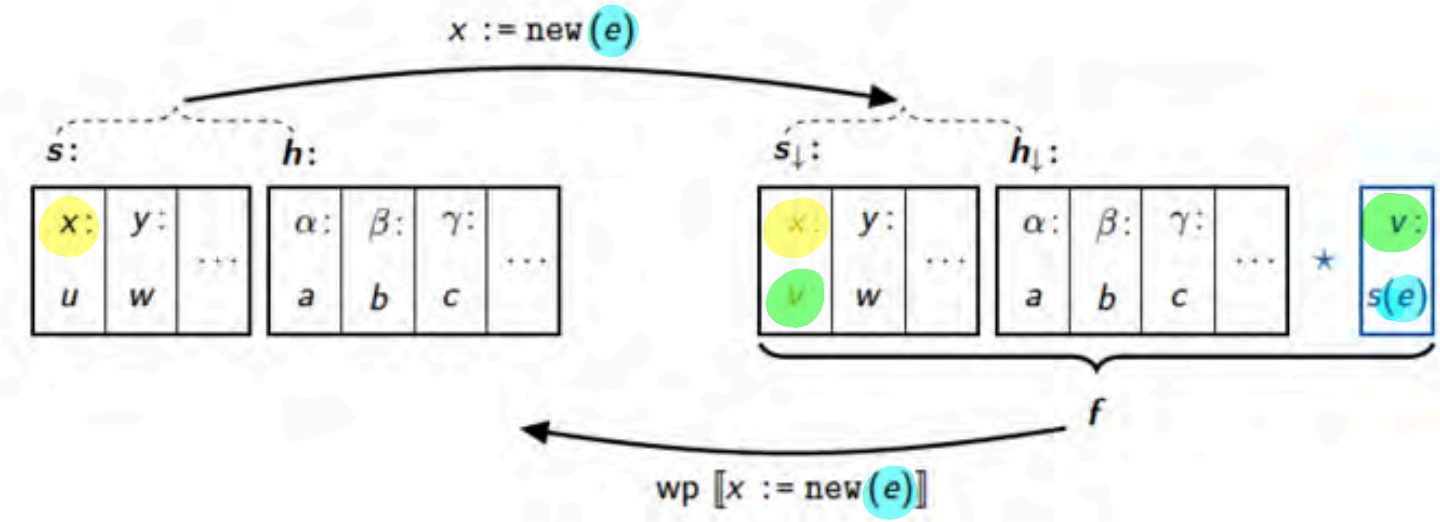


Probabilistic wp for Memory Allocation





Probabilistic wp for Memory Allocation



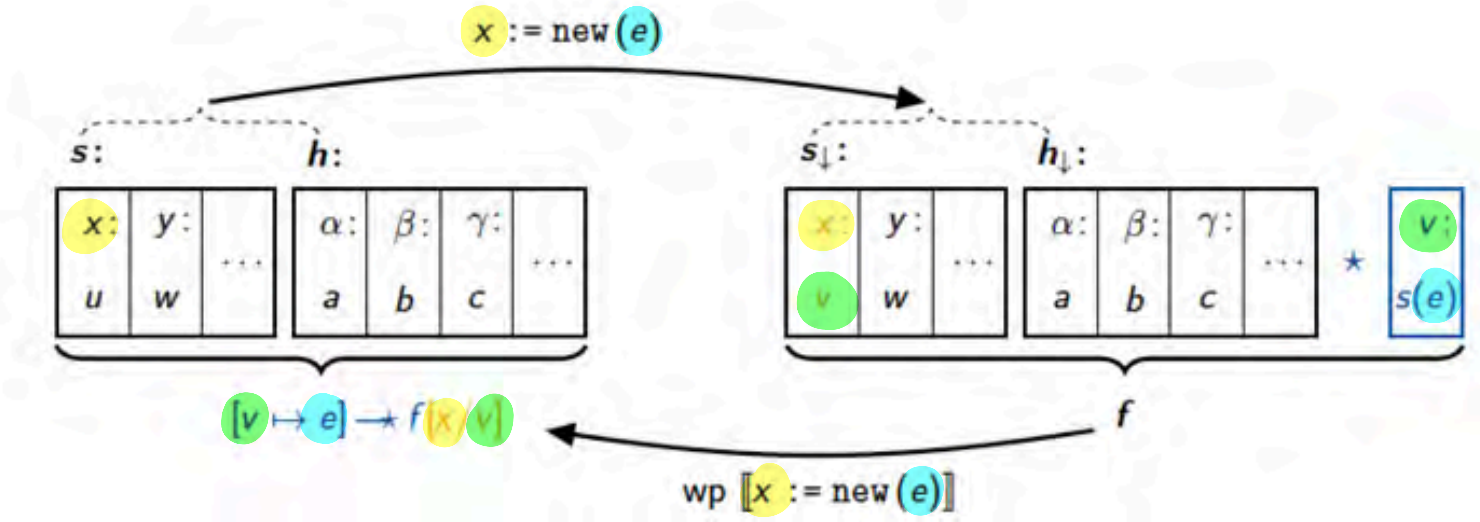
$$\langle x := \texttt{new}(e), s, h\rangle \xrightarrow{v=1} \langle s[x/1], h \star \{1 \mapsto s(e)\}\rangle$$

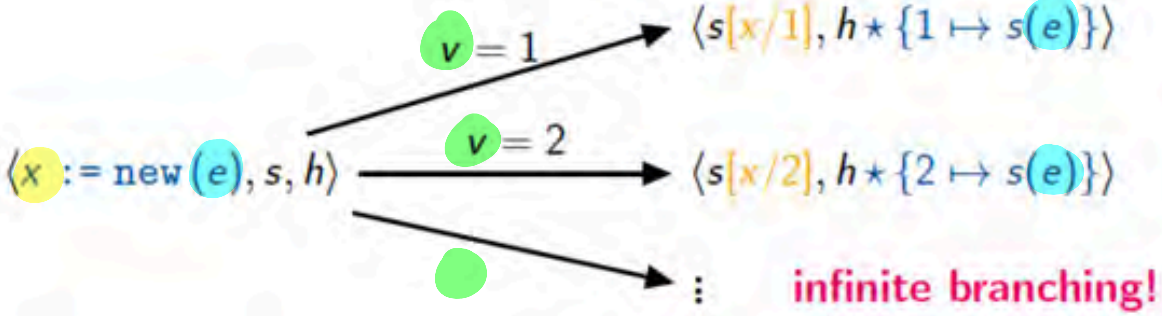
$$\langle x := \texttt{new}(e), s, h\rangle \xrightarrow{v=2} \langle s[x/2], h \star \{2 \mapsto s(e)\}\rangle$$

