CFP: Extracting total Amb programs from proofs

Hideki Tsuiki
Kyoto University

j.w.w. Ulrich Berger

Autumn School Proof and Computation 2025 (2)
2025/9/16
Herrsching

Or-parallel execution based on McCarthy's Amb operator

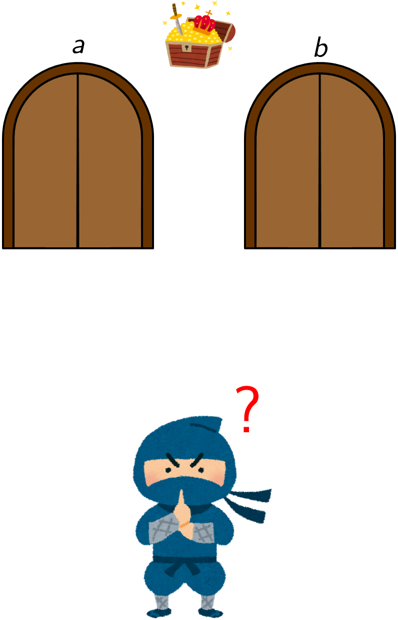
- $\textbf{Amb}(a, b) = \begin{cases} a & \text{if } a \neq \bot \\ b & \text{if } b \neq \bot \\ \bot & \text{if } a = b = \bot. \end{cases}$
- Execute a and b concurrently, and use the value obtained first.
- It can be implemented using Haskell's concurrency module.

Implementation of Amb with Concurrent Haskell

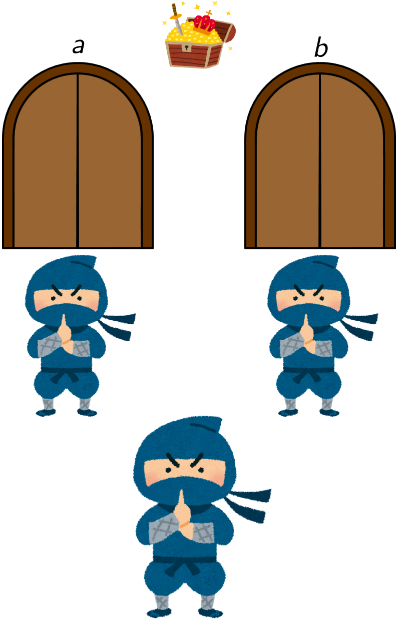
```
ambL :: [D] -> IO D
ambL xs = do
    -- xs is a list of (possibly nonterm.) computations.
  m <- newEmptyMVar
    -- m is a MVar to put the result.
  acts <- sequence [forkIO $ evaluate x >>= putMVar m | x <- xs]
    -- create a process for each x ∈ xs and compute x in parallel to whnf.
    -- Place the result in m. At most one of the processes succeed.
  z <- takeMVar m
    -- take the content of m.
  sequence_ (map killThread acts)
    -- kill the processes which are still running.
  return z
```

(Open Question: How one can prove that this is a correct implementation of **Amb**?)

Explanation of Amb

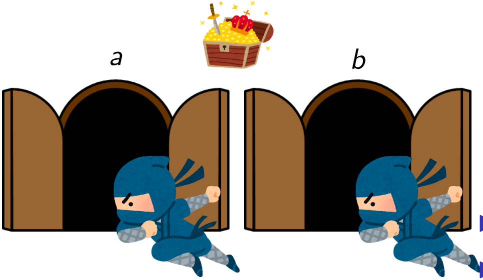


Explanation of Amb



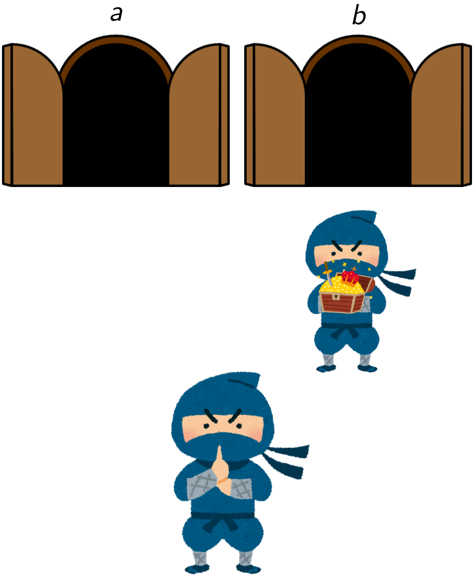
- Ninja’s cloning technique!

Explanation of Amb



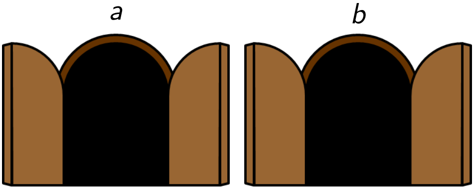
- Ninja's cloning technique!
- Search in parallel.

Explanation of Amb



- Ninja's cloning technique!
- Search in parallel.
- Once a treasure is found and brought back,

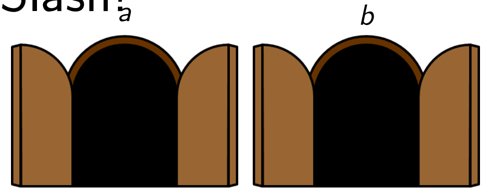
Explanation of Amb



- Ninja's cloning technique!
- Search in parallel.
- Once a treasure is found and brought back,
- Eliminate the others.

Explanation of Amb

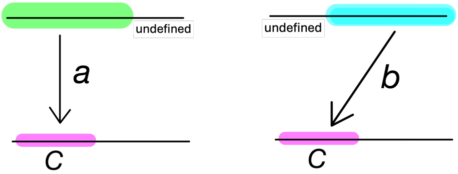
Slash!



- Ninja's cloning technique!
- Search in parallel.
- Once a treasure is found and brought back,
- Eliminate the others.

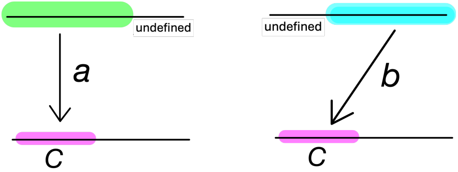
Guaranteeing Termination of a Parallel Program

- We want to guarantee that **Amb**(a, b) is total and correct, i.e., **Amb**$(a, b) \neq \bot$ and the obtained result satisfies the intended specification.



Guaranteeing Termination of a Parallel Program

- We want to guarantee that **Amb**(a, b) is total and correct, i.e., **Amb**$(a, b) \neq \bot$ and the obtained result satisfies the intended specification.



- $a\,\mathbf{r}\,C$: Program a realizes (meets) the specification C.
- $\Downarrow\!\Downarrow(C)$: A specification stating that, as a result of a parallel computation, we obtain a result that satisfies C.
- $c\,\mathbf{r} \Downarrow\!\Downarrow(C) \stackrel{\mathrm{Def}}{=} c = \mathbf{Amb}(a, b) \wedge (a \neq \bot \vee b \neq \bot) \wedge$
  $(a \neq \bot \to a\,\mathbf{r}\,C) \wedge (b \neq \bot \to b\,\mathbf{r}\,C)$
- We want a conditions on a and b that guarantee **Amb**$(a, b)\,\mathbf{r} \Downarrow\!\Downarrow(C)$.

