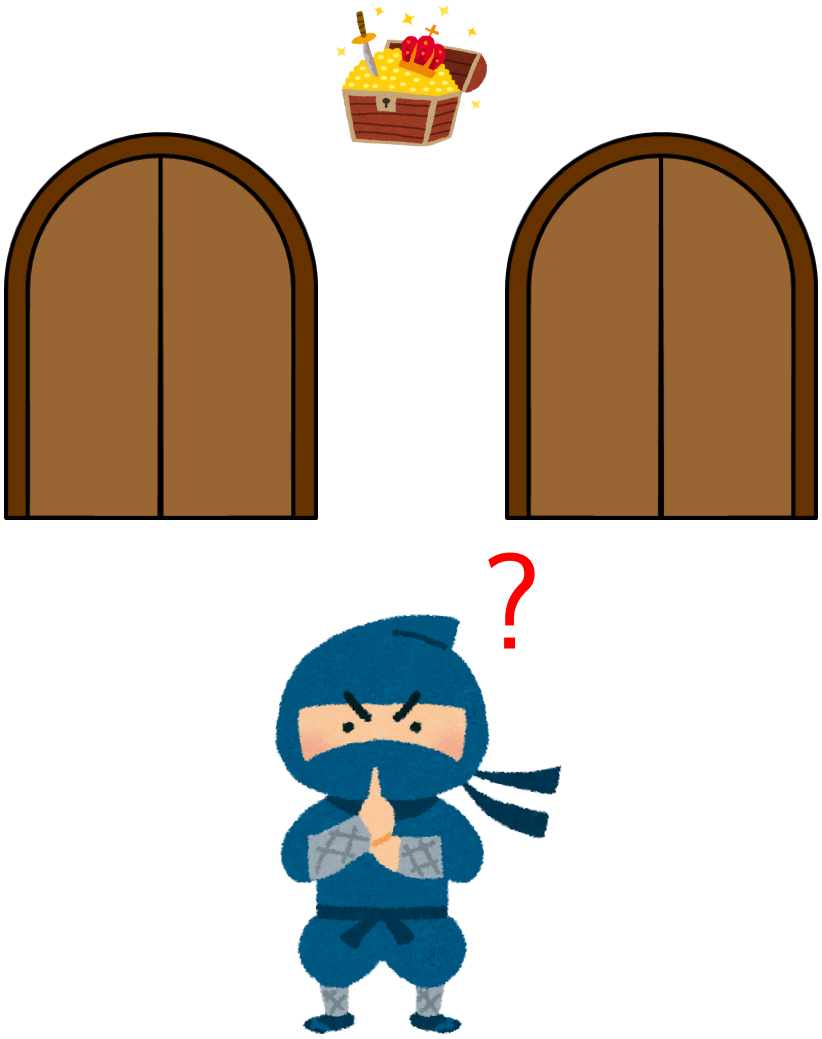
Concurrent program extraction

Ulrich Berger and Hideki Tsuiki

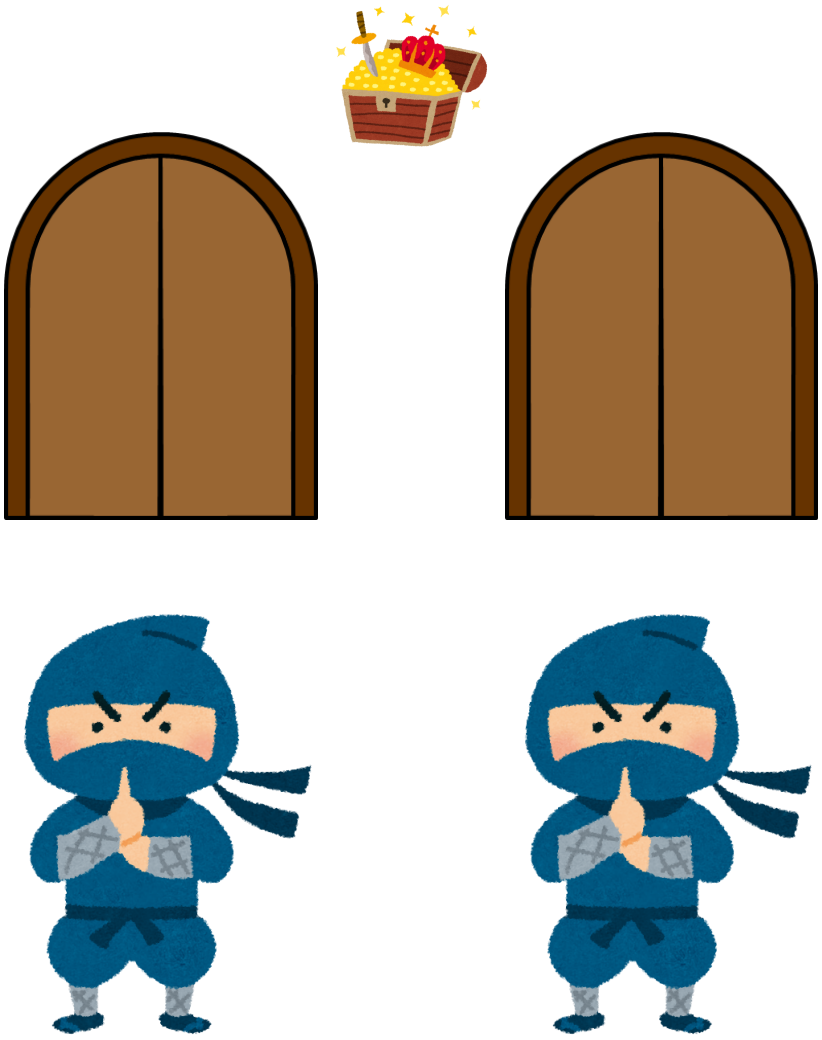
CCC 2017, June 26-30, 2030, Loria, Nancy

Question?



- There are two doors.
- You know that you can get a treasure from at least one of the doors, but do not know which one.
- If you find a treasure, you can return with it. Otherwise, you have to search for it eternally, and no return.
- What shall you do?

Solution!