$$\langle x := \texttt{new}(e), s, h\rangle \xrightarrow{\quad} \vdots$$

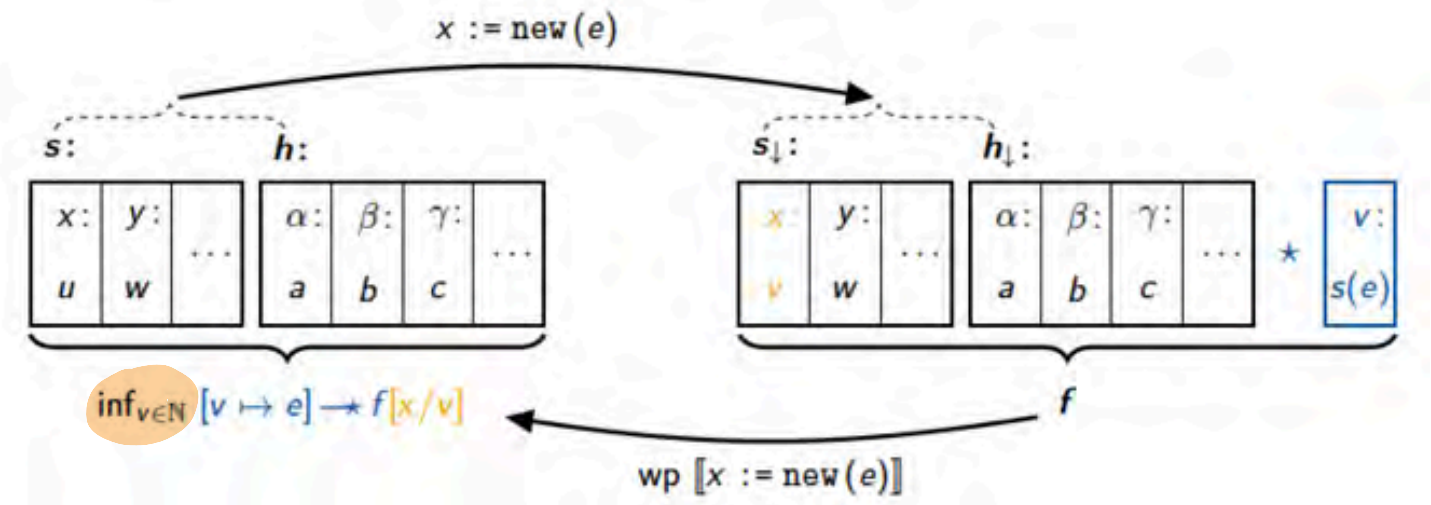
infinite branching!

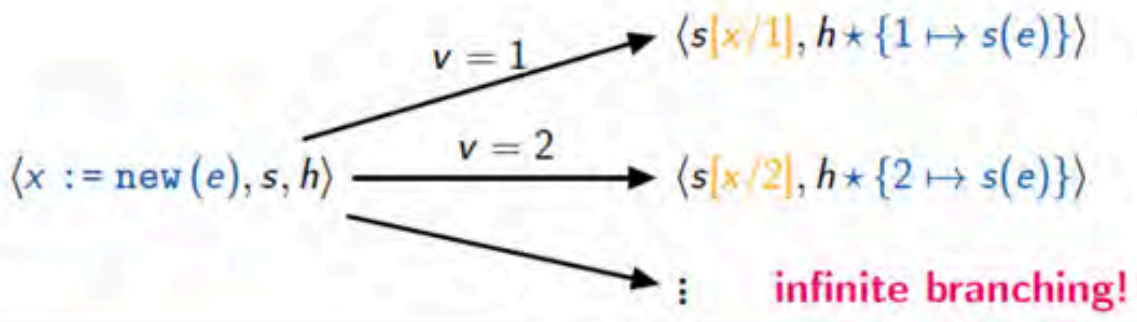
Probabilistic wp for Memory Allocation





Probabilistic wp for Memory Allocation





QSL's Weakest Pre-expectation Calculus

For expectation f in QSL, program C, the $wp(C)(f)$ is defined by:

`skip`	f
$x := E$	$f[x/E]$
$C_1; C_2$	$wp(C_1)(wp(C_2)(f))$
`if` B $\{C_1\}$ `else` $\{C_2\}$	$[B] \cdot wp(C_1)(f) + [\neg B] \cdot wp(C_2)(f)$
`while`$(B)\{C'\}$	$lfp\ X.\, [\neg B] \cdot f + [B] \cdot wp(C')(X)$
$C_1\,[p]\,C_2$	$p \cdot wp(C_1)(f) + (1-p) \cdot wp(C_2)(f)$
$x :=$ `new`(E)	$\inf_{v \in \mathbb{N}} [v \mapsto E] \mathrel{-\!\!\star} f[x/v]$
`free`(E)	$[E \mapsto -] \star f$
$x := \langle E \rangle$	$\sup_{v \in \mathbb{Z}} [E \mapsto v] \star ([E \mapsto v] \mathrel{-\!\!\star} f[x/v])$
$\langle E \rangle := E'$	$[E \mapsto -] \star ([E \mapsto E'] \mathrel{-\!\!\star} f)$