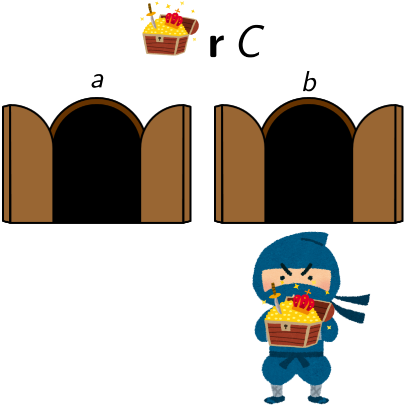
Question

Is the following inference rule sound?

$$\frac{A \to a \,\mathbf{r}\, C \quad B \to b \,\mathbf{r}\, C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b) \,\mathbf{r} \downdownarrows(C)}$$

When B holds:

$$\frac{A \to a \,\mathbf{r}\, C \quad {\color{red} B \to b \,\mathbf{r}\, C} \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b) \,\mathbf{r}\, \Downarrow(C)}$$

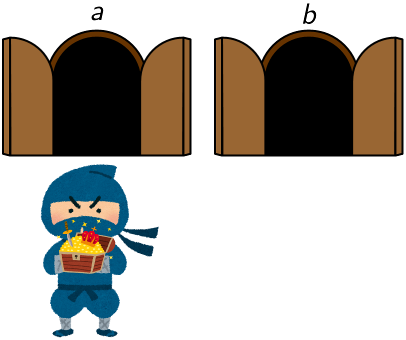


- $b \,\mathbf{r}\, C$ holds.

When A holds:

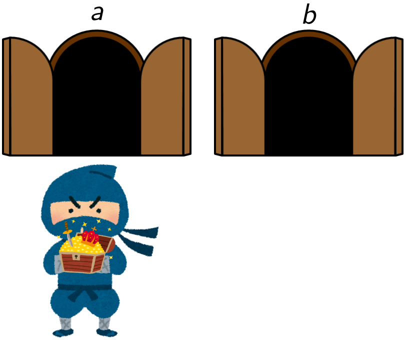
$$\frac{A \to a \mathbf{r} C \quad B \to b \mathbf{r} C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b) \mathbf{r} \ \Downarrow(C)}$$

- $a \mathbf{r} C$ holds.



When A holds:

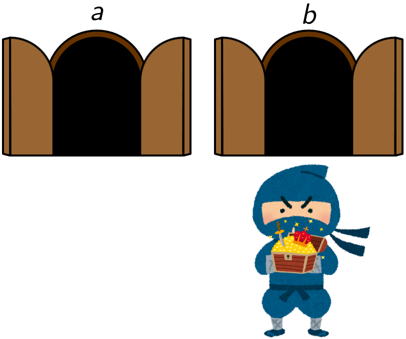
$$\frac{A \to a \,\mathbf{r}\, C \quad B \to b \,\mathbf{r}\, C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b) \,\mathbf{r}\, \downdownarrows(C)}$$



- $a \,\mathbf{r}\, C$ holds.
- We have no information about b. It may be $b = \perp$, or it may not.

When A holds:

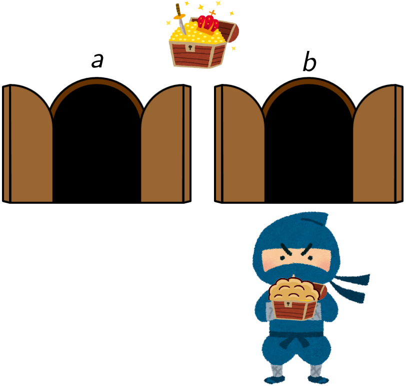
$$\frac{A \to a\,\mathbf{r}\,C \quad B \to b\,\mathbf{r}\,C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b)\,\mathbf{r}\,\Downarrow(C)}$$



- $a\,\mathbf{r}\,C$ holds.
- We have no information about b. It may be $b = \bot$, or it may not.
- If $b \neq \bot$, then possibly $b\,\mathbf{r}\,C$.

When A holds:

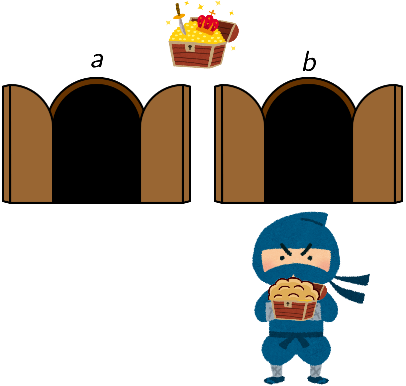
$$\frac{A \to a \,\mathbf{r}\, C \quad B \to b \,\mathbf{r}\, C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b) \,\mathbf{r}\, \Downarrow(C)}$$



- $a \,\mathbf{r}\, C$ holds.
- We have no information about b. It may be $b = \perp$, or it may not.
- If $b \neq \perp$, then possibly $b \,\mathbf{r}\, C$.
- But it may also be that $b \,\mathbf{r}\, C$ does not hold.

When A holds:

$$\frac{A \to a\,\mathbf{r}\,C \quad B \to b\,\mathbf{r}\,C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b)\,\mathbf{r}\;\downdownarrows(C)}$$

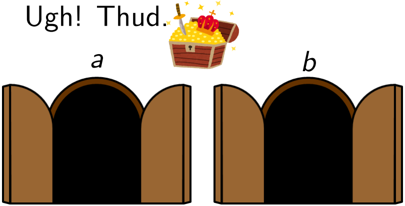


- $a\,\mathbf{r}\,C$ holds.
- We have no information about b. It may be $b = \bot$, or it may not.
- If $b \neq \bot$, then possibly $b\,\mathbf{r}\,C$.
- But it may also be that $b\,\mathbf{r}\,C$ does not hold.
- If the execution of b finishes first, then...

When A holds:

$$\frac{\color{red}{A \to a\,\mathbf{r}\,C} \quad B \to b\,\mathbf{r}\,C \quad \neg\neg(A \vee B)}{\mathbf{Amb}(a, b)\,\mathbf{r}\,\downdownarrows(C)}$$

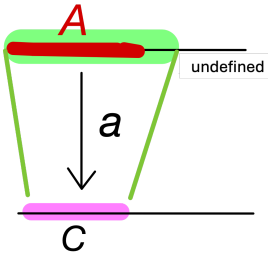
Wait!,
That's a fake!
I will find the true treasure...
Ugh! Thud.



- $a\,\mathbf{r}\,C$ holds.
- We have no information about b. It may be $b = \bot$, or it may not.
- If $b \neq \bot$, then possibly $b\,\mathbf{r}\,C$.
- But it may also be that $b\,\mathbf{r}\,C$ does not hold.
- If the execution of b finishes first, then...

