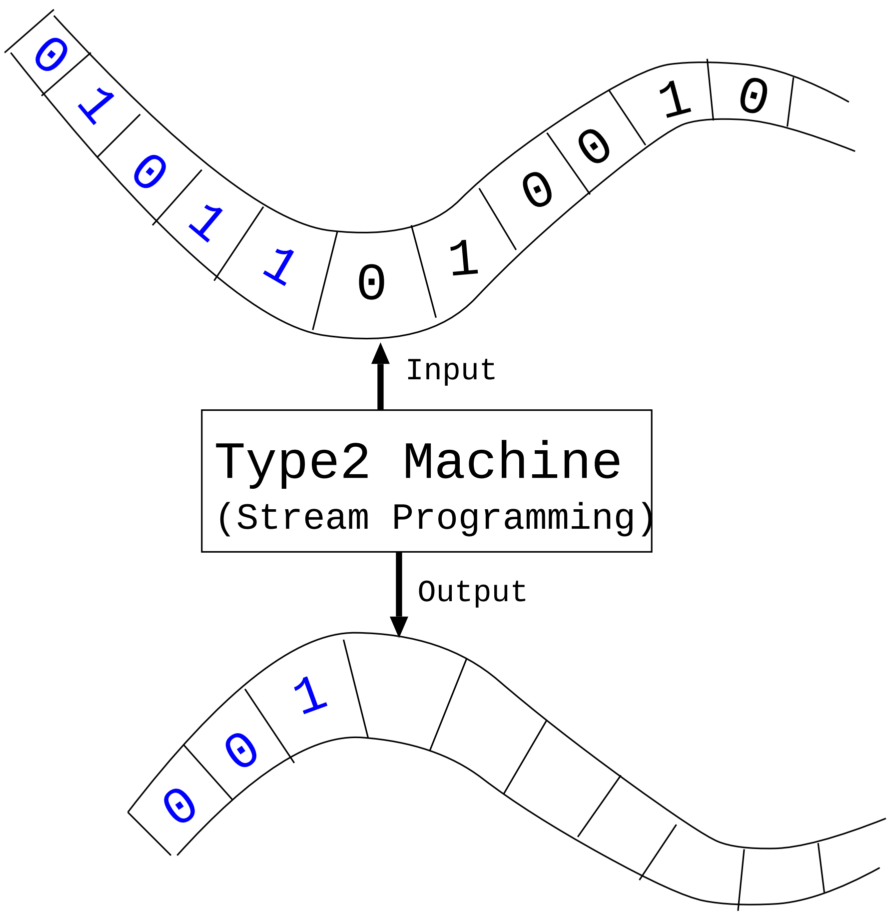
Computation over Topological Spaces via Embeddings in Streams with a Bottom

Hideki Tsuiki (Kyoto University)

The 7th Workshop on Learning with Logic and Logics for Learning (LLLL)

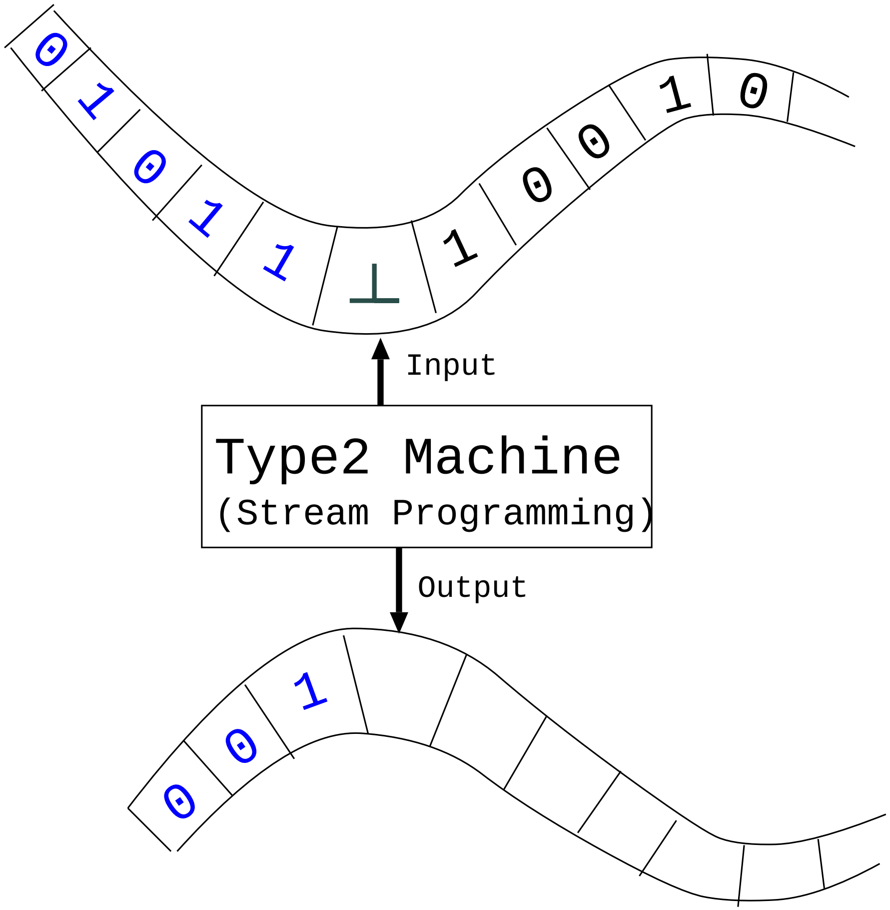
2011/3/29-30, Osaka

Stream Programming



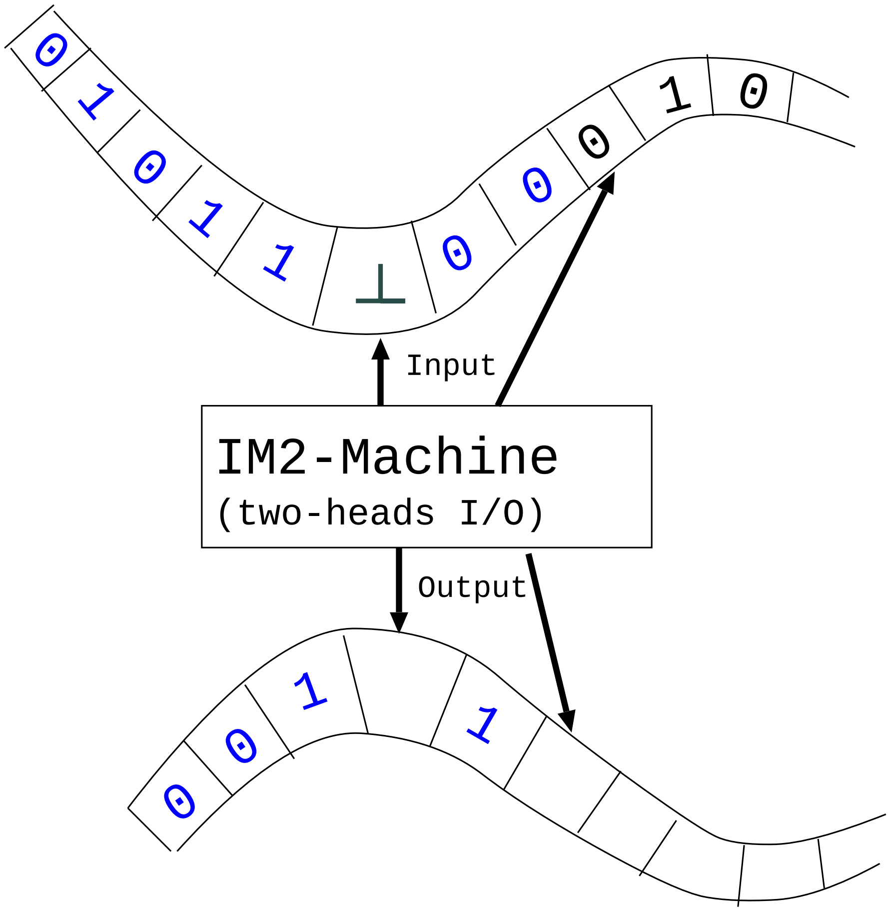
- Type2-machine: extension of Turing Machine so that the input/output tape have infinite length. [Weihrauch, et al.]
- Program with stream input/output.