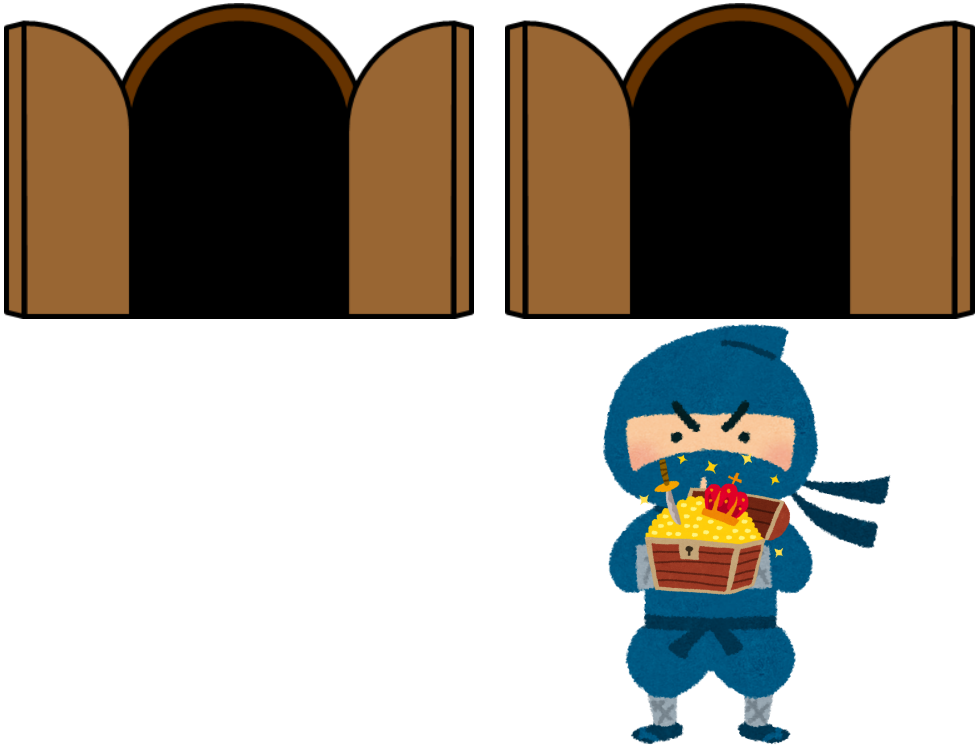


- Duplicate yourself!

Solution!

- Duplicate yourself!
- Search concurrently.

Solution!



- Duplicate yourself!
- Search concurrently.
- If one of them finds a treasure,

Solution!

- Duplicate yourself!
- Search concurrently.
- If one of them finds a treasure,
  Kill the other.

Solution!

ARRGH!

- Duplicate yourself!
- Search concurrently.
- If one of them finds a treasure,
  Kill the other.

Continue with the treasure

Continue with the treasure

We study this kind of concurrent computation.

How can we implement it in programming languages?

Amb: Bottom-avoiding choice operator

- **Amb**: McCarthy's nondeterministic bottom-avoiding choice operator.
- $\textbf{Amb}\ x\ y = \begin{cases} x & \text{if } x \neq \bot \\ y & \text{if } y \neq \bot \\ \bot & \text{if } x = y = \bot. \end{cases}$
- Here, $\bot$ means that computation does not terminate.
- It returns x when $y = \bot$.
- It returns y when $x = \bot$.
- It may return x or y nondeterministically when both x and y are not $\bot$.
- Compute x and y in parallel, and return the one obtained first.
- We want to extract a program that uses the **Amb** operator from a proof that at least one of x or y has a value and therefore **Amb** x y terminates.

The realizability theory with start with

- We extend the realizability theory IFP(Intuitionistic Fixed Point Logic) [B 2010], which is an extension and variation of Kreisel's modified realizability.
  - TCF and Minlog [Schwichtenberg 1991].
  - **q**-realizability of [Tatsuta 1998]
- Logic: Extension of first-order predicate logic by least and greatest fixed points. Note that it is not based on type theory.
- Program: Untyped programs. We allow unrestricted recursion and non-termination. Prove termination through adequacy.
- As the space of programs, consider the Scott domain D defined by a recursive domain equation of the form
  $D = \mathbf{Nil} + \mathbf{Left}\, D + \mathbf{Right}\, D + \mathbf{Pair}\,(D \times D) + \mathbf{Fun}\,(D \to D) + \ldots$
  - $+$: separated sum.
  - $\mathbf{Nil}, \mathbf{Left}, \mathbf{Right}, \ldots$: constructors.
  - All the elements except for $\perp$ are constructor terms like $\mathbf{Left}\ldots, \mathbf{Pair}(\ldots,\ldots), \mathbf{Fun}(\lambda d\ldots.)$.
  - We write $\mathbf{Def}(b)$ to express that b is not $\perp$.

The realizability theory with start with (cont.)

- Formulae are divided into two categories: computational and non-computational (nc). Computational means that it contains $\vee$.
- For a formula A and $c \in D$, we define the predicate $c\,\mathbf{r}\,A$, which means that c is a "computational meaning" of A. Roughly speaking, c computes **Left** or **Right** for each $\vee$ in a computational formula.
- $c\,\mathbf{r}\,A \stackrel{\text{Def}}{=} (c = \mathbf{Nil}) \wedge A$, for a nc formula A.
- $c\,\mathbf{r}\,(A \vee B) \stackrel{\text{Def}}{=} \exists a\,(c = \mathbf{Left}(a) \wedge a\,\mathbf{r}\,A) \vee \exists b\,(c = \mathbf{Right}(b) \wedge b\,\mathbf{r}\,B)$.
- $c\,\mathbf{r}\,(A \to B) \stackrel{\text{Def}}{=} \begin{cases} \forall a\,(a\,\mathbf{r}\,A \to (c\,a)\,\mathbf{r}\,B) & \text{if } A \text{ is computational.} \\ A \to c\,\mathbf{r}\,B & \text{if } A \text{ is nc.} \end{cases}$
- From a derivation of a formula A in IFP, one can extract a program term M and a derivation of $M\,\mathbf{r}\,A$ (Soundness Theorem).

Our extension

- We add a new formula $\mathbf{S}_n(A)$ which means that the computational meaning of A is obtained through n parallel threads of computation.
- We consider a new constructor **Amb** which means the **Amb** operator with n arguments. We add to D the branch $D = ... + \mathbf{Amb}([D])$. Here, $[D]$ is the domain of lists of D.
- **Note:** We do not consider a power domain. $\mathbf{Amb}([d_1, \ldots, d_n])$ is a list of values obtained by each computation.

How can we derive that at least one of a or b terminate?

- First candidate (it is not valid):