Another Logical Symbol

- In addition to $\Downarrow(A)$, we also introduce $C|_A$.
- $a\,\mathbf{r}\,C|_A$ means two things:
  1. If A holds, then the computation a terminates.
  2. (Regardless of A) If a terminates, then its result satisfies C.
- $a\,\mathbf{r}\,C|_A \stackrel{\text{Def}}{=} (\mathbf{r}\ A \rightarrow a \neq \bot) \wedge (a \neq \bot \rightarrow a\,\mathbf{r}\,C)$



Another Logical Symbol

$\mathbf{r}\ A \cdots \exists c\ c\ \mathbf{r}\ A$

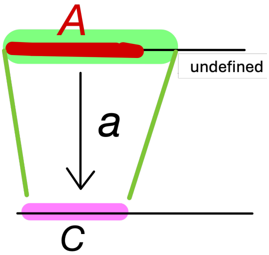
$\mathbf{H}(A) \cdots$ Harrop formula A is realizable.

$$c\ \mathbf{r} \Downarrow\!(A) \stackrel{\mathrm{Def}}{=} c = \mathbf{Amb}(a,b) \wedge (a \neq \bot \vee b \neq \bot) \wedge (a \neq \bot \to a\ \mathbf{r}\ A) \wedge (b \neq \bot \to b\ \mathbf{r}\ A)$$

- In addition to $\Downarrow\!(A)$, we als
- $a\ \mathbf{r}\ C|_A$ means two things:
  1. If A holds, then the computation a terminates.
  2. (Regardless of A) If a terminates, then its result satisfies C.
- $a\ \mathbf{r}\ C|_A \stackrel{\mathrm{Def}}{=} (\mathbf{r}\ A \to a \neq \bot) \wedge (a \neq \bot \to a\ \mathbf{r}\ C)$
- The correct inference is:

$$\frac{a\ \mathbf{r}\ C|_A \quad b\ \mathbf{r}\ C|_B \quad \mathbf{H}(\neg\neg(A \vee B))}{\mathbf{Amb}(a,b)\ \mathbf{r} \Downarrow\!(C)}$$

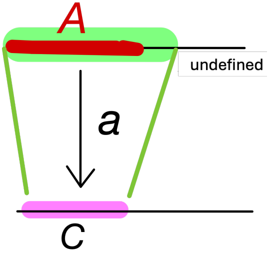
- This inference can be proved with classical logic.



Another Logical Symbol

$\mathbf{r}\,A \cdots \exists c\; c\,\mathbf{r}\,A$
$\mathbf{H}(A) \cdots$ Harrop formula A is realizable.
$c\,\mathbf{r} \downdownarrows(A) \stackrel{\mathrm{Def}}{=} c = \mathbf{Amb}(a, b) \wedge (a \neq \bot \vee b \neq \bot) \wedge (a \neq \bot \to a\,\mathbf{r}\,A) \wedge (b \neq \bot \to b\,\mathbf{r}\,A)$

- In addition to $\downdownarrows(A)$, we als
- $a\,\mathbf{r}\,C|_A$ means two things:
  1. If A holds, then the computation a terminates.
  2. (Regardless of A) If a terminates, then its result satisfies C.
- $a\,\mathbf{r}\,C|_A \stackrel{\mathrm{Def}}{=} (\mathbf{r}\,A \to a \neq \bot) \wedge (a \neq \bot \to a\,\mathbf{r}\,C)$



- The correct inference is:

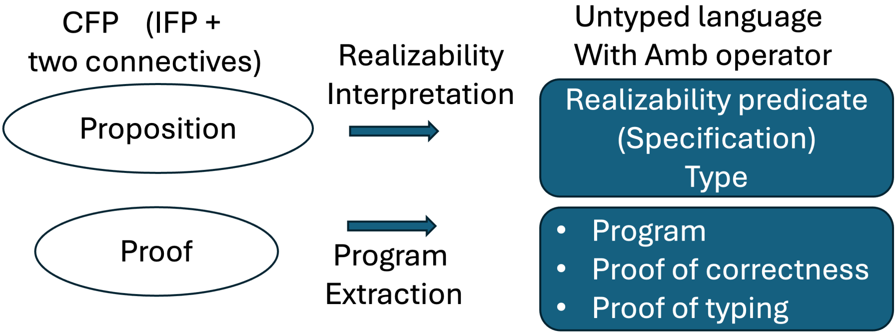
$$\frac{a\,\mathbf{r}\,C|_A \quad b\,\mathbf{r}\,C|_B \quad \mathbf{H}(\neg\neg(A \vee B))}{\mathbf{Amb}(a, b)\,\mathbf{r} \downdownarrows(C)}$$

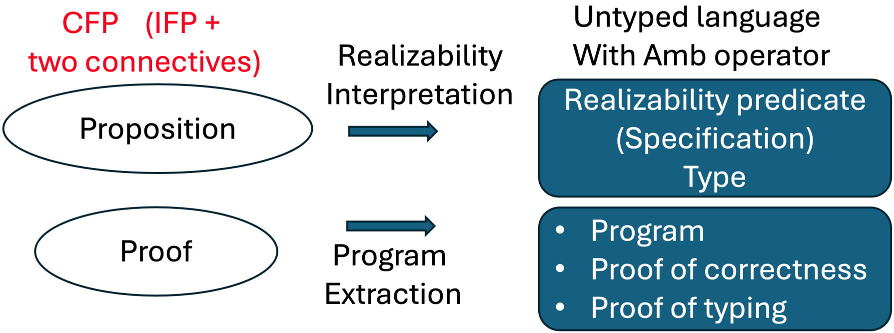
- This inference can be proved with classical logic.
- Instead of proving it directly, we introduce an extension CFP of IFP in which one can prove

$$C|_A \to C|_B \to \neg\neg(A \vee B) \to \downdownarrows(C)$$

and from the proof, one can extract the program $\lambda a.\lambda b.\mathbf{Amb}(a, b)$ and the proof of the above inference in RCFP (classical version of RIFP).

CFP and program extraction




CFP (IFP +
two connectives)
Proposition
Proof
Realizability
Interpretation
Program
Extraction
Untyped language
With Amb operator
Realizability predicate
(Specification)
Type
Program
Proof of correctness
Proof of typing

CFP

- Language: many-sorted first-order language (e.g., a sort for reals).
- Syntax:

$$\begin{aligned}
\mathit{Formulas} \ni A, B &::= P(t) \mid A \wedge B \mid A \vee B \mid A \to B \\
&\quad \mid \forall x\, A \mid \exists x\, A \mid B|_A \mid {\downdownarrows}(B) \\
\mathit{Predicates} \ni P, Q &::= X \mid P_c \mid \lambda \vec{x} A \mid \mu(\Phi) \mid \nu(\Phi) \\
\mathit{Operators} \ni \Phi \quad &::= \lambda X\, P
\end{aligned}$$

  (In $\lambda X\, P$, P must be strictly positive in X.
  In $B|_A$ and ${\downdownarrows}(B)$, B must be strict (see below).)