Stream with a bottom



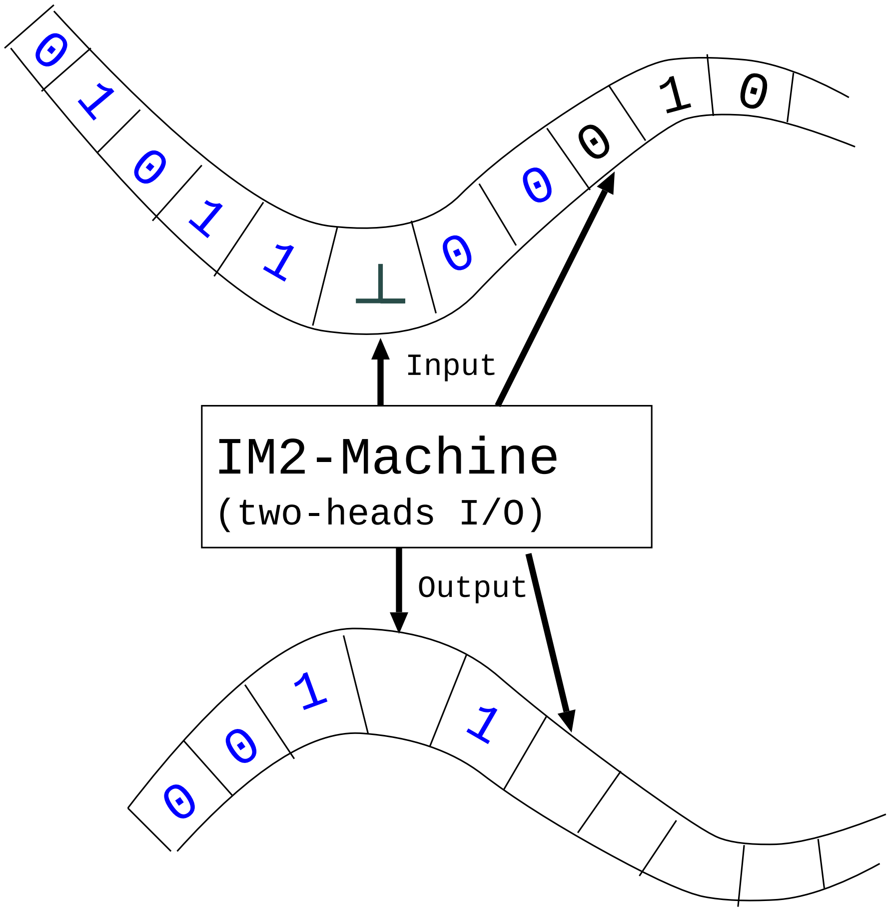
- If a bottom cell $\perp$ exists in the input, a Type2 machine get stuck and cannot read the rest of the input.
- $\perp$: Non-terminating computing.
- In Haskell, an expression of type Bool may have the value $\perp$ and a sequence in [Bool] may contain $\perp$.

Solution



Multiple head machine.

Solution

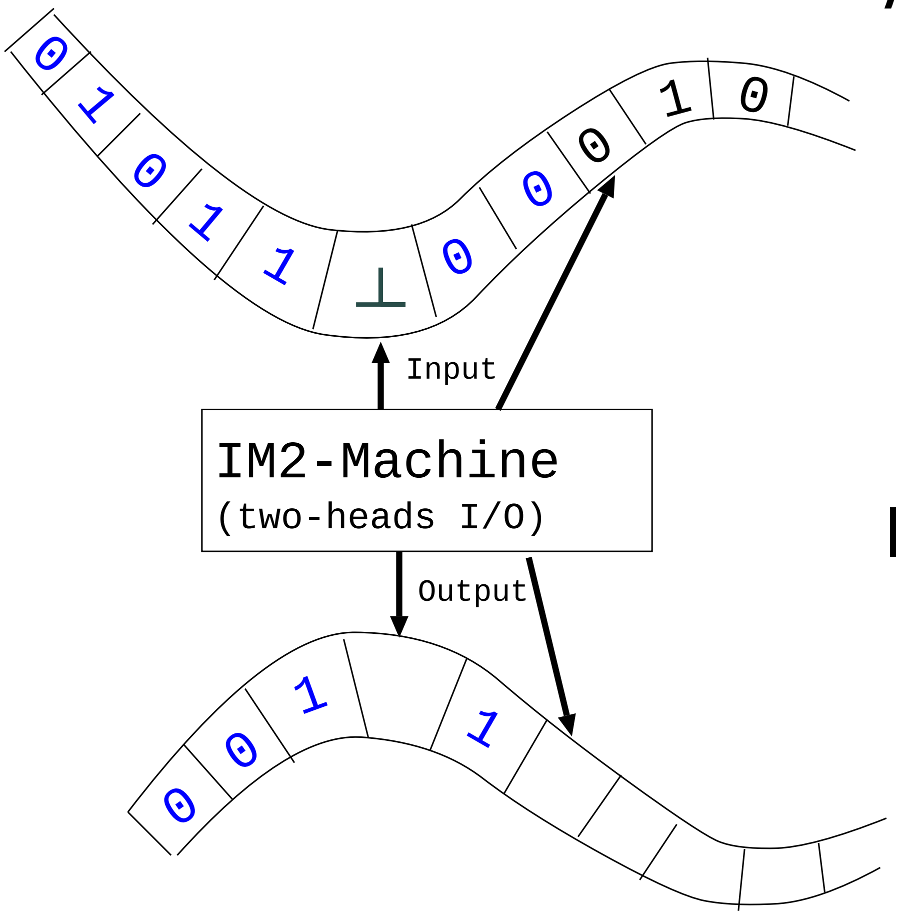


Application:

- Real Number Computation.
- Representation of Topological Spaces.

Multiple head machine.

Solution



Multiple head machine.

Application:

- Real Number Computation.
- Representation of Topological Spaces.

Implemented

- in GHC: Logic programming language with committed choice.
- Extension of Haskell, by modifying Hags system.

Part one:

Representation of Reals as bottomed sequences.

Injective Coding of $\mathbf{I}$ in $\{0,1\}^\omega$.

- $\Sigma = \{0,1\}$.
- Consider a unique coding of $\mathbf{I} = [0,1]$ in Σ^ω. That is, an injective function φ from $\mathbf{I}$ to Σ^ω.
- φ and its inverse should be continuous (i.e. φ is an embedding) because real number computation we consider is the limit of approximation intervals
$$(a_0, b_0) \supset (a_1, b_1) \supset (a_2, b_2) \supset \ldots \to x$$
and it should be implemented as extension of words
$$p_0 \to p_0p_1 \to p_0p_1p_2 \to \ldots \to \varphi(x).$$

Injective Coding of $\mathbf{I}$ in $\{0,1\}^\omega$.

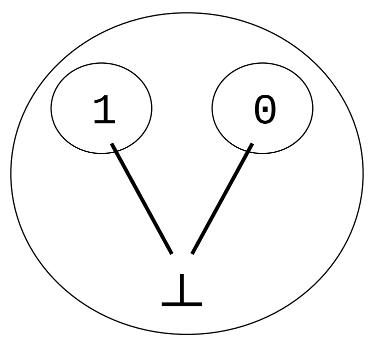
- $\Sigma = \{0, 1\}$.
- Consider a unique coding of $\mathbf{I} = [0, 1]$ in Σ^ω. That is, an injective function φ from $\mathbf{I}$ to Σ^ω.
- φ and its inverse should be continuous (i.e. φ is an embedding) because real number computation we consider is the limit of approximation intervals
$$(a_0, b_0) \supset (a_1, b_1) \supset (a_2, b_2) \supset \ldots \to x$$
and it should be implemented as extension of words
$$p_0 \to p_0p_1 \to p_0p_1p_2 \to \ldots \to \varphi(x).$$
- Impossible to embed $\mathbf{I}$ in Σ^ω (Cantor Space).
- $\mathbf{I}$ is connected, but Σ^ω is totally disconnected.
- Impossible to injectively code $\mathbf{I}$ in Σ^ω.

Gray coding of $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$.

- However, it is possible to embed $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$ by the Gray-code embedding.[Gianantonio],[T]

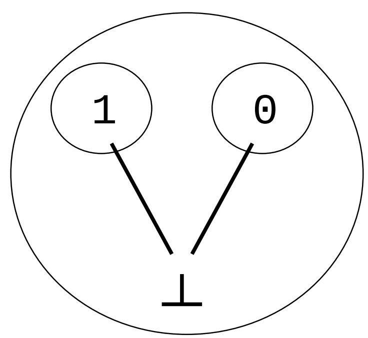
Gray coding of $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$.

- However, it is possible to embed $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$ by the Gray-code embedding.[Gianantonio],[T]
- $\Sigma^{\omega}_{\perp,1} \subset (\Sigma \cup \{\perp\})^{\omega}$ (i.e. Plotkin's T_{ω}:)
  Infinite sequences of $\Sigma \cup \{\perp\}$ with at most one $\perp$, with the subspace topology of ${\Sigma_{\perp}}^{\omega}$ (with the Scott topology)
  ex. $010\perp1000\ldots, 00110011\ldots$.
- topology of $\Sigma_{\perp}$:



Gray coding of $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$.

- However, it is possible to embed $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$ by the Gray-code embedding.[Gianantonio],[T]
- $\Sigma^{\omega}_{\perp,1} \subset (\Sigma \cup \{\perp\})^{\omega}$ (i.e. Plotkin's T_{ω}:)
  Infinite sequences of $\Sigma \cup \{\perp\}$ with at most one $\perp$, with the subspace topology of ${\Sigma_{\perp}}^{\omega}$ (with the Scott topology)
  ex. $010\perp1000\ldots, 00110011\ldots$.
- topology of $\Sigma_{\perp}$:



Gray code of Natural numbers

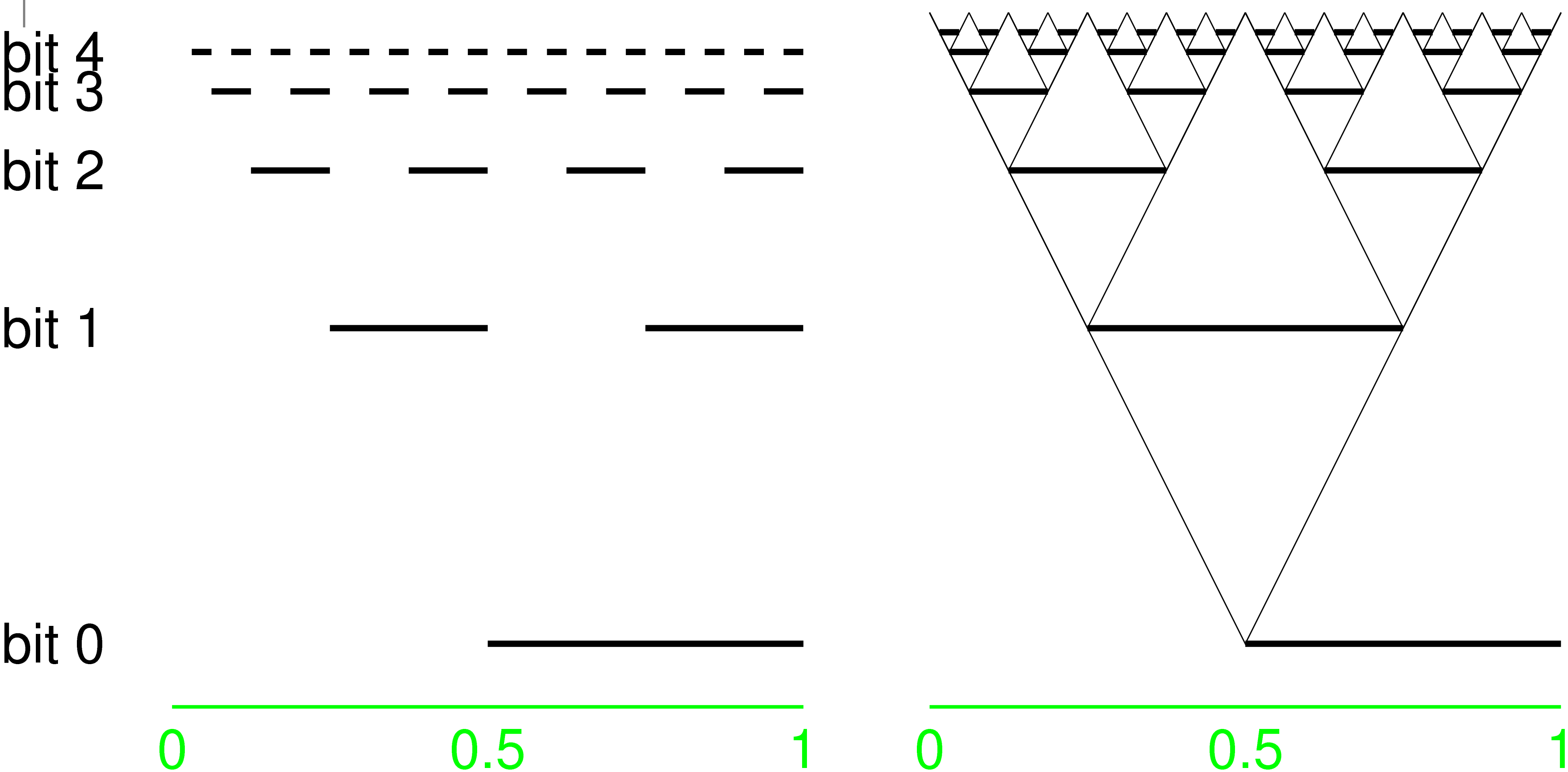
(Binary reflected) Gray code: another code of natural numbers with $\Sigma = \{0, 1\}$.

number	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100

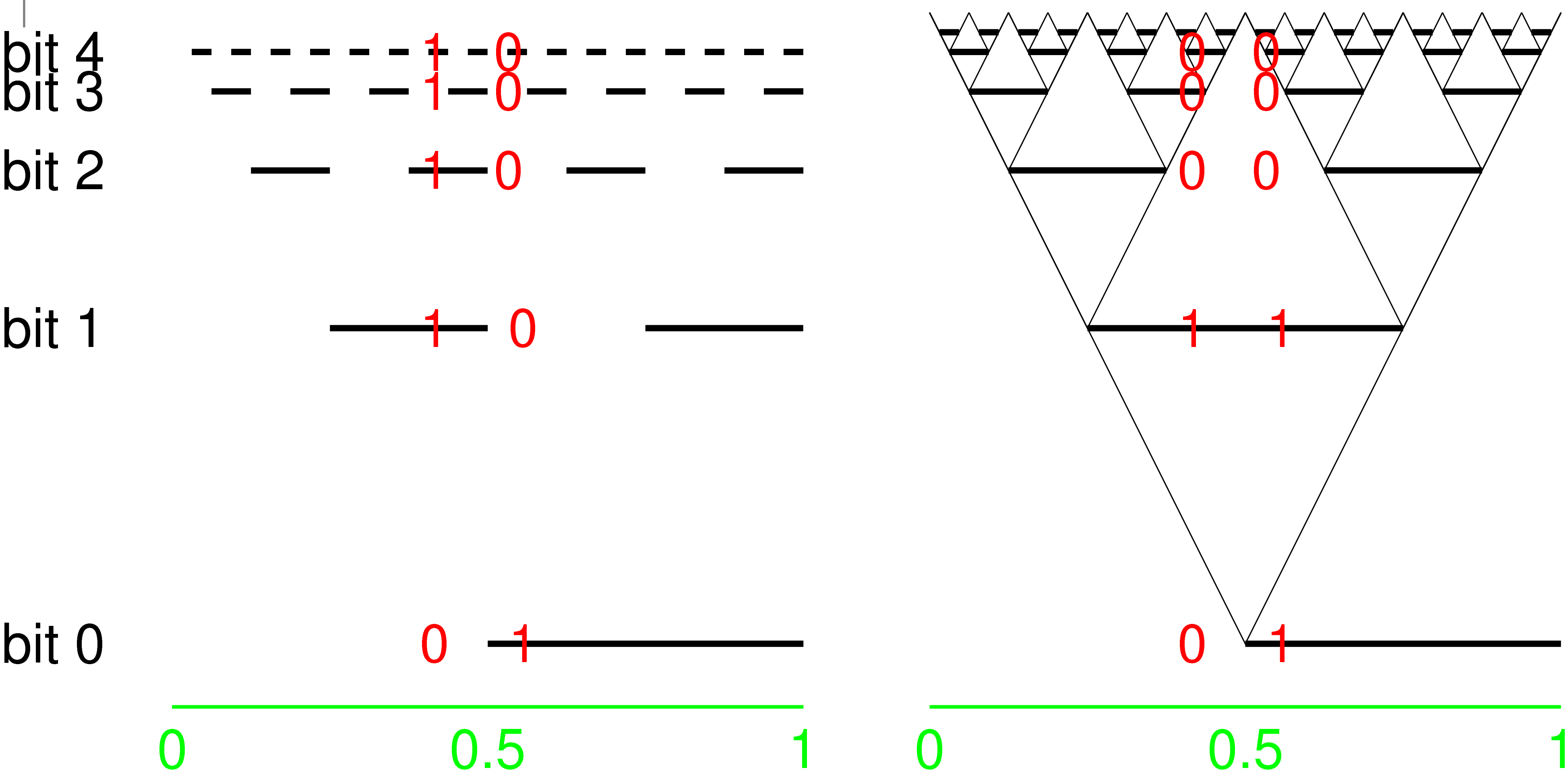
- Only one bit changes by the increment operation.
- Conversion from ordinary binary code to the Gray code: one-bit shift and xor.

```
conv s = map xor
    (zip s (0:s))
```

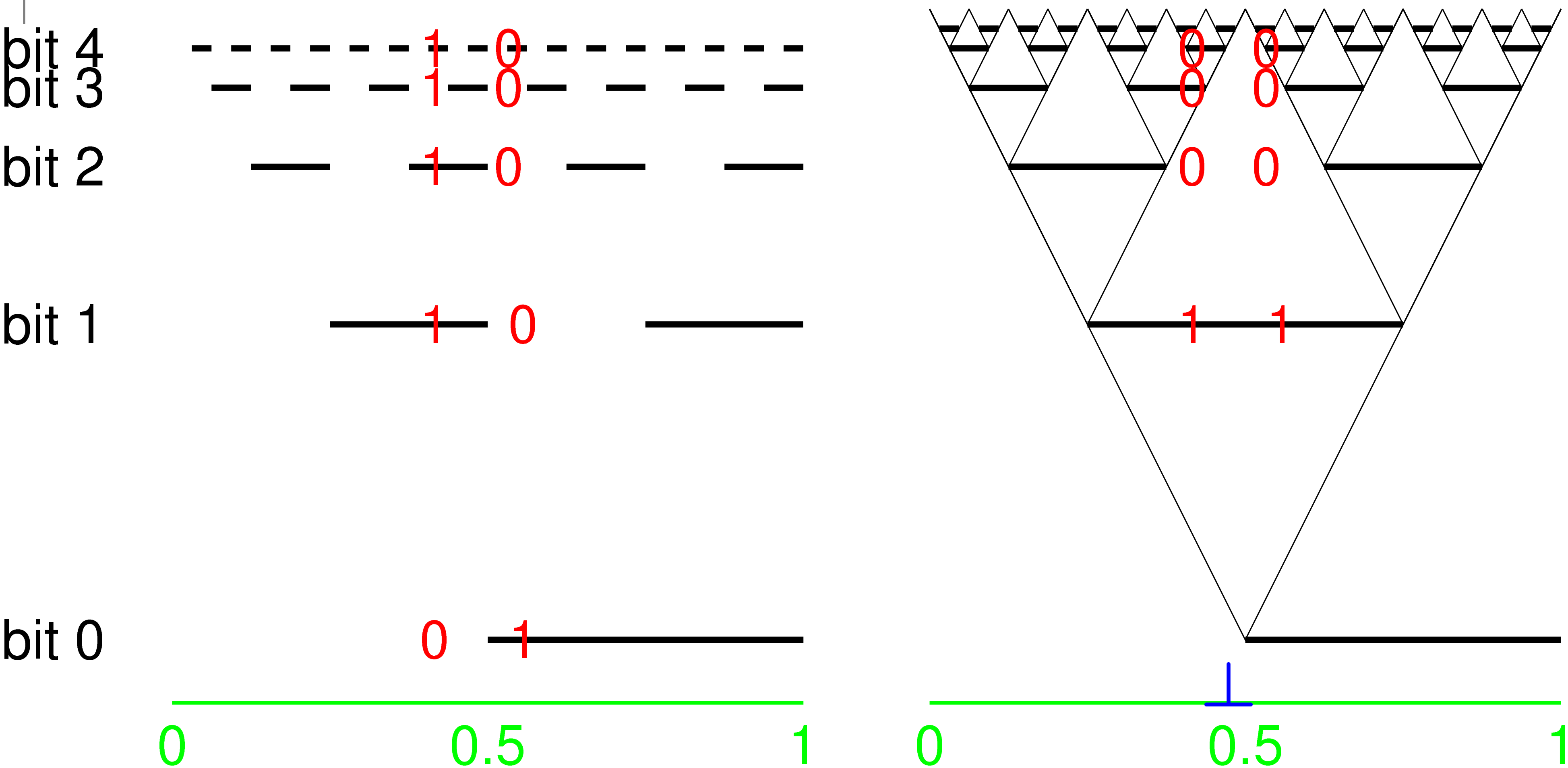
Binary/Gray expansion of I = [0,1]