Heap Manipulation: SL versus QSL

For predicate F in SL, $wp(C)(F)$ is:

$x := \texttt{new}(E)$	$\forall v.\, v \mapsto E \mathrel{-\!\!\star} F[x/v]$
$\texttt{free}(E)$	$E \mapsto - \star F$
$x := \langle E \rangle$	$\exists v.\, E \mapsto v \star (E \mapsto v \mathrel{-\!\!\star} F[x/v])$
$\langle E \rangle := E'$	$E \mapsto - \star \big(E \mapsto E' \mathrel{-\!\!\star} F\big)$

For expectation f in QSL, $wp(C)(f)$ is:

$x := \texttt{new}(E)$	$\inf_{v \in \mathbb{N}} [v \mapsto E] \mathrel{-\!\!\star} f[x/v]$
$\texttt{free}(E)$	$[E \mapsto -] \star f$
$x := \langle E \rangle$	$\sup_{v \in \mathbb{Z}} [E \mapsto v] \star ([E \mapsto v] \mathrel{-\!\!\star} f[x/v])$
$\langle E \rangle := E'$	$[E \mapsto -] \star \big([E \mapsto E'] \mathrel{-\!\!\star} f\big)$

A Small Example

$$
\begin{array}{l}
/\!/\ \ \mathsf{len}(x) + 1 \\
c := 1 \,; \\
/\!/\ \ \mathsf{len}(x) + [c = 1] \\
\texttt{while}\,(\,c = 1\,)\,\{ \\
\quad /\!/\ \ \mathsf{len}(x) + \tfrac{1}{2}(1 + [c = 1]) \\
\quad /\!/\ \ \tfrac{1}{2}\Big(\mathsf{len}(x) + \inf_{v}\, [v \mapsto x] \mathrel{-\!\!*} (\mathsf{len}(x) + [c = 1])\Big) \\
\quad \{\, c := 0 \,\}\ [1/2]\ \{\, x := \texttt{new}\,(x) \,\} \\
\quad /\!/\ \ \mathsf{len}(x) + [c = 1] \\
\} \\
/\!/\ \ \mathsf{len}(x)
\end{array}
$$

Theorem 1: Conservativity

QSL conservatively extends both SL and weakest preexpectations.

Let F, G be SL formulas and $[\![F]\!]$ and $[\![G]\!]$ be their corresponding expectations.

For all states (s, h) and all non-randomised pointer programs:

$$(s, h) \vDash F \quad \text{if and only if} \quad [\![F]\!](s, h) = 1$$

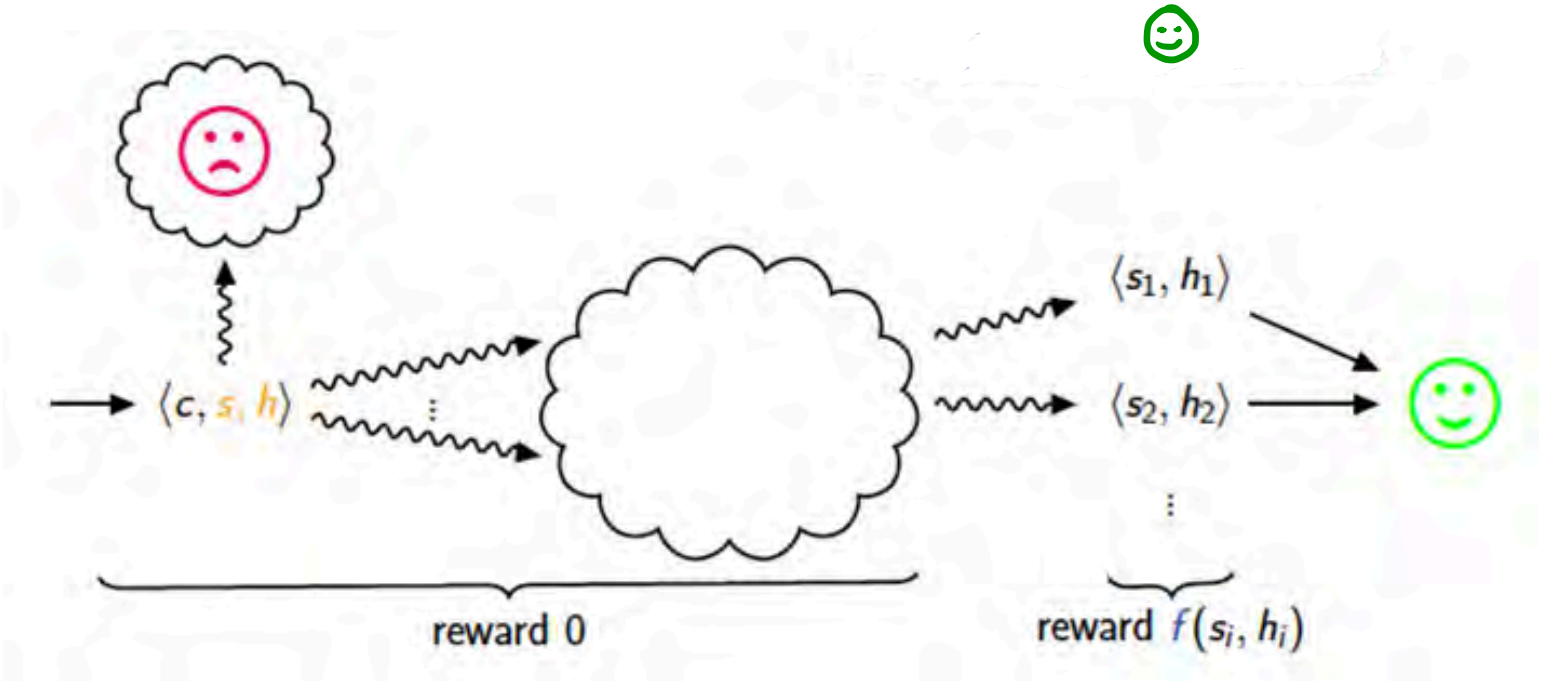
$$\{F\}\, C\, \{G\} \quad \text{if and only if} \quad [\![F]\!] \leq wp(C)([\![G]\!])$$

Theorem 2: Soundness

QSL's wp is equivalent to a simple operational model.

For all programs C, expectation f and state (s, h):

$wp(C)(f)(s, h)$ = expected reward w.r.t. f to reach success in MDP of C



Theorem 3: The Quantitative Frame Rule

The classical frame rule:

For all SL formulas F, G and R with $\mathrm{Mod}(C) \cap \mathrm{Vars}(R) = \varnothing$:

$$\frac{\{F\}\, C\, \{G\}}{\{F \star R\}\, C\, \{G \star R\}}$$

For $F = wp(C)(G)$, this reduces to: $wp(C)(G) \star R \;\Rightarrow\; wp(C)(G \star R)$

The quantitative frame rule:

For all expectations g, r with $\mathrm{Mod}(C) \cap \mathrm{Vars}(r) = \varnothing$ it holds:

$$wp(C)(g) \star r \;\leq\; wp(C)(g \star r)$$

Remark

This variant of the frame rule is **unsound**:

$$wp(C)(g) \star r \quad \geq \quad wp(C)(g \star r)$$

Counterexample with $\mathrm{Mod}(\langle x \rangle := 0) \cap [x \hookrightarrow 0] = \varnothing$:

$$wp(\langle x \rangle := 0)(\underbrace{[\mathbf{emp}]}_{g}) \star \underbrace{[x \hookrightarrow 0]}_{r} \quad \not\geq \quad wp(\langle x \rangle := 0)(\underbrace{[\mathbf{emp}] \star [x \hookrightarrow 0]}_{g \star r})$$