- B is strict: $\bot$ cannot be a realizer of B and B does not have the form ${\downdownarrows}(B')$.
- Axioms: nc-formulas (formulas without $\vee$, free predicate variables, $\downdownarrows$, or $|$) for extraction.
- Inference rules: those of IFP (in particular rules for induction and coinduction) plus

Inference Rules of CFP

$$\frac{A \to (B_0 \lor B_1) \quad \neg A \to B_0 \land B_1}{(B_0 \lor B_1)|_A} \text{ Rest-intro } (A, B_0, B_1 \text{ Harrop})$$

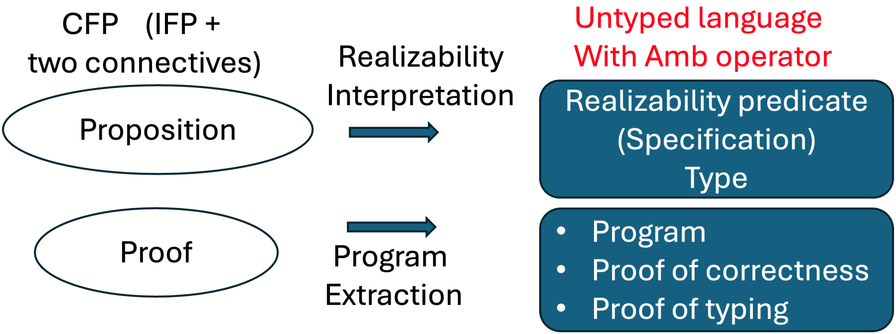
$$\frac{B|_A \quad B \to (B'|_A)}{B'|_A} \text{ Rest-bind} \qquad \frac{B}{B|_A} \text{ Rest-return}$$

$$\frac{A' \to A \quad B|_A}{B|_{A'}} \text{ Rest-antimon} \qquad \frac{B|_A \quad A}{B} \text{ Rest-mp}$$

$$\frac{}{B|_{\mathbf{False}}} \text{ Rest-efq} \qquad \frac{B|_A}{B|_{\neg\neg A}} \text{ Rest-stab}$$

$$\frac{B|_A \quad B|_{\neg A}}{\downdownarrows(B)} \text{ Conc-lem} \qquad \frac{A}{\downdownarrows(A)} \text{ Conc-return}$$

$$\frac{A \to B \quad \downdownarrows(A)}{\downdownarrows(B)} \text{ Conc-mp}$$


CFP (IFP +
two connectives)
Proposition
Proof
Realizability
Interpretation
Program
Extraction
Untyped language
With Amb operator
Realizability predicate
(Specification)
Type
Program
Proof of correctness
Proof of typing

Programming Language

- Untyped lambda calculus (+ recursion, constructors)

$$
\begin{aligned}
M, N, L ::= {} & a, b \ \ \text{(program variables)} \mid \lambda a.\, M \mid M\, N \mid \mathbf{rec}\, M \mid \bot \\
& \mid \mathbf{Nil} \mid \mathbf{Left}(M) \mid \mathbf{Right}(M) \mid \mathbf{Pair}(M, N) \\
& \mid \mathbf{case}\, M\, \mathbf{of}\, \{\mathbf{Nil} \to N\} \mid \mathbf{case}\, M\, \mathbf{of}\, \{\mathbf{Left}(a) \to L;\ \mathbf{Right}(b) \to N\} \\
& \mid \mathbf{case}\, M\, \mathbf{of}\, \{\mathbf{Pair}(a, b) \to N\} \mid M{\downarrow}N \mid \mathbf{Amb}(M, N) \\
& \mid \mathbf{case}\, M\, \mathbf{of}\, \{\mathbf{Amb}(a, b) \to N\}
\end{aligned}
$$

- $M{\downarrow}N$: strict application (i.e., evaluate N to WHNF before applying M).
- Recursive type system:

$$
\begin{aligned}
\rho, \sigma ::= {} & \alpha \ \text{(type variable)} \mid \mathbf{1} \mid \rho \times \sigma \mid \rho + \sigma \mid \rho \Rightarrow \sigma \\
& \mid \mathbf{fix}\, \alpha \,.\, \rho \mid \mathbf{A}(\rho)
\end{aligned}
$$

(with side conditions for $\mathbf{fix}\, \alpha \,.\, \rho$ and $\mathbf{A}(\rho)$)

- Typing rules: Those of IFP plus $\dfrac{\Gamma \vdash M : \rho \qquad \Gamma \vdash N : \rho}{\Gamma \vdash \mathbf{Amb}(M, N) : \mathbf{A}(\rho)}$

Difficulty of the **Amb** operator

- The ordinary **Amb** operator does not commute with function application.
  - $f \overset{\mathrm{Def}}{=} \lambda a.\mathbf{case}\, a\, \mathbf{of}\ \{\mathbf{Left}(\_) \to \mathbf{Left}(\mathbf{Nil}); \mathbf{Right}(\_) \to \bot\}$
  - $f\ 0 = 0$, $f\ 1 = \bot$. (Recall $0 = \mathbf{Left}(\mathbf{Nil})$, $1 = \mathbf{Right}(\mathbf{Left}(\mathbf{Nil}))$)
  - Therefore $\mathbf{Amb}(f\ 0, f\ 1) \overset{\mathrm{c}}{\rightsquigarrow} 0$
  - However, $f(\mathbf{Amb}(0, 1)) \overset{\mathrm{c}}{\rightsquigarrow} 0$ or diverge.
- Usually, **Amb** is interpreted in a power domain, and requires complicated mathematical structure.
- Our type system disallow function application $f(\mathbf{Amb}(a, b))$
- $\mathbf{Amb}(0, 1)$ has type $\mathbf{A}(\mathbf{nat})$, not **nat**.

Two-level Semantics

- Instead of function application, we use "mapamb f **Amb**(a, b)" where mapamb $\stackrel{\mathrm{Def}}{=} \lambda f.\lambda c.\mathbf{case}\, c\, \mathbf{of}\, \{\mathbf{Amb}(a; b) \to \mathbf{Amb}(f{\downarrow}a, f{\downarrow}b)\}$

  mapamb M $\mathbf{Amb}(N_1, N_2) \rightsquigarrow \mathbf{Amb}(M{\downarrow}N_1, M{\downarrow}N_2)$
  $\stackrel{\mathrm{c}}{\rightsquigarrow}^{*} \mathbf{Amb}(M{\downarrow}N_1', M{\downarrow}N_2') \stackrel{\mathrm{c}}{\rightsquigarrow} \ldots$
- **Amb** is just a pair. Parallel execution is done only when **Amb** comes to the head position (or under constructors).
- In this sense, **Amb** is a globally angelic choice operator.
- It may be useful with current multi-core hardwares.

Operational Semantics

(s-i) $(\lambda a.\, M)\; N \rightsquigarrow M[N/a]$

(s-ii) $$\frac{M \rightsquigarrow M'}{M\, N \rightsquigarrow M'\, N}$$

(s-iii) $(\lambda a.\, M){\downarrow}N \rightsquigarrow M[N/a]$ if N is a WHNF.

(s-iv) $$\frac{M \rightsquigarrow M'}{M{\downarrow}N \rightsquigarrow M'{\downarrow}N}$$ if N is a WHNF.