Binary/Gray expansion of I = [0,1]

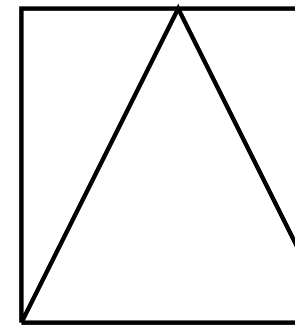


Binary/Gray expansion of I = [0,1]



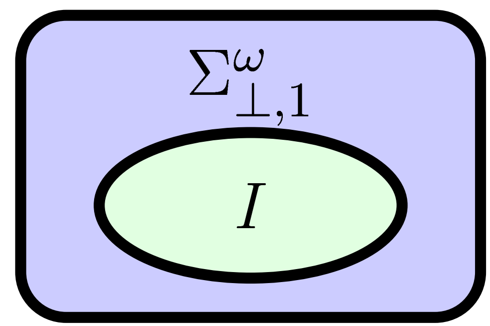
Gray embedding of $\mathbf{I}$ in $\Sigma^{\omega}_{\perp,1}$

$$t : I \to I,\ \ t(x) = \begin{cases} 2x & (0 \leq x \leq 1/2) \\ 2(1-x) & (1/2 < x \leq 1) \end{cases}.$$



$$\varphi_G : \mathbf{I} \to \Sigma^{\omega}_{\perp,1}\ \varphi_G(x)(n) = \begin{cases} 0 & (t^n(x) < {}^1\!/_2) \\ \perp & (t^n(x) = {}^1\!/_2) \\ 1 & (t^n(x) > {}^1\!/_2) \end{cases}.$$

We call φ_G the Gray embedding.



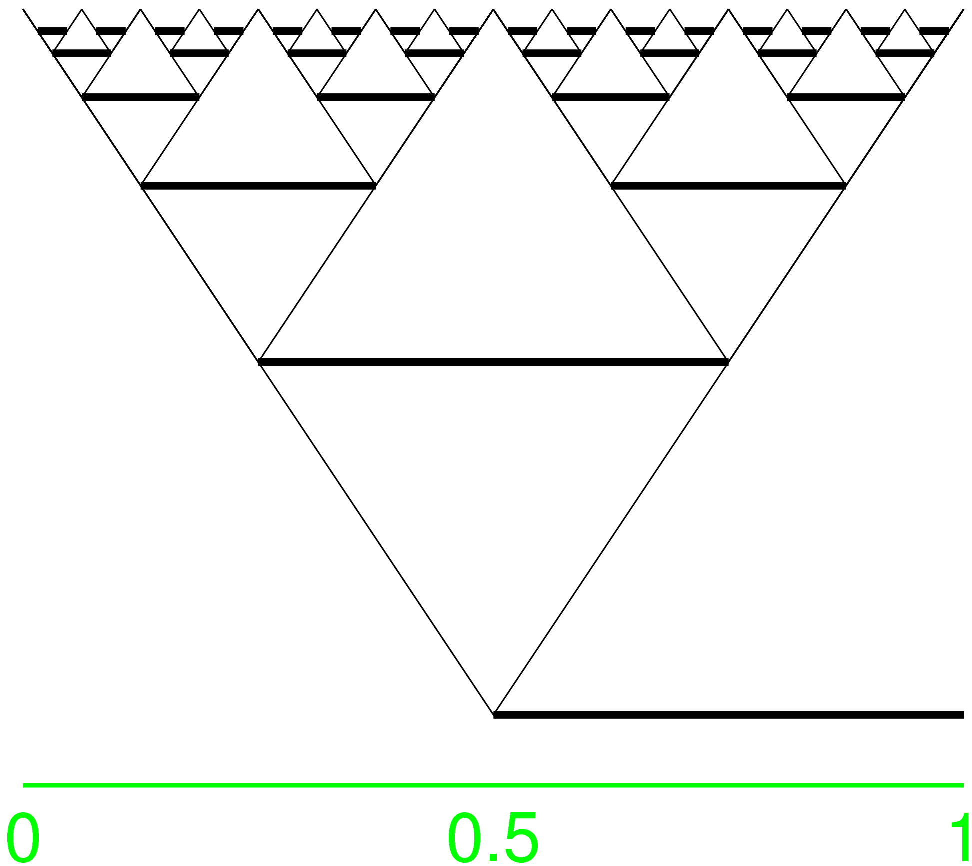
- Itinerary of the tent function.
- Topological embedding of I in $\Sigma^{\omega}_{\perp,1}$.
- Continuously changing code.
- Can be used to define computation over I (or R) with IM2-machines.

Part two:

IM2-machines and their implementations.

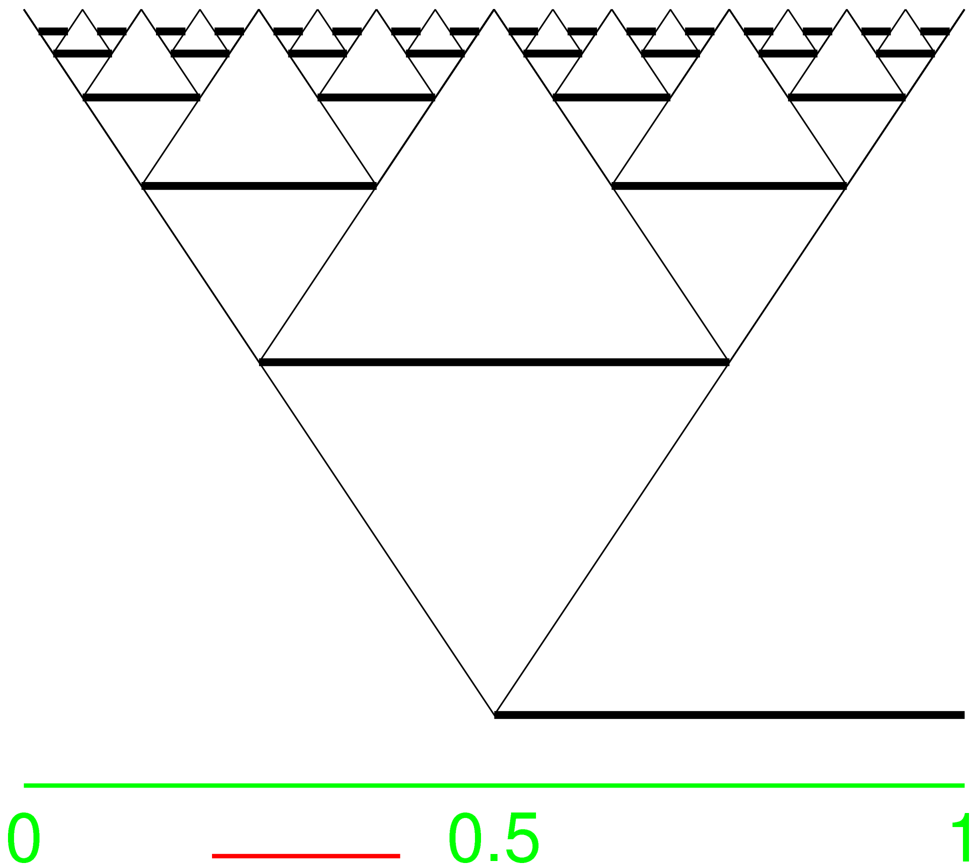
How can we input/output Gray-code?

Real number computation as the limit of approximations (shrinking open intervals).