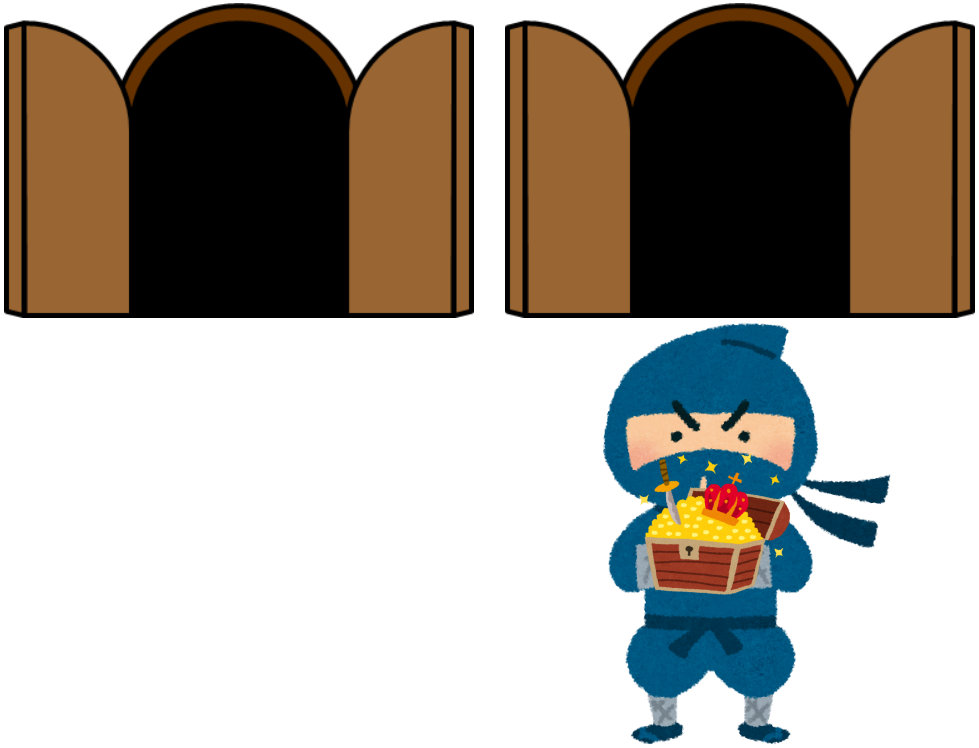
$$\frac{B \to A \quad C \to A \quad \neg\neg(B \vee C)}{\mathbf{S}_2(A)}\ B, C : nc$$

$$\frac{b\,\mathbf{r}\,(B \to A) \quad c\,\mathbf{r}\,(C \to A) \quad \neg\neg(B \vee C)}{\mathbf{Amb}([b, c])\,\mathbf{r}\,\mathbf{S}_2(A)}\ B, C : nc$$

$$\frac{B \to b\,\mathbf{r}\,A \quad C \to c\,\mathbf{r}\,A \quad \neg\neg(B \vee C)}{\mathbf{Amb}([b, c])\,\mathbf{r}\,\mathbf{S}_2(A)}\ B, C : nc$$

- If B holds, b will produce A.
- If C holds, c will produce A.
- If B or C holds.
- Therefore, by executing b and c in parallel, we obtain A.
- However, this reasoning is not valid.

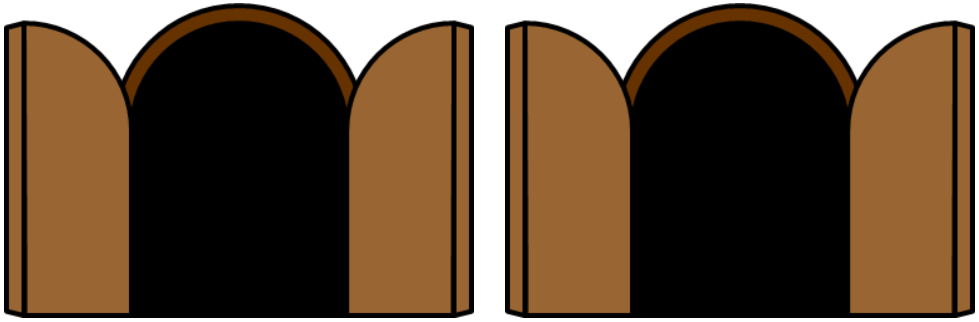
The case C holds:



C

- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(A)}$$
- c returns with a treasure.

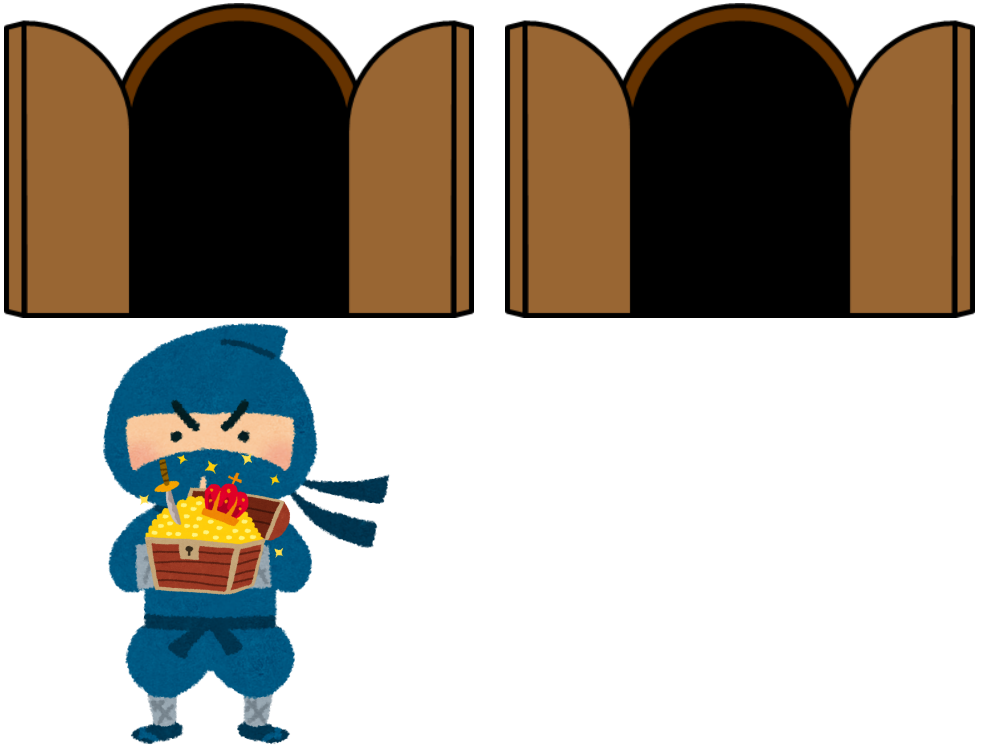
The case B holds:



b

- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(C)}$$
- b returns with a treasure.
-
-
-
-