(s-v) $$\frac{N \rightsquigarrow N'}{M{\downarrow}N \rightsquigarrow M{\downarrow}N'}$$

(s-vi) $\mathbf{rec}\; M \rightsquigarrow M\,(\mathbf{rec}\; M)$

(s-vii)
$\mathbf{case}\; C(\vec{M})\;\mathbf{of}\;\{\ldots; C(\vec{b}) \to N; \ldots\} \rightsquigarrow N[\vec{M}/\vec{b}]$

(s-viii) $$\frac{M \rightsquigarrow M'}{\mathbf{case}\; M\;\mathbf{of}\;\{\vec{Cl}\} \rightsquigarrow \mathbf{case}\; M'\;\mathbf{of}\;\{\vec{Cl}\}}$$

(s-ix) $M \rightsquigarrow \bot$ if M is $\bot$-like

(c-i) $$\frac{M \rightsquigarrow M'}{M \overset{\mathrm{c}}{\rightsquigarrow} M'}$$

(c-ii) $$\frac{M_1 \rightsquigarrow M_1'}{\mathbf{Amb}(M_1, M_2) \overset{\mathrm{c}}{\rightsquigarrow} \mathbf{Amb}(M_1', M_2)}$$

(c-ii') $$\frac{M_2 \rightsquigarrow M_2'}{\mathbf{Amb}(M_1, M_2) \overset{\mathrm{c}}{\rightsquigarrow} \mathbf{Amb}(M_1, M_2')}$$

(c-iii) $\mathbf{Amb}(M_1, M_2) \overset{\mathrm{c}}{\rightsquigarrow} M_1$ if M_1 is a WHNF.

(c-iii') $\mathbf{Amb}(M_1, M_2) \overset{\mathrm{c}}{\rightsquigarrow} M_2$ if M_2 is a WHNF.

(p-i) $$\frac{M \overset{\mathrm{c}}{\rightsquigarrow} M'}{M \overset{\mathrm{p}}{\rightsquigarrow} M'}$$

(p-ii) $$\frac{M_i \overset{\mathrm{p}}{\rightsquigarrow} M_i' \;\; (i = 1, \ldots, k)}{C(M_1, \ldots, M_k) \overset{\mathrm{p}}{\rightsquigarrow} C(M_1', \ldots, M_k')}$$
$(C \in \mathrm{C_d})$

(p-iii) $\lambda a.\, M \overset{\mathrm{p}}{\rightsquigarrow} \lambda a.\, M$

Denotational Semantics (First Step)

$$D = (\mathbf{Nil} + \mathbf{Left}(D) + \mathbf{Right}(D) + \mathbf{Pair}(D \times D) + \mathbf{Fun}(D \to D) + \mathbf{Amb}(D \times D))_\perp$$

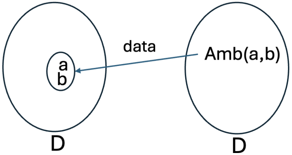
- A program M is interpreted as $[\![M]\!] \in D$.
- $\mathbf{Amb}(a, b)$ is treated as a plain pair (no use of a power domain).

$$
\begin{array}{rcl}
[\![a]\!]\eta & = & \eta(a) \\
[\![\lambda a.\, M]\!]\eta & = & \mathbf{Fun}(f) \quad \text{where } f(d) = [\![M]\!]\eta[a \mapsto d] \\
[\![M\ N]\!]\eta & = & f([\![N]\!]\eta) \quad \text{if } [\![M]\!]\eta = \mathbf{Fun}(f) \\
[\![M {\downarrow} N]\!]\eta & = & f([\![N]\!]\eta) \quad \text{if } [\![M]\!]\eta = \mathbf{Fun}(f) \text{ and } [\![N]\!] \neq \perp \\
[\![\mathbf{rec}\, M]\!]\eta & = & \text{the least fixed point of } f \text{ if } [\![M]\!]\eta = \mathbf{Fun}(f) \\
[\![C(M_1, \ldots, M_k)]\!]\eta & = & C([\![M_1]\!]\eta, \ldots, [\![M_k]\!]\eta) \\
 & & \quad (C \text{ a constructor including } \mathbf{Amb}) \\
[\![\mathbf{case}\, M\, \mathbf{of}\, \vec{Cl}]\!]\eta & = & [\![K]\!]\eta[\vec{a} \mapsto \vec{d}] \quad \text{if } [\![M]\!]\eta = C(\vec{d}) \text{ and } C(\vec{a}) \to K \in \vec{Cl} \\
[\![M]\!]\eta & = & \perp \ \text{ in all other cases, in particular } [\![\perp]\!]\eta = \perp
\end{array}
$$

Denotational Semantics (Second Step)

For each $a \in D$, define the set $\mathrm{data}(a) \subseteq D$ of possible values obtained by computing a.

$$
\begin{aligned}
\mathrm{data}(\mathbf{Amb}(a,b)) &= \mathrm{data}(a) \cup \mathrm{data}(b) && \text{(if } a \neq \bot \text{ and } b \neq \bot\text{).}\\
\mathrm{data}(\mathbf{Amb}(a,\bot)) &= \mathrm{data}(a)\\
\mathrm{data}(\mathbf{Amb}(\bot,b)) &= \mathrm{data}(b)\\
\mathrm{data}(\mathbf{Amb}(\bot,\bot)) &= \{\bot\}\\
\mathrm{data}(\mathbf{Nil}) &= \{\mathbf{Nil}\}\\
\mathrm{data}(\mathbf{Left}(a)) &= \{\mathbf{Left}(a') \mid a' \in \mathrm{data}(a)\}\\
\mathrm{data}(\mathbf{Pair}(a,b)) &= \{\mathbf{Pair}(a',b') \mid a' \in \mathrm{data}(a), b' \in \mathrm{data}(b)\}\\
\mathrm{data}(\mathbf{Fun}(f)) &= \{\mathbf{Fun}(f)\}\\
\mathrm{data}(\bot) &= \{\bot\}
\end{aligned}
$$



Adequacy Theorem

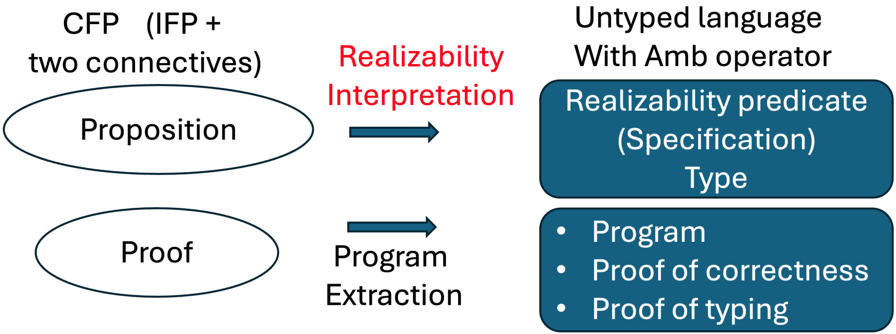
Adequacy Theorem: For a typable program M, the following are equivalent:

- $d \in \mathrm{data}([\![M]\!])$.
- There exists a computation sequence $M = M_0 \stackrel{\mathrm{p}}{\rightsquigarrow} M_1 \stackrel{\mathrm{p}}{\rightsquigarrow} M_2 \stackrel{\mathrm{p}}{\rightsquigarrow} \dots$ such that $d = \sqcup_{i \in \mathbf{N}}((M_i)_D)$.

cf. Computational adequacy for IFP:

Theorem[Computational Adequacy of IFP]

In IFP, for all closed program M, there is a unique reduction sequence $M = M_0 \stackrel{\mathrm{p}}{\rightsquigarrow} M_1 \stackrel{\mathrm{p}}{\rightsquigarrow} M_2 \stackrel{\mathrm{p}}{\rightsquigarrow} \dots$ and $\sqcup_{i \in \mathbf{N}}((M_i)_D) = [\![M]\!]$.