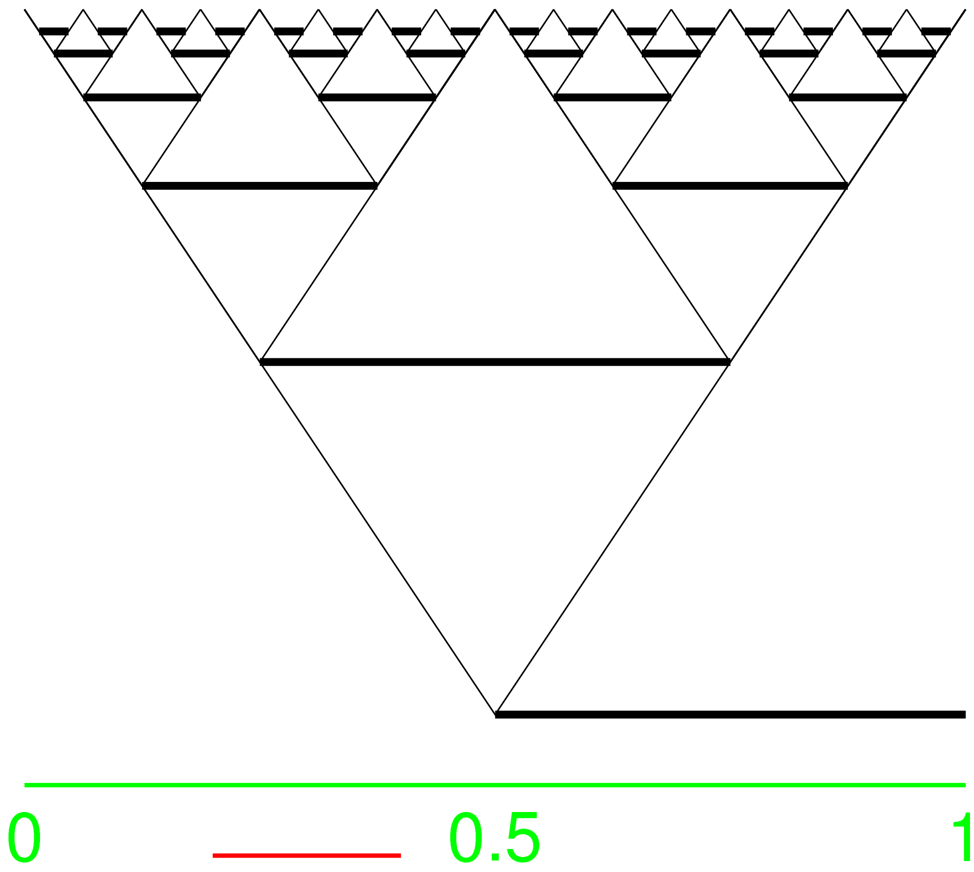
How can we input/output Gray-code?

Real number computation as the limit of approximations (shrinking open intervals).



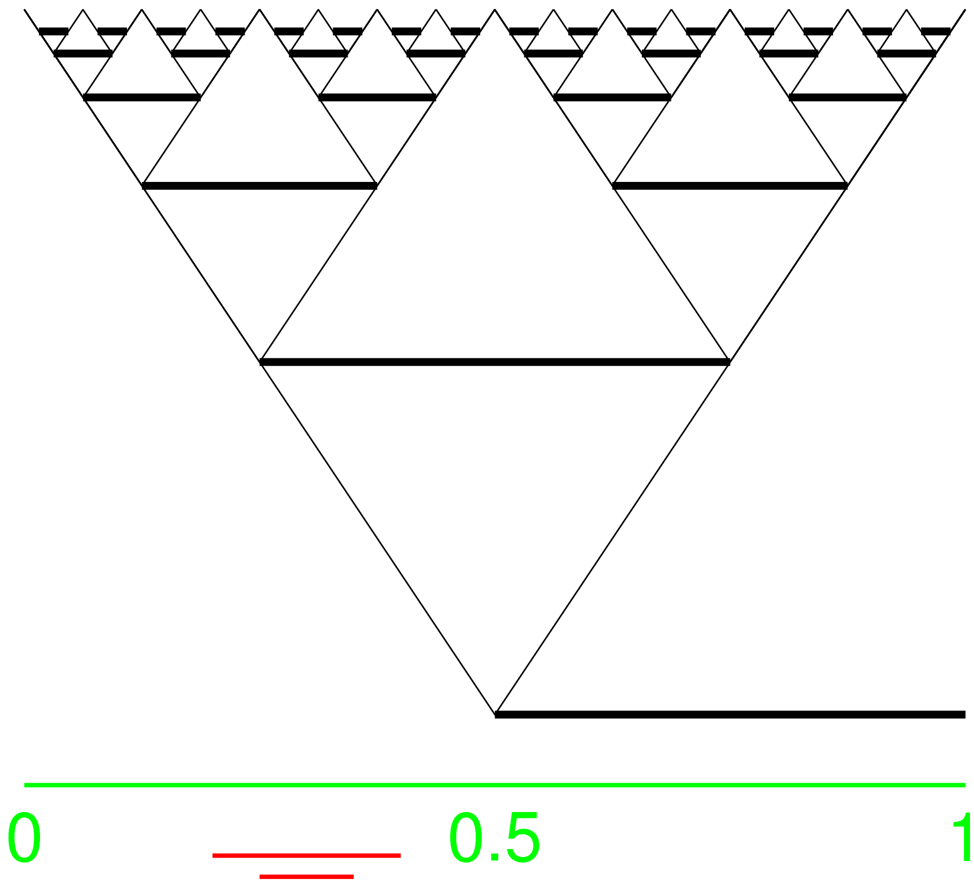
How can we input/output Gray-code?

Real number computation as the limit of approximations (shrinking open intervals).



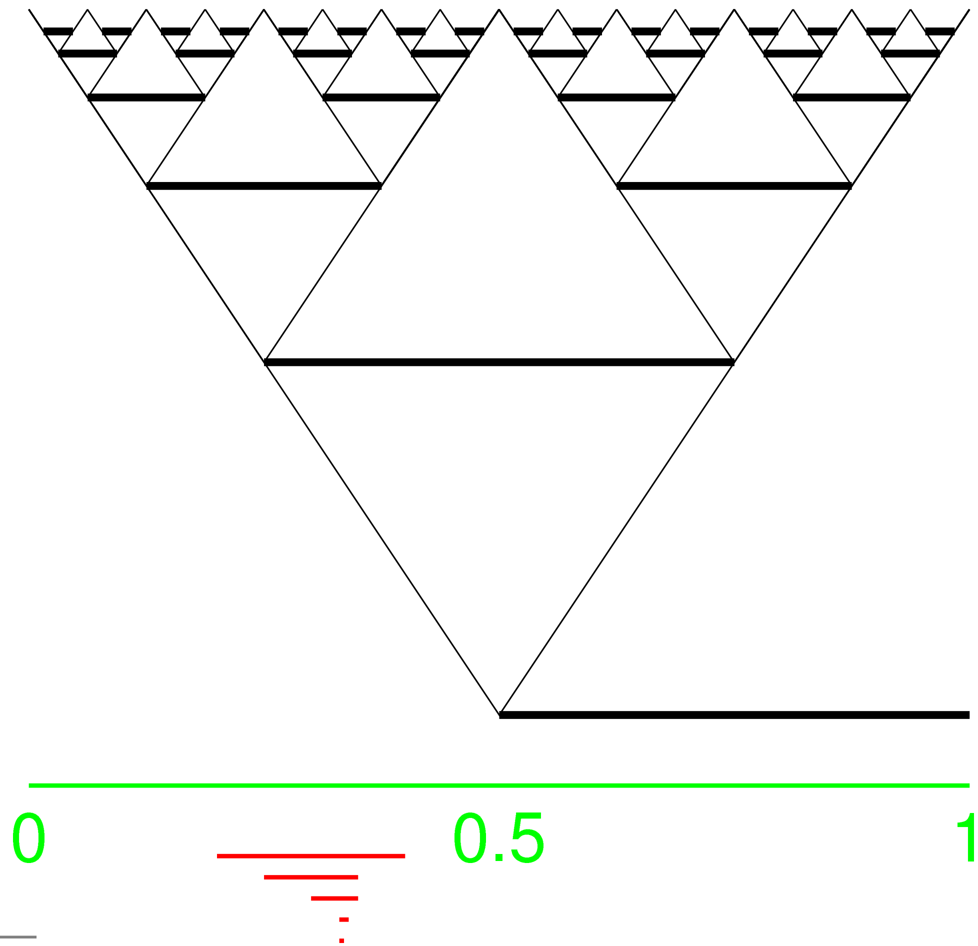
How can we input/output Gray-code?

Real number computation as the limit of approximations (shrinking open intervals).