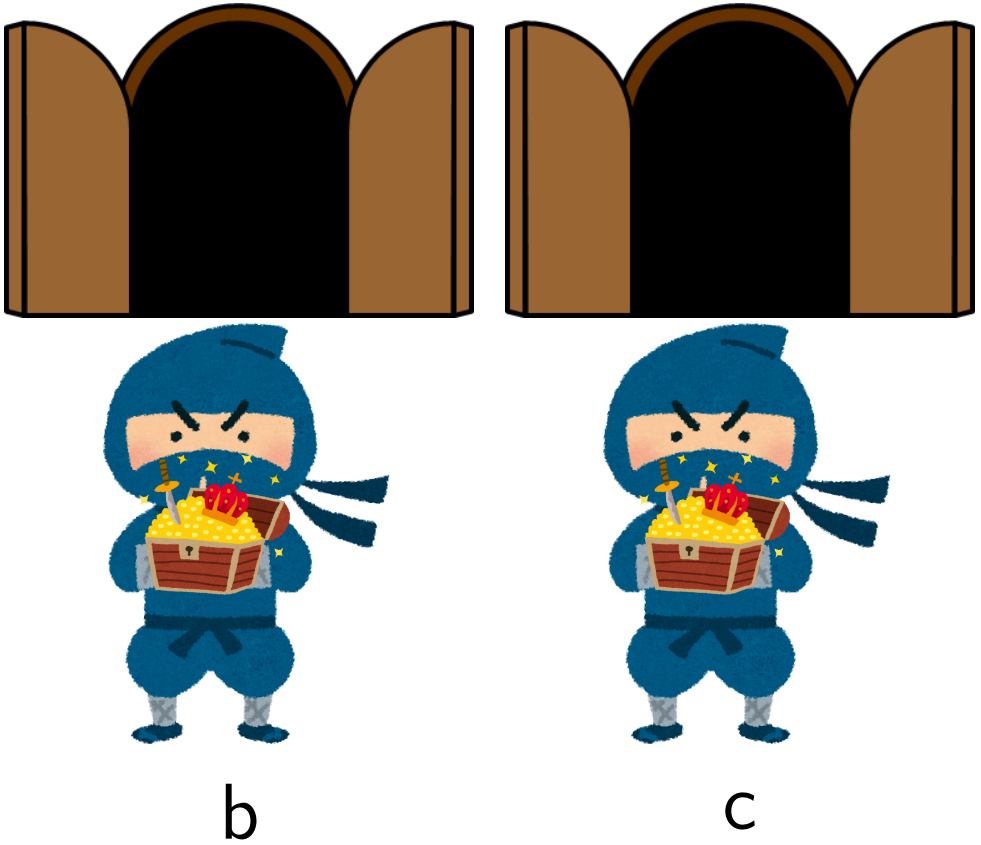
The case B holds:



b

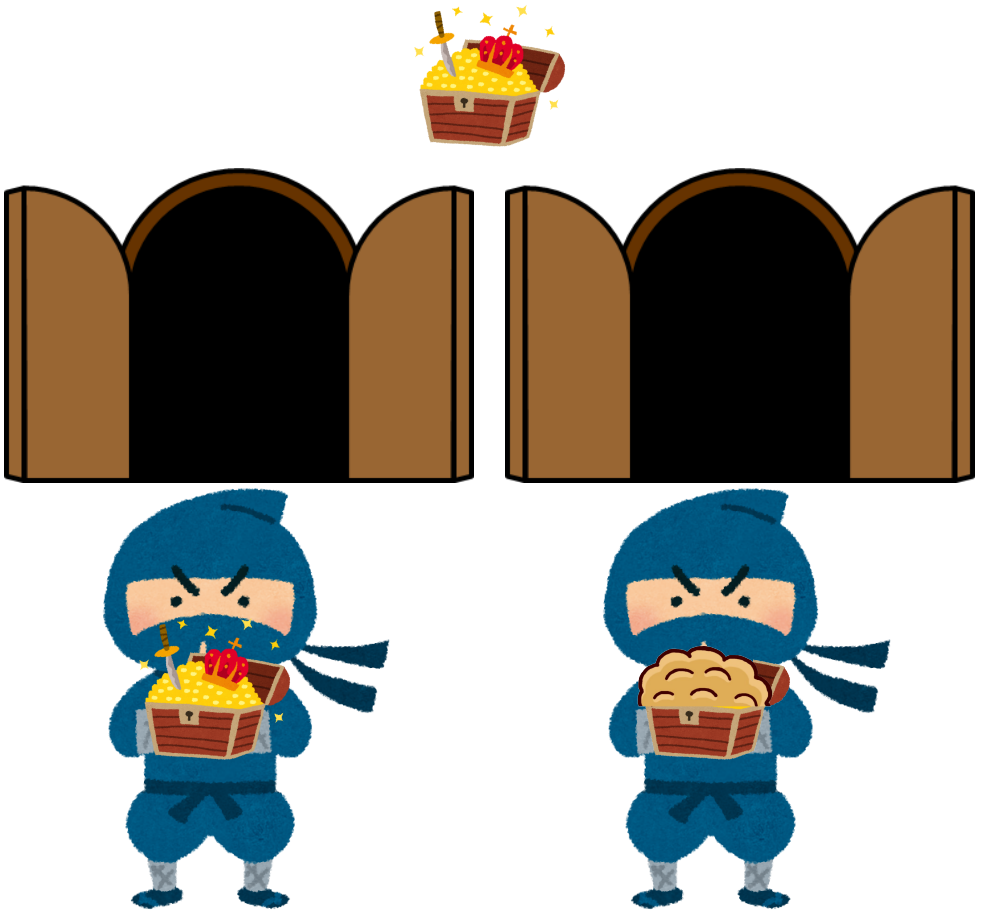
- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(C)}$$
- b returns with a treasure.
- No information about C. Therefore, c may or may not return.
-
-
-

The case B holds:



- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(C)}$$
- b returns with a treasure.
- No information about C. Therefore, c may or may not return.
- c may return with a treasure.
-
-

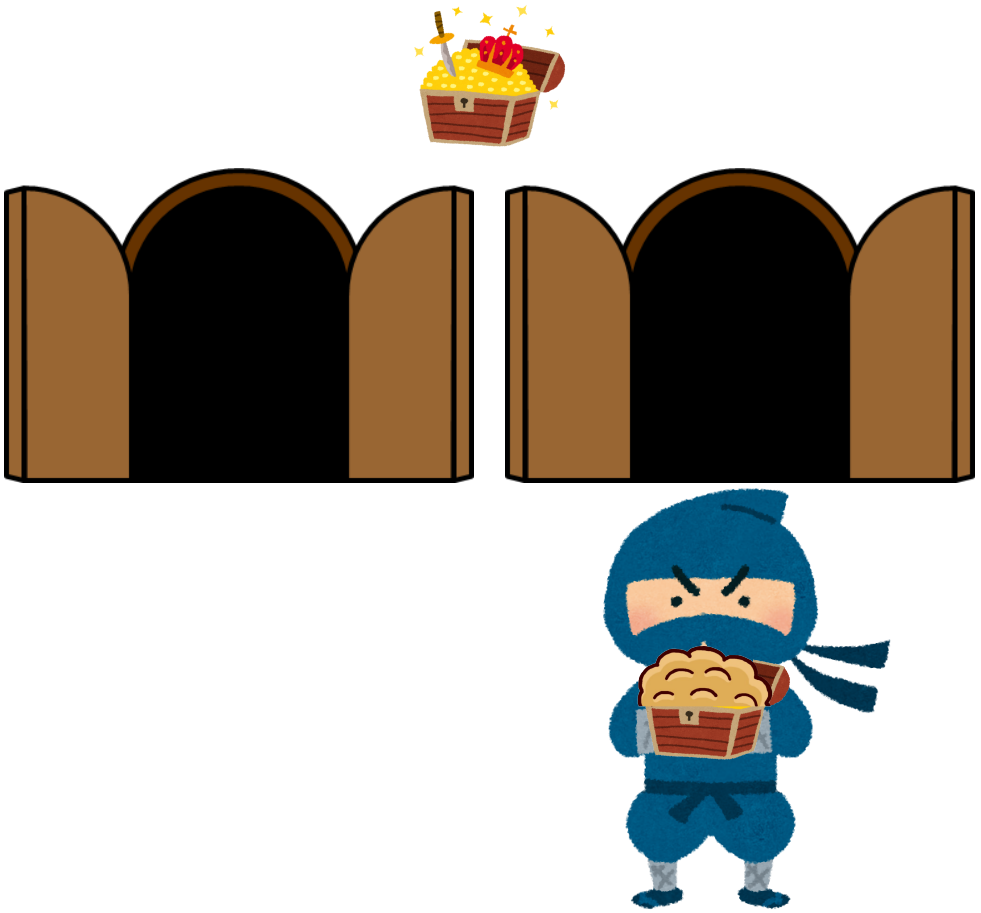
The case B holds:



c

- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(C)}$$
- b returns with a treasure.
- No information about C. Therefore, c may or may not return.
- c may return with a treasure.
- However, c may return with something else.
-

The case B holds:



c

- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(C)}$$
- b returns with a treasure.
- No information about C. Therefore, c may or may not return.
- c may return with a treasure.
- However, c may return with something else.
- If c returns fast,...

The case B holds:

- $$\frac{\begin{array}{c} B \to b\,\mathbf{r}\,A \\ C \to c\,\mathbf{r}\,A \\ \neg\neg(B \vee C) \end{array}}{\mathbf{Amb}([b,c])\,\mathbf{r}\,\mathbf{S}_2(C)}$$
- b returns with a treasure.
- No information about C. Therefore, c may or may not return.
- c may return with a treasure.
- However, c may return with something else.
- If c returns fast,...

New connectives.

- $\mathbf{S}_n(A)$ (Concurrently A)
  - $a \,\mathbf{r}\, \mathbf{S}_n(A) \stackrel{\text{Def}}{=} \exists m\,(1 \leq m \leq n \wedge \; a = \mathbf{Amb}([a_1, \ldots, a_m]) \wedge \; \exists i \leq m\,(\mathbf{Def}(a_i)) \wedge \; \forall i \leq m\,(\mathbf{Def}(a_i) \to a_i \,\mathbf{r}\, A))$.
- $A \parallel B$ (A if B)
  - $a \,\mathbf{r}\, (A \parallel B) \stackrel{\text{Def}}{=} (B \to \mathbf{Def}(a)) \wedge (\mathbf{Def}(a) \to a \,\mathbf{r}\, A)$.
  - (We only consider the case B is nc.)
- Bounded non-determinism and restriction:

$$\frac{A \parallel B \quad A \parallel C \quad \neg\neg(B \vee C)}{\mathbf{S}_2(A)} \text{ nondet-class-orelim}$$

Realizable rules for $\mathbf{S}_n$:

- $\dfrac{A}{\mathbf{S}_n(A)}$ return, realized by $\lambda a\, \mathbf{Amb}([a])$
  for strict A, that is, for a formula for which $\bot\, \mathbf{r} A$ does not hold.

- $\dfrac{A \to B}{\mathbf{S}_n(A) \to \mathbf{S}_n(B)}$ mon, by $\lambda f \lambda a\ \mathbf{case}\, a\, \mathbf{of}\, \{\mathbf{Amb}(b) \to \mathbf{Amb}(map\ f\ \ b)\}$
  for strict B.

- $\dfrac{\mathbf{S}_1(A)}{A}$ one, by $\lambda a\, \mathbf{case}\, a\, \mathbf{of}\, \{\mathbf{Amb}(a_1 : b) \to a_1\}$.

- $\dfrac{\mathbf{S}_n(A)}{A}$ nc, by $\lambda a\, \mathbf{Nil}$
  where A is nc.

Realizable rules for $\parallel$

- $\dfrac{A}{A \parallel B}$ return, by $\lambda a\, a$.
- $\dfrac{A \parallel B \quad A \to (A' \parallel B)}{A' \parallel B}$ bind, by $\lambda a\, \lambda f\, f{\downarrow}a$. ($\downarrow$: strict application.)
- $\dfrac{A \parallel B \quad B' \to B}{A \parallel B'}$ antimon, by $\lambda a\, \lambda f\ a$
- $\dfrac{A \parallel B}{A \parallel \neg\neg B}$ $\parallel$ stab, realized classically by the identity.
- $\dfrac{}{A \parallel \mathbf{False}}$ restriction-efq, by $\bot$
- $\dfrac{A \parallel B \quad B}{A}$ restriction-mp, by $\lambda a\, \lambda\, b\, a$
- $\dfrac{B \to A_0 \vee A_1 \quad \neg B \to A_0 \wedge A_1}{A_0 \vee A_1 \parallel B}$ $\parallel$ I, by

  $\lambda a\, \mathbf{case}\, a\, \mathbf{of}\, \{\mathbf{Left}\ b \to \mathbf{Left}\ b\,;\ \mathbf{Right}\ b \to \mathbf{Right}\ b\}$

  where A_0, A_1, B must be nc.

$$\frac{B \to A_0 \vee A_1 \quad \neg B \to A_0 \wedge A_1}{A_0 \vee A_1 \parallel B} \parallel \mathsf{I}, \text{ where } A_0, A_1, B \text{ must be nc.}$$

by $\lambda a\, \mathbf{case}\, a\, \mathbf{of}\, \{\mathbf{Left}\ b \to \mathbf{Left\ Nil}\,;\ \mathbf{Right}\ b \to \mathbf{Right\ Nil}\}$

$a\,\mathbf{r}\,(A \parallel B) \stackrel{\text{Def}}{=} (B \to \mathbf{Def}(a)) \wedge (\mathbf{Def}(a) \to a\,\mathbf{r}\,A).$

Suppose that $a\,\mathbf{r}\,B \to A_0 \vee A_1$. That is, $B \to a\,\mathbf{r}\,A_0 \vee A_1$.

- If B holds, $a\,\mathbf{r}\,A_0 \vee A_1$ and obviously $a\,\mathbf{r}\,A_0 \vee A_1 \parallel B$. Since A_0 and A_1 are nc, a should be **Left Nil** or **Right Nil**.
- If B does not hold, then a may be anything.
  - If a does not have the form **Left** b or **Right** b, then $\bot\,\mathbf{r}\,A_0 \vee A_1 \parallel B$.
  - if a has the form **Left** b or **Right** b, then **LeftNil** or **RightNil** realizes $A_0 \vee A_1$ because A_0 and A_1 are both true.