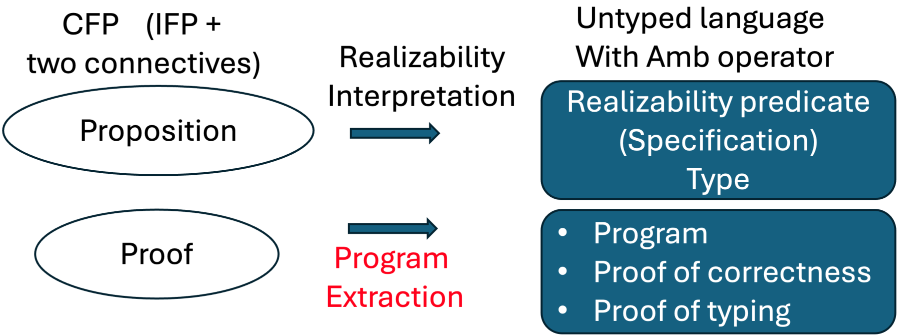

CFP (IFP +
two connectives)
Proposition
Proof
Realizability
Interpretation
Program
Extraction
Untyped language
With Amb operator
Realizability predicate
(Specification)
Type
Program
Proof of correctness
Proof of typing

Realizability Interpretation $a \mathbf{r} A$

- Formalized in RCFP : the extension of RIFP with classical logic. Note that RCFP does not have logical connectives $\downdownarrows(A)$ and $B|_A$.
- Extension of the realizability interpretation of IFP with the followings:

$$c \,\mathbf{r} \downdownarrows(C) \stackrel{\mathrm{Def}}{=} c = \mathbf{Amb}(a, b) \wedge (a \neq \bot \vee b \neq \bot) \wedge \\ (a \neq \bot \to a \,\mathbf{r}\, C) \wedge (b \neq \bot \to b \,\mathbf{r}\, C)$$

$$a \,\mathbf{r}\, C|_A \stackrel{\mathrm{Def}}{=} (\mathbf{r}\ A \to a \neq \bot) \wedge (a \neq \bot \to a \,\mathbf{r}\, C)$$


CFP (IFP +
two connectives)
Proposition
Proof
Realizability
Interpretation
Program
Extraction
Untyped language
With Amb operator
Realizability predicate
(Specification)
Type
• Program
• Proof of correctness
• Proof of typing

Realizability Interpretation of CFP Inference Rules

Lemma

The following inference rules are provable in RCFP.

$$\frac{b\ \mathbf{r}\ (A \to (B_0 \vee B_1)) \quad \mathbf{H}(\neg A \to B_0 \wedge B_1)}{b\ \mathbf{r}\ (B_0 \vee B_1)|_A}$$ Rest-intro (A, B_0, B_1 Harrop)

$$\frac{a\ \mathbf{r}\ B|_A \quad f\ \mathbf{r}\ (B \to (B'|_A))}{(f{\downarrow}a)\ \mathbf{r}\ B'|_A}$$ Rest-bind (B non-Harrop) ($(a\,\mathbf{seq}\,f)\ \mathbf{r}\ B'|_A$ (B Harrop))

$$\frac{a\ \mathbf{r}\ B}{a\ \mathbf{r}\ B|_A}$$ Rest-return

$$\frac{\mathbf{r}\ (A' \to A) \quad a\ \mathbf{r}\ B|_A}{a\ \mathbf{r}\ B|_{A'}}$$ Rest-antimon

$$\frac{b\ \mathbf{r}\ B|_A \quad \mathbf{r}\ A}{b\ \mathbf{r}\ B}$$ Rest-mp

$$\frac{}{\bot\ \mathbf{r}\ B|_{\mathbf{False}}}$$ Rest-efq

$$\frac{b\ \mathbf{r}\ B|_A}{b\ \mathbf{r}\ B|_{\neg\neg A}}$$ Rest-stab

$$\frac{a\ \mathbf{r}\ B|_A \quad b\ \mathbf{r}\ B|_{\neg A}}{\mathbf{Amb}(a, b)\ \mathbf{r}\ {\downdownarrows}(B)}$$ Conc-lem

$$\frac{a\ \mathbf{r}\ A}{\mathbf{Amb}(a, \bot)\ \mathbf{r}\ {\downdownarrows}(A)}$$ Conc-return

$$\frac{f\ \mathbf{r}\ (A \to B) \quad c\ \mathbf{r}\ {\downdownarrows}(A)}{(\mathrm{mapamb}\ f\ c)\ \mathbf{r}\ {\downdownarrows}(B)}$$ Conc-mp (A non-Harrop) ($\mathbf{Amb}(f, \bot)\ \mathbf{r}\ {\downdownarrows}(B)$ (A Harrop))

Realizability Interpretation of CFP Inference Rules

Lemma

The following inference rules are provable in RCFP.

$$\frac{b\ \mathbf{r}\ (A \to (B_0 \vee B_1)) \quad \mathbf{H}(\neg A \to B_0 \wedge B_1)}{b\ \mathbf{r}\ (B_0 \vee B_1)|_A}$$ Rest-intro (A, B_0, B_1 Harrop)

Proof: We use classical logic.
$b\,\mathbf{r}\,(A \to (B_0 \vee B_1))$ means $b : \tau(B_0 \vee B_1)$ and $\mathbf{H}(A) \to b\,\mathbf{r}\,(B_0 \vee B_1)$.
$\mathbf{H}(\neg A \to B_0 \wedge B_1) \equiv \neg \mathbf{H}(A) \to \mathbf{H}(B_0) \wedge \mathbf{H}(B_1)$.
We claim that $(B_0 \vee B_1)|_A$ is realized by b.
Assume $\mathbf{r}\,A$, that is, $\mathbf{H}(A)$. Then b realizes $B_0 \vee B_1$. Hence $b \in \{\mathbf{Left}, \mathbf{Right}\}$ and therefore $b \neq \bot$.
Now assume $b \neq \bot$. We do a classical case analysis on whether or not $\mathbf{H}(A)$.
If $\mathbf{H}(A)$, then $b\,\mathbf{r}\,(B_0 \vee B_1)$. If $\neg\mathbf{H}(A)$, then $\mathbf{H}(B_0)$ and $\mathbf{H}(B_1)$. Hence, **Left** and **Right** both realize $B_0 \vee B_1$. Since $b : \tau(B_0 \vee B_1)$ and $b \neq \bot$, $b \in \{\mathbf{Left}, \mathbf{Right}\}$.
Therefore, $b\,\mathbf{r}\,(B_0 \vee B_1)$.