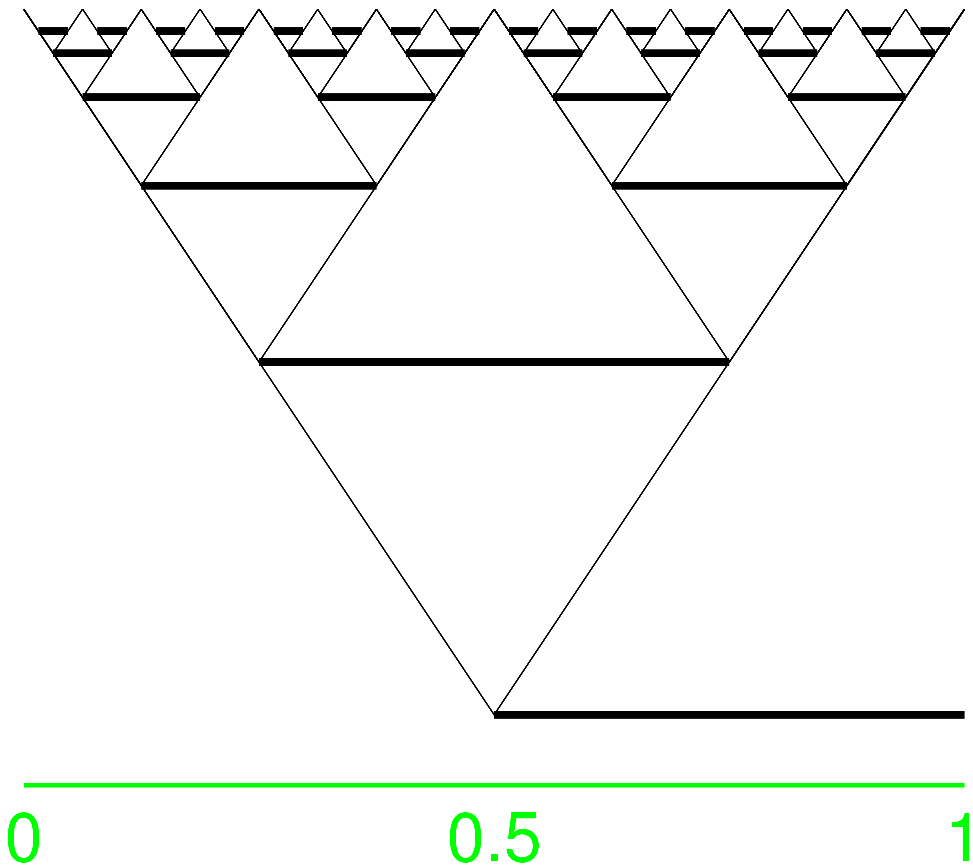
How can we input/output Gray-code?

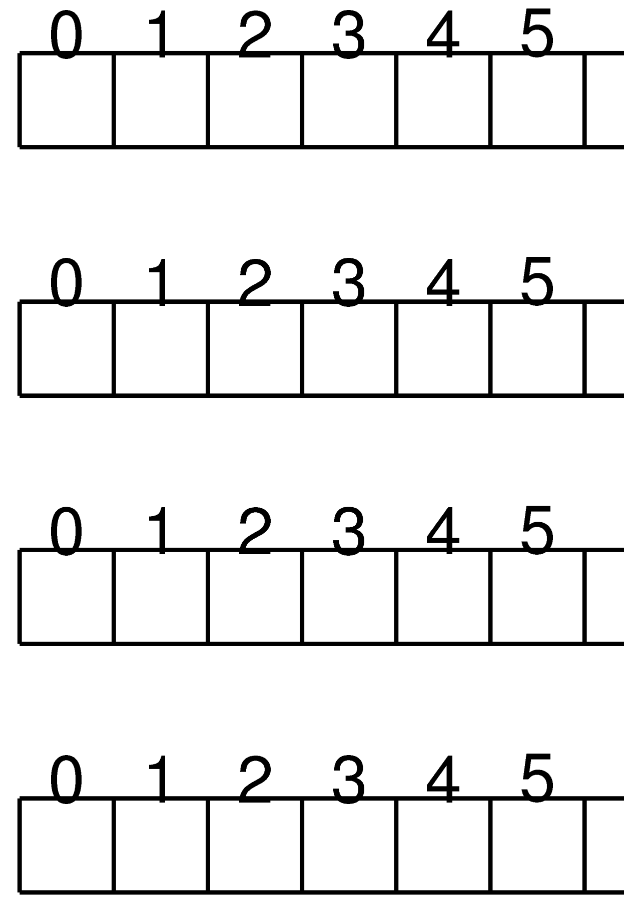
Real number computation as the limit of approximations (shrinking open intervals).



How can we output Gray-code?

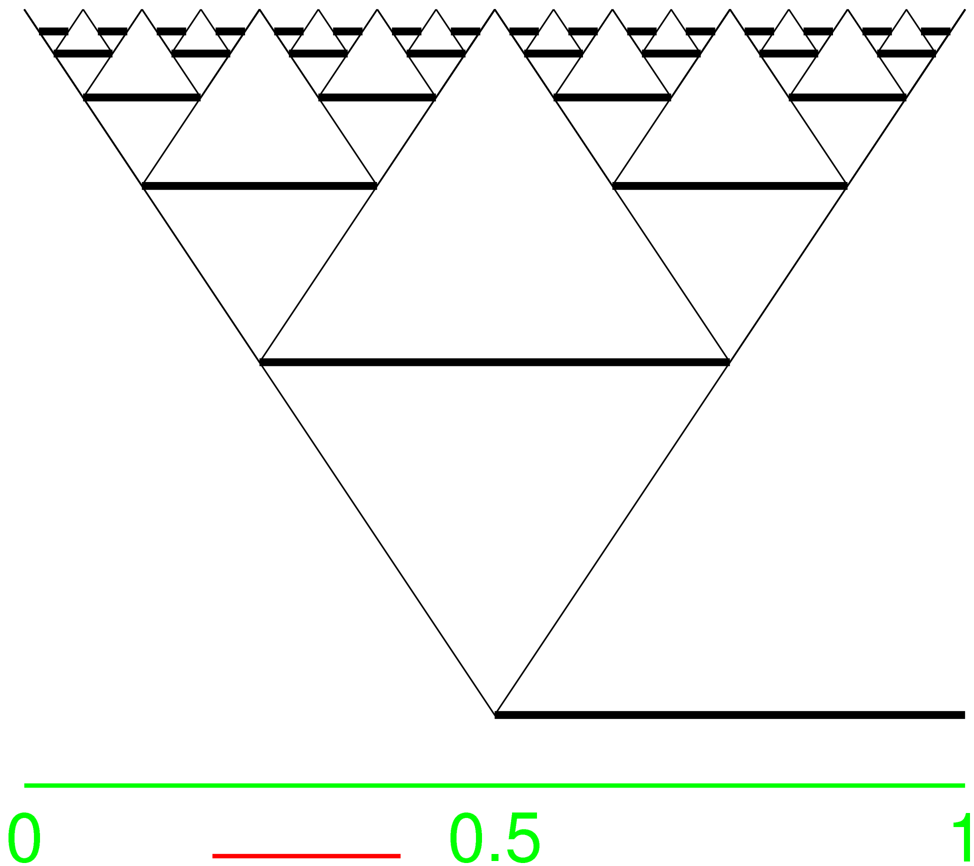
Real number computation as a limit of approximations (shrinking open intervals).

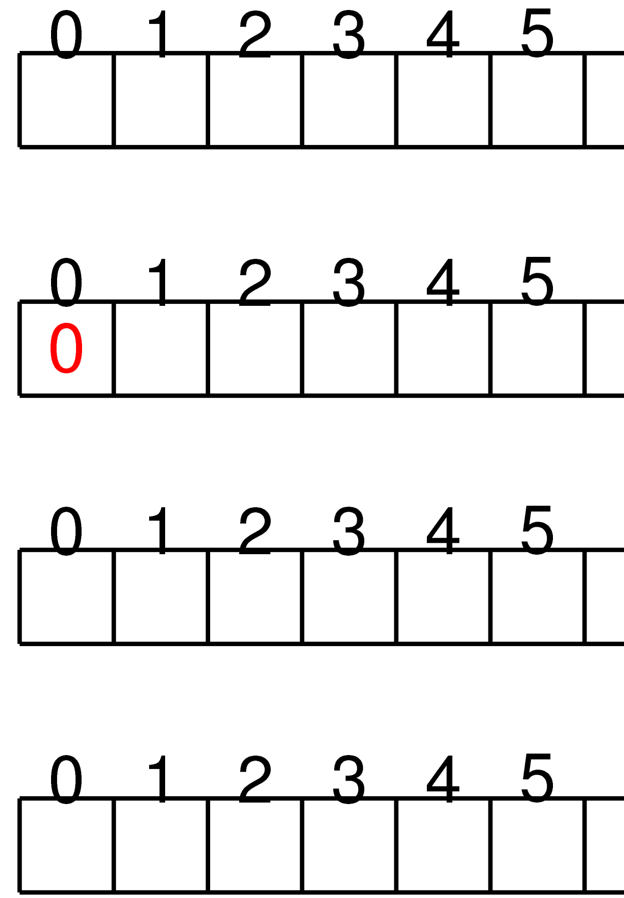




How can we output Gray-code?