Realizable rules for the combination of $\mathbf{S}_n$ and $\parallel$

- $\dfrac{A \parallel B \quad A \parallel \neg B}{\mathbf{S}_2(A)}$ nondet-lem, by $\lambda\, a\, \lambda\, b\, \mathbf{amb}([a, b])$
- $\dfrac{\neg\neg(B \vee C) \quad A \parallel B \quad A \parallel C}{\mathbf{S}_2(A)}$ nondet-class-orelim, by $\lambda\, a\, \lambda\, b\, \mathbf{amb}([a, b])$

We call the extended system CFP (Concurrent Fixedpoint Logic).

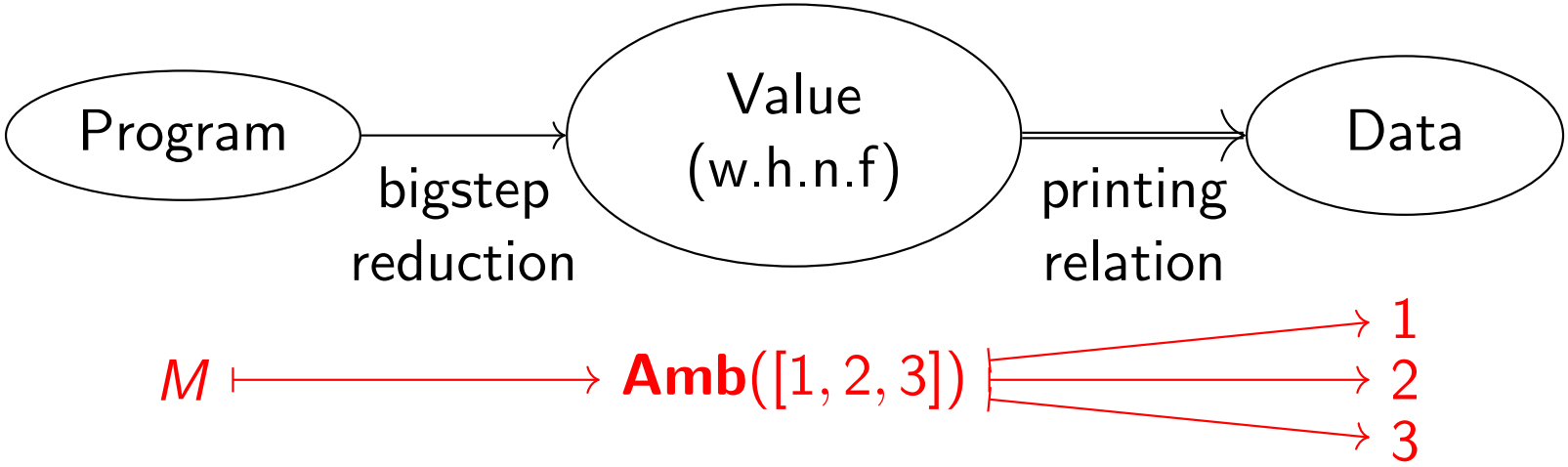
Soundness, Adequacy, Program Extraction Theorem

- A (closed) value is an *intro program* (i.e., weak head normal form). $\mathbf{Amb}[bot, 1, 2]$ is a value. ($bot = rec\ x.x$)
- We define a bigstep reduction relation $M \to v$ between (closed) program M and values v. We have $[\![M]\!] = [\![v]\!]$ and such a v is unique if it exists.
- A data is a term defined inductively only with **Nil**, **Left**, **Right**, **Pair**.
- The printing relation $M \Longrightarrow d$ between program M and data d. (Evaluating deeply, and expanding **Amb**.)
- It is a multi-valued relation. $\mathbf{Amb}[bot, 1, 2] \Longrightarrow 1$, $\mathbf{Amb}[bot, 1, 2] \Longrightarrow 2$
- We define $data(a)$ for $a \in D$ as the set of values obtained by expanding **Amb** for non-bottom components. $data(\mathbf{Amb}[\bot, 1, 2]) = \{1, 2\}$.
- **[Computational Adequacy]** For every closed term M, $d \in data([\![M]\!])$ iff $M \Longrightarrow d$.

Program Extraction

- For any formula A of CFP without restriction($\|$), we define a formula A^- of IFP obtained by deleting all $\mathbf{S}_n$.
- A data formula is a formula without implications.
- If A is a realizable data formula of CFP, A^- is realizable in IFP though one cannot compute a realizer of A^- from a realizer of A in a continuous way in general.
- **[Program Extraction]** From a proof of A in CFP, one can extract a terminating program M such that whenever $M \Longrightarrow d$ then d realizes A^- in IFP.

Haskell implementation



- $data\ D = \mathbf{Nil} + \mathbf{Left}\ D + \ldots + \mathbf{Amb}([D])$ is a Haskell datatype.
- Our realizers are Haskell programs.
- From a proof, we hand-extract a haskell program of type D.
- "Value" is a weak head normal form and our bigstep reduction is compatible with the evaluation of haskell.
- "Printing relation" on **Amb** is implemented in (concurrent) haskell [Jones,Gordon,Finne,96].
- It uses forkIO (spawn a concurrent process) and MVar (mutable variable with blocking).

Haskell implementation of Amb

```
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
```

– With ambL, we define the printing relation printing :: D -> IO D

```
printing (Amb d) = ambL d >>= printing
...
```

Example1: Gray $\rightarrow$ Signed Digit Conversion

- Gray-code of real number (on the unit interval [-1, 1]) is a non-redundant code of real number as $\{0, 1, \perp\}$-sequences.
- At most one $\perp$ is included in each code.
- It is equivalent to signed digit.
- Let ONEBOT be the set of $\{0, 1, \perp\}$-sequences with at most one $\perp$.
- SDtoGRAY : $\{-1, 0, 1\}^\omega \rightarrow$ ONEBOT.
- GRAYtoSD : ONEBOT $\rightarrow \{-1, 0, 1\}^\omega$.
- SDtoGRAY can be written in Haskell. (Haskell list type contain partial infinite sequence like $[0, 1, \perp, 1, 0, 0, \ldots]$).
- For GRAYtoSD, we need to read in a one-bottom sequence.
- To read a one-bottom sequence, one needs to evaluate two cells in parallel because a computation of a cell may not terminate.