Since:

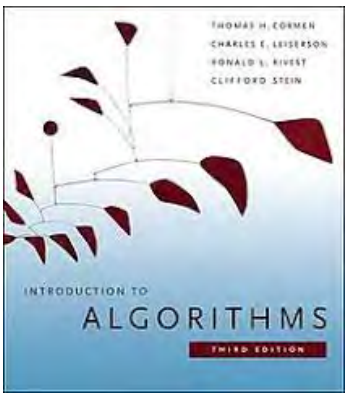
$$wp(\langle x \rangle := 0)([\mathbf{emp}]) \quad = \quad \underbrace{[x \mapsto -] \star ([x \mapsto 0] \mathrel{-\!\!\star} [\mathbf{emp}])}_{=0}$$

$$wp(\langle x \rangle := 0)([\mathbf{emp}] \star [x \hookrightarrow 0]) \quad = \quad \underbrace{[x \mapsto -] \star ([x \mapsto 0] \mathrel{-\!\!\star} ([\mathbf{emp}] \star [x \hookrightarrow 0])}_{=[x \hookrightarrow 0]}$$

Overview

Example 1: Array Randomisation

```
randomise(array,n) {
   i := 0;
   while (0 <= i < n) {
      j := uniform(i,n-1);
      swap(array,i,j);
      i++
  }
}
```



The probability of any fixed array configuration is $\frac{1}{n!}$

$$wp(\texttt{randomise}(\texttt{array}, n))(\texttt{array} \mapsto \alpha_0, \ldots, \alpha_{n-1}) = \frac{1}{n!}$$

Example 2: Faulty Garbage Collector

```
delete (x) {
   if (x != 0) {{
     skip // fails with probability p
   } [p] { // flip biased coin
     left := <x> ; right := <x+1>;
     delete(left) ;
     delete(right);
     free(x) ; free(x+1)
  }}
}
```

The probability of deleting a tree with root x is at least $(1-p)^{\text{size of heap}}$

$$wp(\texttt{delete}(x))([\mathbf{emp}]) \;\geq\; [\text{tree}(x)] \cdot (1-p)^{\mathbf{size}}$$

Proof Snapshot

```
/// (1 − p)^spt(x)
delete (x)
    if (x ≠ 0) {{
        skip
    } [p] {
        l := x.1 ; r := x + 1.1 ;
        /// [x ↦ −, −] ⋆ (1 − p)^spt(l) ⋆ (1 − p)^spt(r)        (recursion+frame rule)
        delete (l) ;
        /// [x ↦ −, −] ⋆ (1 − p)^spt(r)                        (recursion+frame rule)
        delete (r) ;
        /// [x ↦ −, −] ⋆ [emp]
        free(x) ; free(x + 1)
}}}
/// [emp]
```

Example 3: Lossy List Reversal

```
lossyReversal (hd) {
   r := 0;
   while (hd != 0) {
       t := <hd>;
       { <hd> := r; r := hd}
       [0.5]
       { free(hd) };
       hd := t
   }
}
```

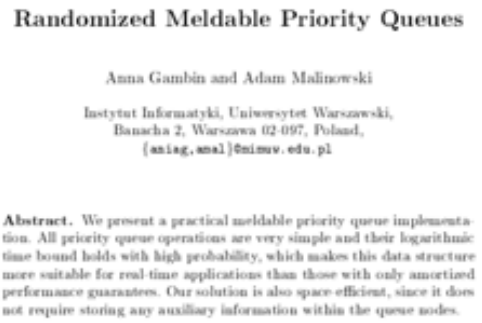
In expectation, the length of the reversed list r is at most half the length of the input list

$$wp(\texttt{lossyReversal}(hd))(\textbf{len}(\mathrm{r}, 0)) \;\leq\; \tfrac{1}{2} \cdot [hd \neq 0] \cdot \textbf{len}(hd, 0)$$