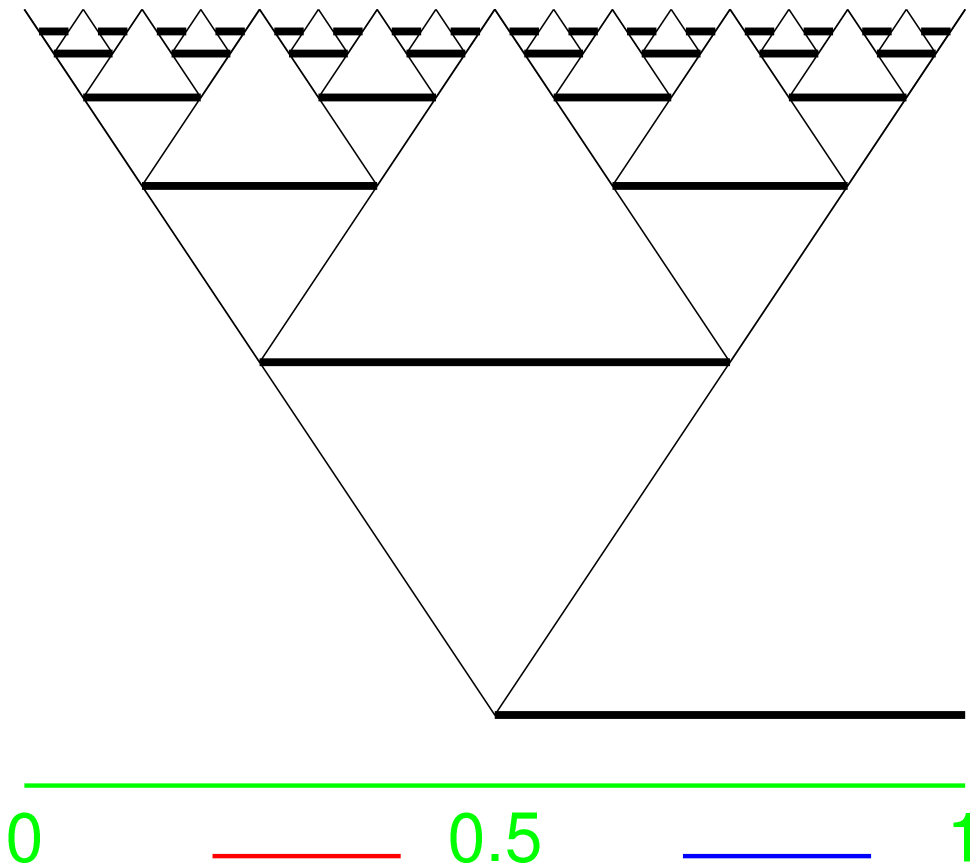
Real number computation as a limit of approximations (shrinking open intervals).

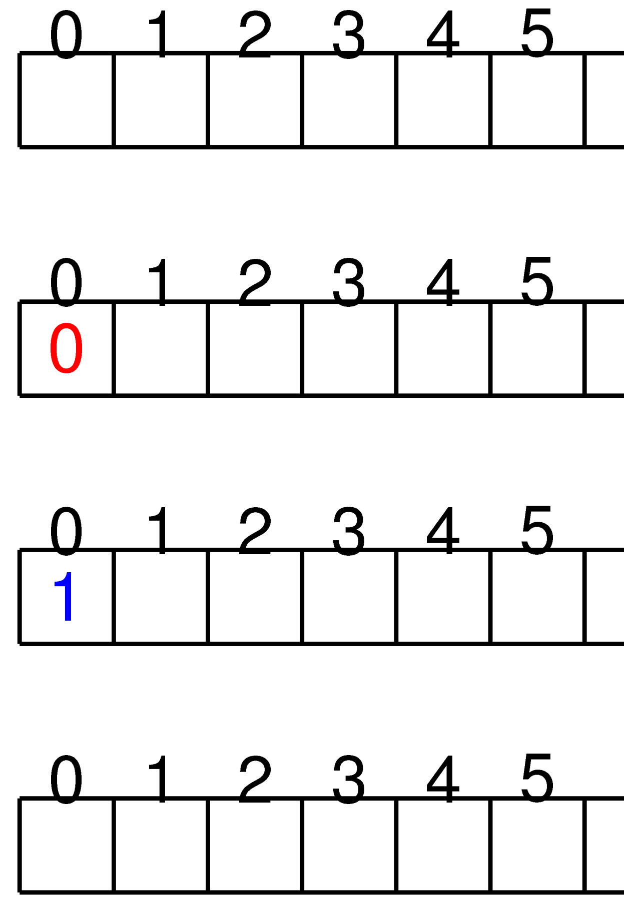




How can we output Gray-code?

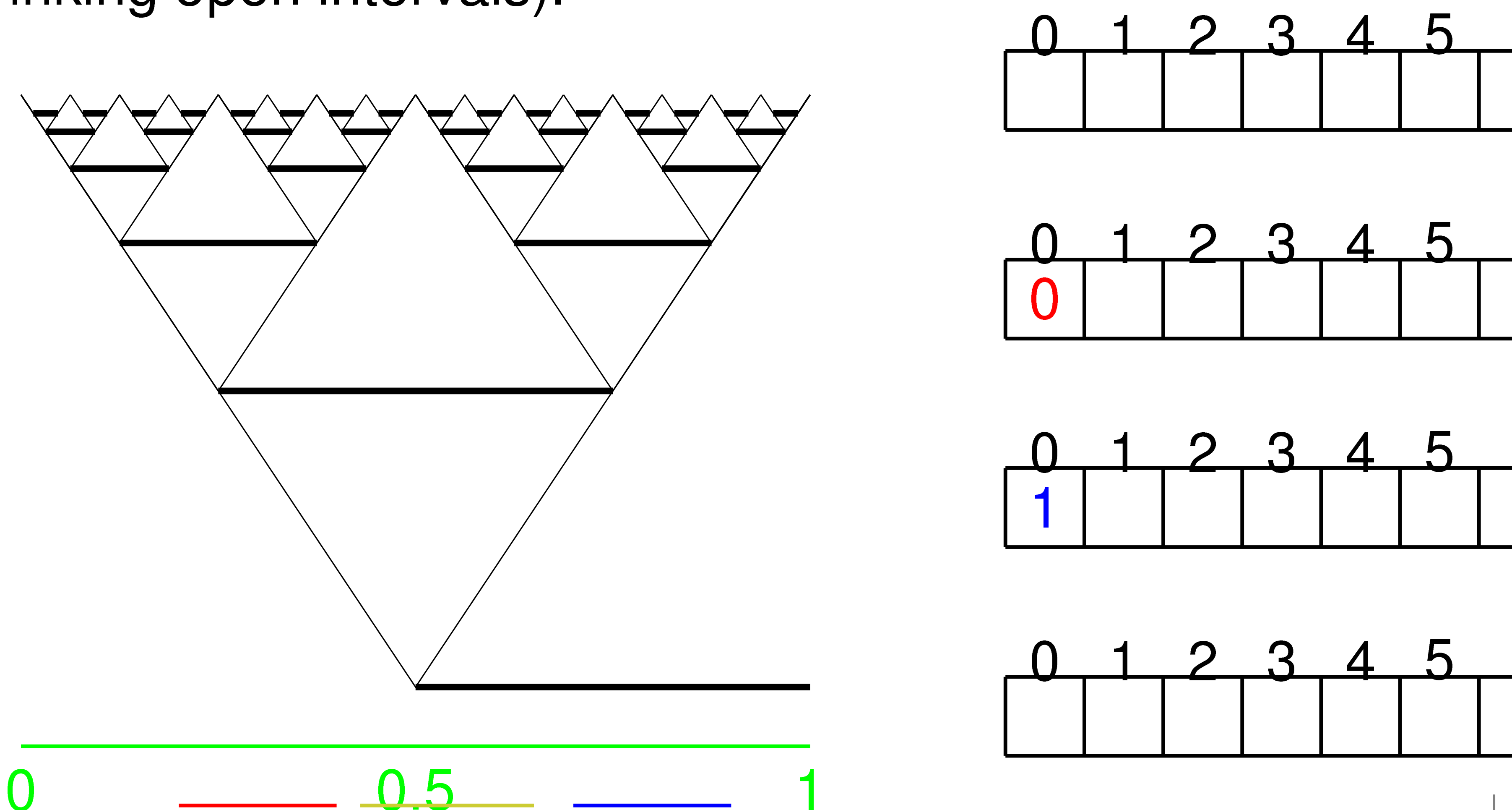
Real number computation as a limit of approximations (shrinking open intervals).





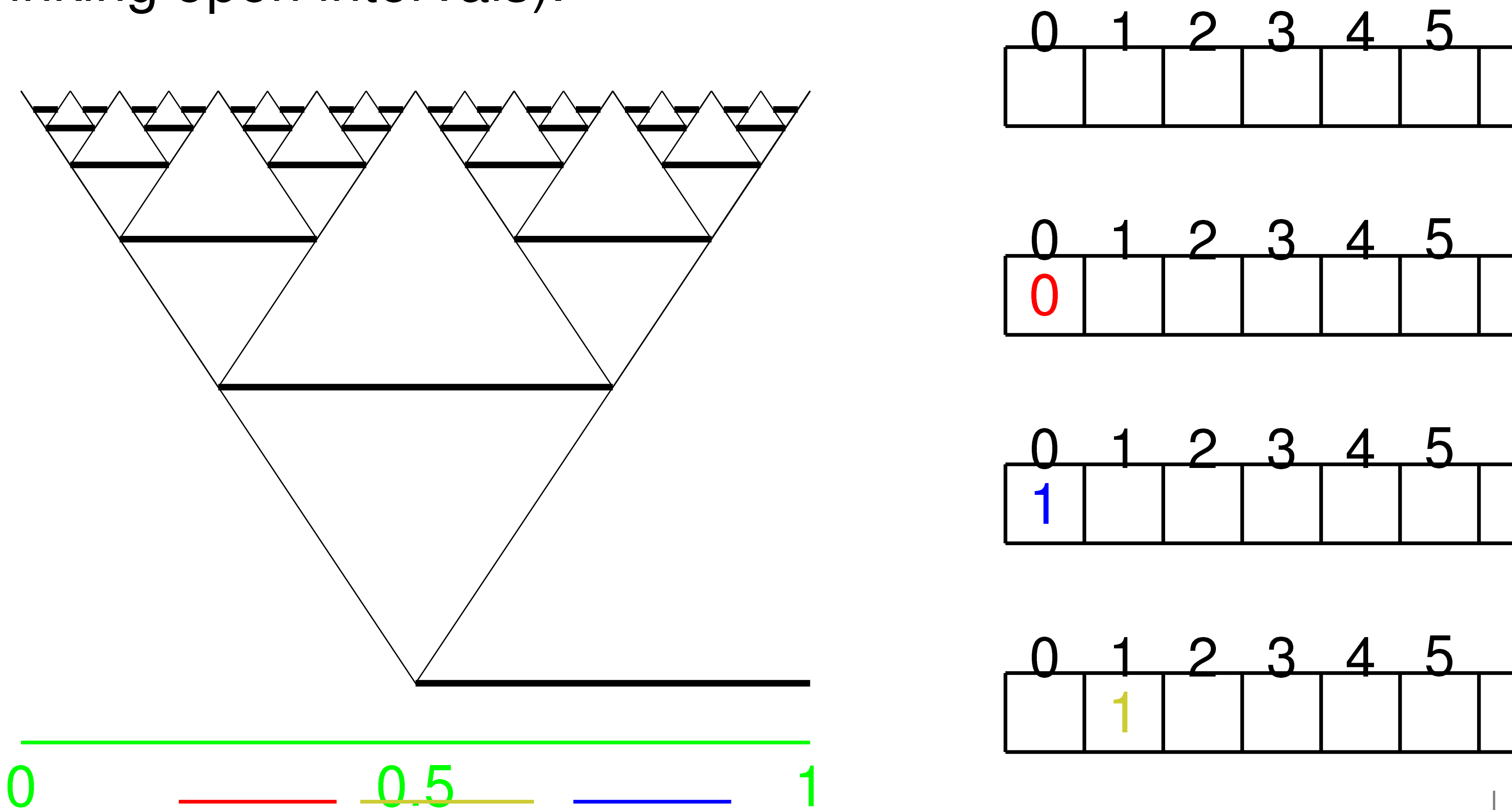
How can we output Gray-code?