The code of $-1/4$ is

0	1	$\perp$	1	0	0	0

$\cdots$

Gray $\to$ Signed Digit Conversion (2)

- $\mathbf{SD} \stackrel{\text{Def}}{=} \{-1, 0, 1\}$
- $\mathbb{I}_d \stackrel{\text{Def}}{=} [d/1 - 1/2, d/2 + 1/2]$
- $\mathrm{C}(x) \stackrel{\nu}{=} \exists d \in \mathbf{SD}\,(x \in \mathbb{I}_d \wedge \mathrm{C}(2x - d))$

  x has a signed digit representation. Defined coinductively.
- $\mathrm{C}_2(x) \stackrel{\nu}{=} \mathbf{S}_2(\exists d \in \mathbf{SD}\,(x \in \mathbb{I}_d \wedge \mathrm{C}_2(2x - d)))$

  x has a signed digit representation, computed concurrently.
- $\mathrm{D}(x) \stackrel{\text{Def}}{=} x \neq 0 \to x \leq 0 \vee x \geq 0$

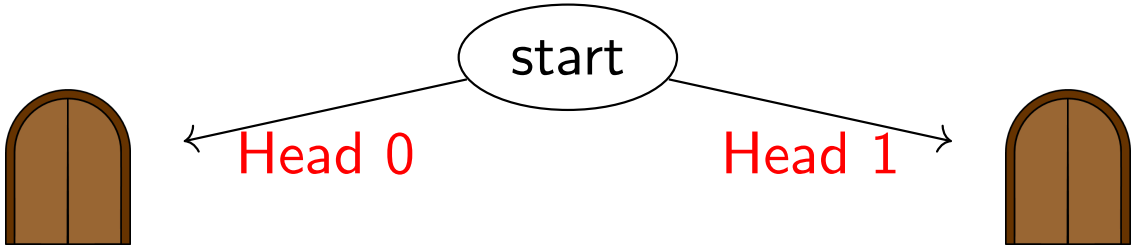
  If $x \neq 0$, then it is computable whether $x \leq 0$ or $x \geq 0$.
- $\mathrm{G}(x) \stackrel{\nu}{=} \mathrm{D}(x) \wedge \mathrm{G}(\mathbf{t}(x))$

  x has a Gray-code.
- We proved $\forall x(\mathrm{C}(x) \to \mathrm{G}(x))$. The extracted program is a Haskell program.
- We proved $\forall x(\mathrm{G}(x) \to \mathrm{C}_2(x))$. The extracted program produces a Haskell program with the **Amb** constructor. Signed digit represesentation is obtained with printing.

Result of computation of gc($\bot 10^\omega$)

Gray code of $-1/4$ is $01\bot 10^\omega$.

$$
\begin{array}{rl}
gc(0:1:\bot:1:[0,0..]) & = \mathbf{Amb}(\mathbf{Mi}(A),\ \mathbf{Le}(B)) \\
\Uparrow\ \Uparrow \qquad\qquad (A) & = (\mathbf{Amb}(\mathbf{Le}(C),\ \bot) \\
(B) & = (\mathbf{Amb}(\mathbf{Ri}(D),\ \bot) \\
(C) & \ldots
\end{array}
$$



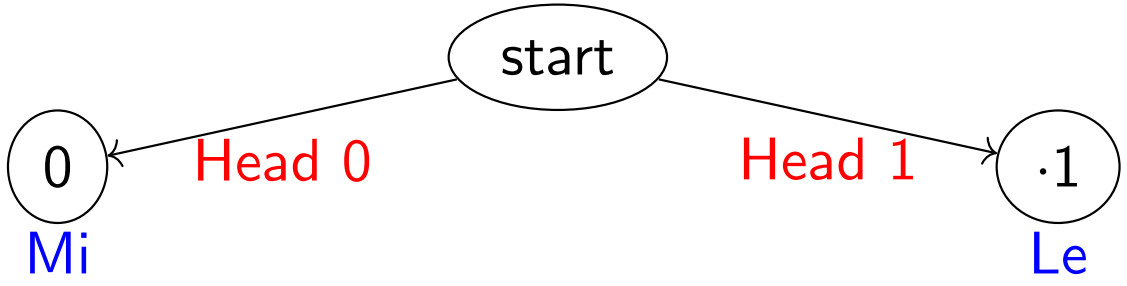
$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Mi}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Le}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

Result of computation of gc($\bot 10^\omega$)

Gray code of $-1/4$ is $01\bot 10^\omega$.

$$
\begin{array}{rll}
gc(0:1:\bot:1:[0,0..]) & = & \mathbf{Amb}(\mathbf{Mi}(A),\ \mathbf{Le}(B)) \\
\Uparrow\ \ \Uparrow \qquad\qquad (A) & = & (\mathbf{Amb}(\mathbf{Le}(C),\ \bot) \\
(B) & = & (\mathbf{Amb}(\mathbf{Ri}(D),\ \bot) \\
(C) & & \ldots
\end{array}
$$



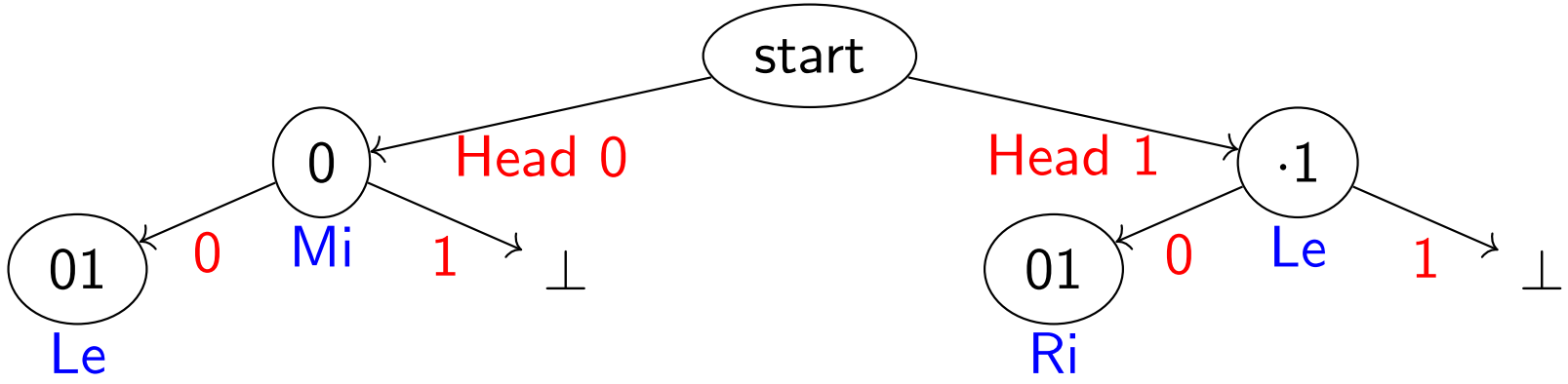
$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Mi}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Le}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

Result of computation of gc($\bot 10^\omega$)

Gray code of $-1/4$ is $01\bot 10^\omega$.

$$
\begin{array}{rll}
gc(0:1:\bot:1:[0,0..]) & = & \mathbf{Amb}(\mathbf{Mi}(A),\ \mathbf{Le}(B)) \\
\Uparrow\ \Uparrow \quad (A) & = & (\mathbf{Amb}(\mathbf{Le}(C),\ \bot) \\
(B) & = & (\mathbf{Amb}(\mathbf{Ri}(D),\ \bot) \\
(C) & & \ldots
\end{array}
$$