Conc-mp (A non Harrop)

$(\mathrm{mapamb}\ f\ c)\ \mathbf{r}\ {\Downarrow}(B)$ ($\mathbf{Amb}(f, \bot)\ \mathbf{r}\ {\Downarrow}(B)$ (A Harrop))

Realizability Interpretation of CFP Inference Rules

Lemma

The following inference rules are provable in RCFP.

$$\frac{b\ \mathbf{r}\ (A \to (B_0 \lor B_1)) \quad \mathbf{H}(\neg A \to B_0 \land B_1)}{b\ \mathbf{r}\ (B_0 \lor B_1)|_A}\ \text{Rest-intro}\ (A, B_0, B_1\ \text{Harrop})$$

$$\frac{a\ \mathbf{r}\ B|_A \quad f\ \mathbf{r}\ (B \to (B'|_A))}{(f{\downarrow}a)\ \mathbf{r}\ B'|_A}\ \text{Rest-bind}\ (B\ \text{non-Harrop})\ ((a\,\mathbf{seq}\,f)\ \mathbf{r}\ B'|_A\ (B\ \text{Harrop}))$$

$$\frac{a\ \mathbf{r}\ B}{a\ \mathbf{r}\ B|_A}\ \text{Rest-return}$$

$$\frac{\mathbf{r}\ (A' \to A) \quad a\ \mathbf{r}\ B|_A}{a\ \mathbf{r}\ B|_{A'}}\ \text{Rest-antimon} \qquad \frac{b\ \mathbf{r}\ B|_A \quad \mathbf{r}\ A}{b\ \mathbf{r}\ B}\ \text{Rest-mp}$$

$$\frac{}{\bot\ \mathbf{r}\ B|_{\mathbf{False}}}\ \text{Rest-efq} \qquad \frac{b\ \mathbf{r}\ B|_A}{b\ \mathbf{r}\ B|_{\neg\neg A}}\ \text{Rest-stab}$$

$$\frac{a\ \mathbf{r}\ B|_A \quad b\ \mathbf{r}\ B|_{\neg A}}{\mathbf{Amb}(a, b)\ \mathbf{r}\ \Downarrow(B)}\ \text{Conc-lem} \qquad \frac{a\ \mathbf{r}\ A}{\mathbf{Amb}(a, \bot)\ \mathbf{r}\ \Downarrow(A)}\ \text{Conc-return}$$

$$\frac{f\ \mathbf{r}\ (A \to B) \quad c\ \mathbf{r}\ \Downarrow(A)}{(\text{mapamb}\ f\ c)\ \mathbf{r}\ \Downarrow(B)}\ \text{Conc-mp}\ (A\ \text{non-Harrop})\ (\mathbf{Amb}(f, \bot)\ \mathbf{r}\ \Downarrow(B)\ (A\ \text{Harrop}))$$

Realizability Interpretation of CFP Inference Rules

Lemma

The following inference rules are provable in RCFP.

$$\frac{b\ \mathbf{r}\ (A \to (B_0 \vee B_1)) \quad \mathbf{H}(\neg A \to B_0 \wedge B_1)}{b\ \mathbf{r}\ (B_0 \vee B_1)|_A}\ \text{Rest-intro}\ (A, B_0, B_1\ \text{Harrop})$$

$$\frac{a\ \mathbf{r}\ B|_A \quad f\ \mathbf{r}\ (B \to (B'|_A))}{(f{\downarrow}a)\ \mathbf{r}\ B'|_A}\ \text{Rest-bind}\ (B\ \text{non-Harrop})\ \ ((a\ \mathbf{seq}\ f)\ \mathbf{r}\ B'|_A\ (B\ \text{Harrop})) \qquad \frac{a\ \mathbf{r}\ B}{a\ \mathbf{r}\ B|_A}\ \text{Rest-return}$$

$$\frac{\mathbf{r}\ (A' \to A) \quad a\ \mathbf{r}\ B|_A}{a\ \mathbf{r}\ B|_{A'}}\ \text{Rest-antimon} \qquad \frac{b\ \mathbf{r}\ B|_A \quad \mathbf{r}\ A}{b\ \mathbf{r}\ B}\ \text{Rest-mp}$$

$$\frac{}{\bot\ \mathbf{r}\ B|_{\mathbf{False}}}\ \text{Rest-efq} \qquad \frac{b\ \mathbf{r}\ B|_A}{b\ \mathbf{r}\ B|_{\neg\neg A}}\ \text{Rest-stab}$$

$$\frac{a\ \mathbf{r}\ B|_A \quad b\ \mathbf{r}\ B|_{\neg A}}{\mathbf{Amb}(a, b)\ \mathbf{r}\ {\downdownarrows}(B)}\ \text{Conc-lem} \qquad \frac{a\ \mathbf{r}\ A}{\mathbf{Amb}(a, \bot)\ \mathbf{r}\ {\downdownarrows}(A)}\ \text{Conc-return}$$

Proof: By classical logic $\mathbf{r}\,A$, or $\neg(\mathbf{r}\,A)$ i.e. $\mathbf{r}\,(\neg A)$. In the first case $a \neq \bot$ and in the second case $b \neq \bot$. Further, if $a \neq \bot$, then a is a realizer of B since a realizes $B|_A$. Similarly for b.

Soundness Theorem I

Theorem (Soundness I)

Let $\mathcal{A}$ be a set of nc axioms. From a $\mathrm{CFP}(\mathcal{A})$ *proof of a closed formula A one can extract a program $M : \tau(A)$ such that $M\,\mathbf{r}\,A$ is provable in* $\mathrm{RCFP}(\mathcal{A})$.

Soundness Theorems II

Admissible formula: a syntactic condition mainly about the interaction of fixedpoint, functional implication, and $\downdownarrows$.

Theorem (Soundness II)

If $a \in D$ realizes an admissible formula A, then all $d \in \mathrm{data}(a)$ realize A^-. Here A^- is the formula obtained from A by removing $\downdownarrows$.

Theorem (Program Extraction)

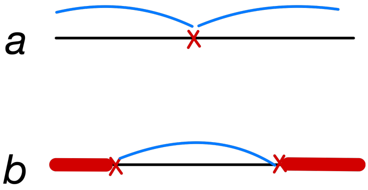
Let M be the program extracted from a $\mathrm{CFP}(\mathcal{A})$ proof of an admissible formula A. For any computation sequence $M = M_0 \overset{\mathrm{c}}{\rightsquigarrow} M_1 \overset{\mathrm{c}}{\rightsquigarrow} \ldots$, $\sqcup_{i \in \mathbf{N}}((M_i)_D)$ realizes A^-.

Example (Restriction)

- $\mathbf{D}(x) \stackrel{\text{Def}}{=} x \neq 0 \to (x \leq 0 \vee x \geq 0)$
- $\mathbf{D}'(x) \stackrel{\text{Def}}{=} (x \leq 0 \vee x \geq 0)|_{x \neq 0}$
- $\mathbf{D} \subseteq \mathbf{D}'$ by (Res-Intro), realizable by $\lambda a.a$.
- $a \,\mathbf{r}\, \mathbf{D}(0)$ means that a is any element of type $\mathbf{1} + \mathbf{1}$.
- $a \,\mathbf{r}\, \mathbf{D}'(0)$means that a may either diverge or terminate, but if it terminates, the result must be $\mathbf{Left}(\mathbf{Nil})$ or $\mathbf{Right}(\mathbf{Nil})$.
- Therefore, they are equivalent.