Example 4: Randomised Meldable Heaps

A more formal proof of a less simple randomised data structure

```
randomLeaf (root) {
   nextL := <root>;
   nextR := <root+1>;
   if (nextL = 0 and nextR = 0) {
        return root
   } else {
        { next := nextL }
        [0.5]
        { next := nextR };
        return randomLeaf (root)
   }
}
```



Randomized Meldable Priority Queues

Anna Gambin and Adam Malinowski

Instytut Informatyki, Uniwersytet Warszawski,
Banacha 2, Warszawa 02-097, Poland,
{aniag,amal}@mimuw.edu.pl

Abstract. We present a practical meldable priority queue implementation. All priority queue operations are very simple and their logarithmic time bound holds with high probability, which makes this data structure more suitable for real-time applications than those with only amortized performance guarantees. Our solution is also space efficient, since it does not require storing any auxiliary information within the queue nodes.

[Gambin and Malinowski, 1998]

A Textbook Proof

Lemma 10.1. *The expected length of a random walk in a binary tree with* n *nodes is at most* $\log(n+1)$.

Proof. The proof is by induction on n. In the base case, $n = 0$ and the walk has length $0 = \log(n+1)$. Suppose now that the result is true for all non-negative integers $n' < n$.

Let n_1 denote the size of the root's left subtree, so that $n_2 = n - n_1 - 1$ is the size of the root's right subtree. Starting at the root, the walk takes one step and then continues in a subtree of size n_1 or n_2. By our inductive hypothesis, the expected length of the walk is then

$$\mathrm{E}[W] = 1 + \frac{1}{2}\log(n_1+1) + \frac{1}{2}\log(n_2+1) \ ,$$

since each of n_1 and n_2 are less than n. Since log is a concave function, $\mathrm{E}[W]$ is maximized when $n_1 = n_2 = (n-1)/2$. Therefore, the expected number of steps taken by the random walk is

$$\begin{aligned}\mathrm{E}[W] &= 1 + \frac{1}{2}\log(n_1+1) + \frac{1}{2}\log(n_2+1) \\ &\le 1 + \log((n-1)/2+1) \\ &= 1 + \log((n+1)/2) \\ &= \log(n+1) \ . \end{aligned}$$

□

Overview

Mechanising QSL in 

- The assertion language of QSL has been certified in Isabelle/HOL[3]
  - quantitative separating connectives for general expectations
  - algebraic properties (e.g., adjointness, modus ponens, monotonicity)
  - conservative extension: embedding of SL into QSL
  - .....
  - ≈ 2,000 lines of Isabelle/HOL code, a couple of man-months
- No errors were found; proofs could almost be taken one-to-one
- QSL's weakest pre-expectations have not been mechanised
  - main challenge: unbounded nondeterminism (aka: memory allocation)

[3] courtesy Max Haslbeck, `https://github.com/maxhaslbe` ... `antSepCon`

Separation Logic goes Random

1. Combines discrete probabilities with pointers
2. Mixes probabilistic choices and unbounded nondeterminism
3. Preserves virtually all properties of both:
   - Reynolds and O'Hearn's separation logic, and
   - Kozen, McIver and Morgan's weakest pre-expectations
4. Elementary properties certified in Isabelle/HOL
5. Applicable to reason about actual randomised algorithms
   - Next: verifying (dynamic) probabilistic graphical models
6. Future: quantitative symbolic heaps, entailment, concurrency, ...

A big thanks to my co-authors!

Hannah Arndt, Kevin Batz, Benjamin Kaminski,

Christoph Matheja, Thomas Noll

Further details: POPL 2019 and full version on `arxiv`