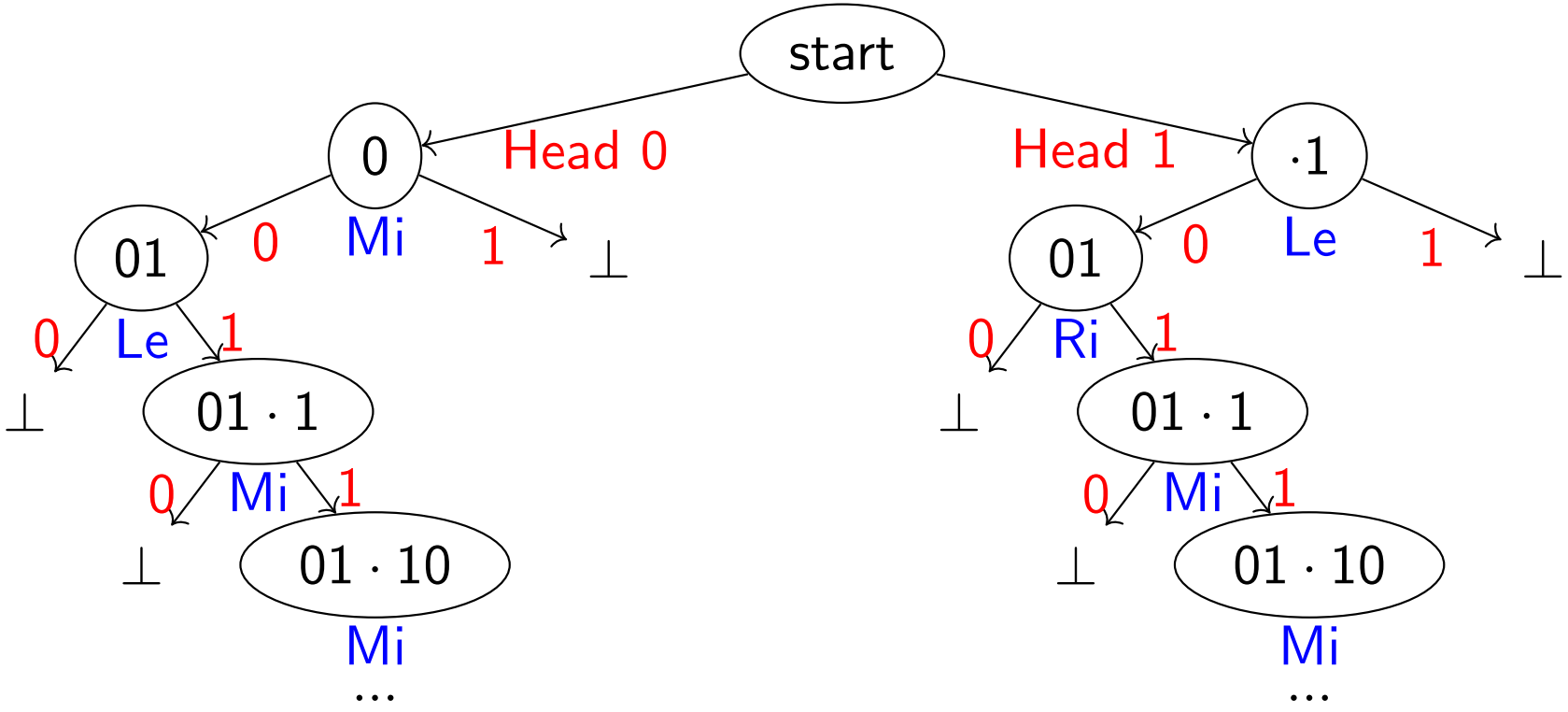
$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Mi}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Le}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

Result of computation of $gc(\bot 10^\omega)$

Gray code of $-1/4$ is $01\bot 10^\omega$.

$$
\begin{aligned}
gc(0:1:\bot:1:[0,0..]) &= \mathbf{Amb}(\mathbf{Mi}(A),\ \mathbf{Le}(B)) \\
\Uparrow \qquad \Uparrow\ (A) &= (\mathbf{Amb}(\mathbf{Le}(C),\ \bot) \\
(B) &= (\mathbf{Amb}(\mathbf{Ri}(D),\ \bot) \\
(C) &\ \ldots
\end{aligned}
$$



$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Mi}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

$$printing(takeD\ 5(gc(0:1:\bot:1:[0,0..]))) = \mathbf{Le}(\mathbf{Ri}(\mathbf{Mi}(\mathbf{Mi}(\mathbf{MiNil}))))$$

Example 2: Pivoting and Gaussian Elimination

Cauchy reals are real numbers satisfying the predicate

$$\mathbf{A}(x) \stackrel{\text{Def}}{=} \forall n \in \mathbf{N}\, \exists q \in \mathbf{Q}\, |x - q| \leq 2^{-n}$$

where $\mathbf{N}(x) \stackrel{\mu}{=} x = 0 \vee \mathbf{N}(x-1)$ and the rational numbers, $\mathbf{Q}$, are defined from $\mathbf{N}$ in the usual way.

$$n \,\mathbf{r}\, \mathbf{N}(x) \quad \Leftrightarrow \quad x \text{ is a natural number with unary representation } n$$

$$f \,\mathbf{r}\, A(x) \quad \Leftrightarrow \quad f \text{ is a fast rational Cauchy sequence converging to } x$$

$$x \# y \stackrel{\text{Def}}{=} \exists k \in \mathbf{N}\, |x - y| \geq 2^{-k}$$

[Pivoting] $\forall n \in \mathbf{N} \forall (x_0, \ldots, x_n) \in \mathbf{A}^{n+1} \setminus \{0^{n+1}\}\ \mathbf{S}_n(\exists i \leq n\, x_i \# 0)$

The extracted program scans (realizers of) Cauchy reals $x_0, \ldots, x_n$ concurrently for evidence of non-zeroness.

printing (takeD 30 (pivot 5 (mkD $[1/2^{22}, 0, 0, 1/2^5, 0]$))) = (1, 24)
printing (takeD 30 (pivot 5 (mkD $[1/2^{22}, 0, 0, 1/2^5, 0]$))) = (4, 7)
Can be used to extract a concurrent program for Gaussian elimination.

References

[B 2016] U. Berger. Extracting Non-Deterministic Concurrent Programs. In 25th EACSL Annual Conference on Computer Science Logic (CSL 2016), volume 62 of LIPIcs, pages 26:126:21, 2016.

[B 2010] U. Berger. Realisability for induction and coinduction with applications to constru]ctive analysis. Jour. Universal Comput. Sci., 16(18):2535 2555, 2010.

[T 2002] H. Tsuiki. Real Number Computation through Gray Code Embedding. Theoretical Computer Science, 284(2):467485, 2002.

[Tatsuta 1998] M. Tatsuta. Realizability of monotone coinductive definitions and its application to program synthesis. In Mathematics of Program Construction, volume 1422 of Lecture Notes in Mathematics, pages 338364. Springer, 1998.

[Schwichtenberg 1991] H. Schwichtenberg. Minimal logic for computable functions. NATO Advanced Study Institute, International Summer School held in Marktoberdorf, 23.Juli 1991 - 04. August 1991, Marktoberdorf, Germany.