Example (Parallel Computation)

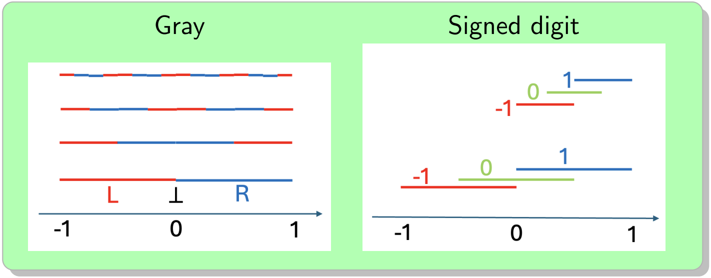
- $\mathbf{ConSD}(x) \stackrel{\mathrm{Def}}{=} \Downarrow((x \leq 0 \lor x \geq 0) \lor |x| \leq 1/2)$.
- $\forall x, \mathbf{D}(x) \land \mathbf{D}(t(x)) \to \mathbf{ConSD}(x)$ is provable for $t(x) \stackrel{\mathrm{Def}}{=} 1 - 2|x|$.
- From its proof, one can extract a program $\mathbf{Amb}(a, b)$ that executes in parallel programs a and b that realize $\mathbf{D}(x)$ and $\mathbf{D}(b)$.
- a and b are partial programs that may not terminate at $x = 0$ and $|x| \geq 1/2$, respectively.
- $\mathbf{Amb}(a, b)$ always terminates.



Conversion from Gray-code to Signed digit representation.

$$\mathbf{D}(x) \overset{\text{Def}}{=} x \neq 0 \to (x \leq 0 \lor x \geq 0)$$
$$\mathbf{G}(x) \overset{\nu}{=} |x| \leq 1 \land \mathbf{D}(x) \land \mathbf{G}(1 - 2|x|)\,.$$

$$\mathbf{SD}(n) \overset{\text{Def}}{=} (n = -1 \lor n = 1) \lor n = 0$$
$$\mathbf{S}(x) \overset{\nu}{=} |x| \leq 1 \land \exists d \in \mathbf{SD}\ \mathbf{S}(2x - d)\,.$$



Conversion from Gray-code to Signed digit representation.

$$\mathbf{D}(x) \stackrel{\mathrm{Def}}{=} x \neq 0 \to (x \leq 0 \vee x \geq 0)$$

$$\mathbf{G}(x) \stackrel{\nu}{=} |x| \leq 1 \wedge \mathbf{D}(x) \wedge \mathbf{G}(1 - 2|x|)\,.$$

$$\mathbf{SD}(n) \stackrel{\mathrm{Def}}{=} (n = -1 \vee n = 1) \vee n = 0$$

$$\mathbf{S}(x) \stackrel{\nu}{=} |x| \leq 1 \wedge \exists d \in \mathbf{SD}\ \mathbf{S}(2x - d)\,.$$

$$\mathbf{S}_2(x) \stackrel{\nu}{=} |x| \leq 1 \wedge \downdownarrows(\exists d \in \mathbf{SD}\ \mathbf{S}_2(2x - d))$$

Theorem

$\forall x(\mathbf{G}(x) \to \mathbf{S}_2(x))$.

Proved in RCFP by coinduction.

gtos Program (Gray Code → Signed Digit Representation)

```
mapamb = \f -> \c -> case c of {Amb(a,b) -> Amb(f $! a, f $! b)}
leftright = \b ->  case b of {Le _ -> Le Nil; Ri _ -> Ri Nil}
conSD = \c -> case c of {Pair(a, b) ->
      Amb(Le $! (leftright a),
          Ri $! (case b of {Le _ -> bot; Ri _ -> Nil}))}
gscomp (Pair(a, Pair(b, p))) = conSD (Pair(a, b))
onedigit  (Pair(a, Pair (b, p))) c = case c of {
       Le d -> case d of {
              Le _ -> Pair(Le(Le Nil), Pair(b,p));
              Ri _  -> Pair(Le(Ri Nil), Pair(notD b,p))};
       Ri _  -> Pair(Ri Nil, Pair(a, nhD p))}
notD a = case a of {Le _ -> Ri Nil; Ri _ -> Le Nil}
nhD (Pair (a, p)) = Pair (notD a, p)
s p = mapamb (onedigit p) (gscomp p)
mon f p = mapamb (mond f) p
   where  mond f (Pair(a,t)) = Pair(a, f t)
gtos = (mon gtos) . s
```

The equivalence of this program and the following can be proved in RCFP.

$$
\begin{aligned}
\text{gtos } (a : b : t) = \mathbf{Amb}(& \\
&(\mathbf{case}\, a\, \mathbf{of}\ \{\mathbf{Left}(\_) \to -1 : \text{gtos } (b : t); \\
&\qquad\qquad\quad \mathbf{Right}(\_) \to 1 : \text{gtos}((\text{not } b) : t)\}), \\
&(\mathbf{case}\, b\, \mathbf{of}\ \{\mathbf{Right}(\_) \to 0 : \text{gtos}(a : (\text{nh } t))\})). \\
&\qquad\qquad\quad \mathbf{Left}(\_) \to \bot\})).
\end{aligned}
$$

Both programs work with the following Haskell deta declaration and the implementation of **Amb** in Concurrent Haskell.

```
data D = Nil| Le D| Ri D| Pair(D,D)| Fun(D->D)| Amb(D,D)
```

Summary

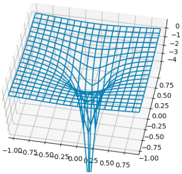
- IFP can be extended to concurrent programs.
- IFP is extended with parallel execution $\downdownarrows(A)$ and restriction $B|_A$ operators, but the program logic does not have these operators.
- The correctness of a program is guaranteed through classical logic.
- Importance of clear denotational semantics for program extraction, and maybe for language design.

Summary

- IFP can be extended to concurrent programs.
- IFP is extended with parallel execution $\downdownarrows(A)$ and restriction $B|_A$ operators, but the program logic does not have these operators.
- The correctness of a program is guaranteed through classical logic.
- Importance of clear denotational semantics for program extraction, and maybe for language design.
- Other example: program to compute the inverse of a regular matrix by Gaussian elimination.

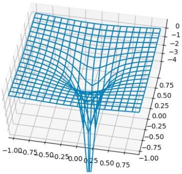
Summary

- IFP can be extended to concurrent programs.
- IFP is extended with parallel execution $\downdownarrows(A)$ and restriction $B|_A$ operators, but the program logic does not have these operators.
- The correctness of a program is guaranteed through classical logic.
- Importance of clear denotational semantics for program extraction, and maybe for language design.
- Other example: program to compute the inverse of a regular matrix by Gaussian elimination.



Summary

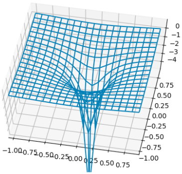
- IFP can be extended to concurrent programs.
- IFP is extended with parallel execution $\downdownarrows(A)$ and restriction $B|_A$ operators, but the program logic does not have these operators.
- The correctness of a program is guaranteed through classical logic.
- Importance of clear denotational semantics for program extraction, and maybe for language design.
- Other example: program to compute the inverse of a regular matrix by Gaussian elimination.



The next lecture is ...

Summary

- IFP can be extended to concurrent programs.
- IFP is extended with parallel execution $\downdownarrows(A)$ and restriction $B|_A$ operators, but the program logic does not have these operators.
- The correctness of a program is guaranteed through classical logic.
- Importance of clear denotational semantics for program extraction, and maybe for language design.
- Other example: program to compute the inverse of a regular matrix by Gaussian elimination.



The next lecture is ...

Thank you very much.