Real number computation as a limit of approximations (shrinking open intervals).



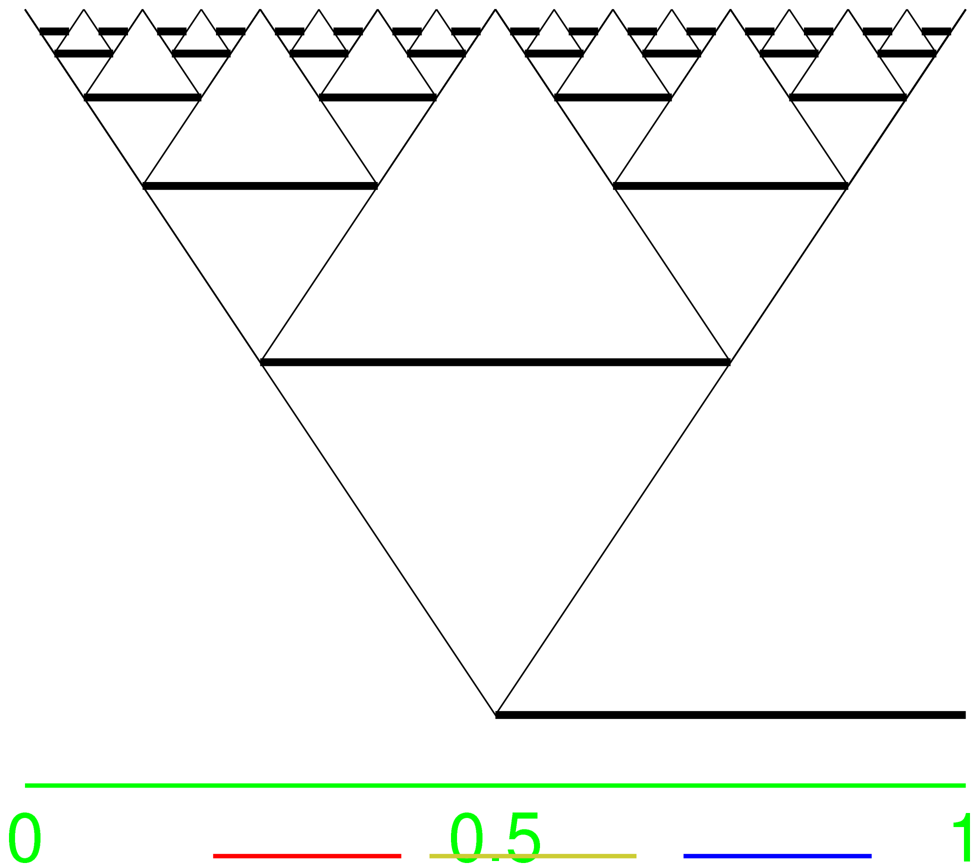
How can we output Gray-code?

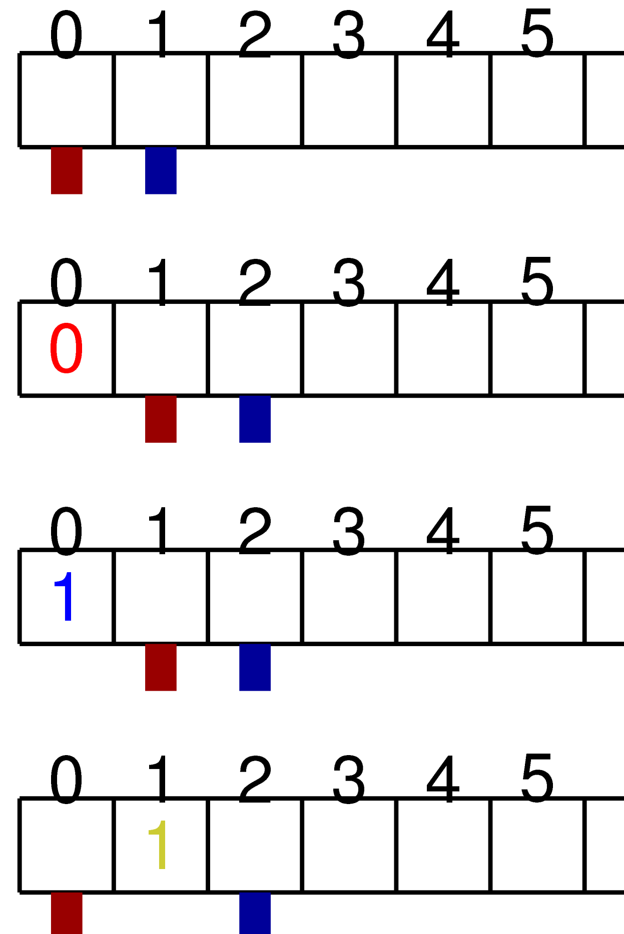
Real number computation as a limit of approximations (shrinking open intervals).



How can we output Gray-code?

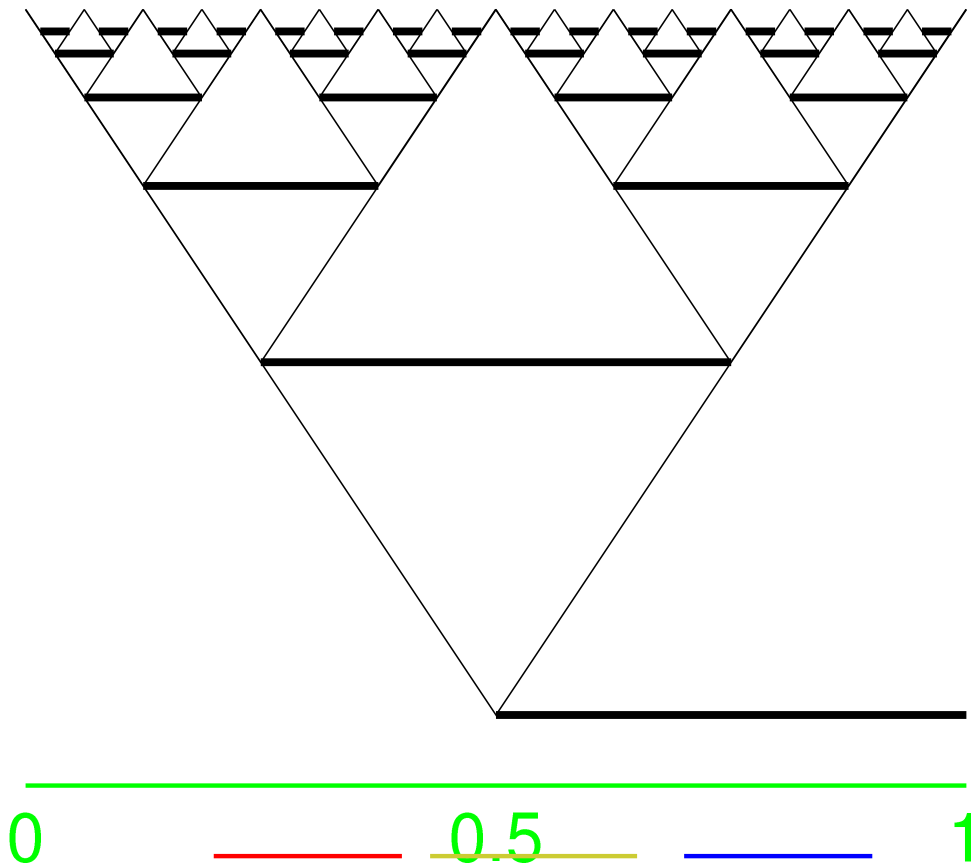
Real number computation as a limit of approximations (shrinking open intervals).

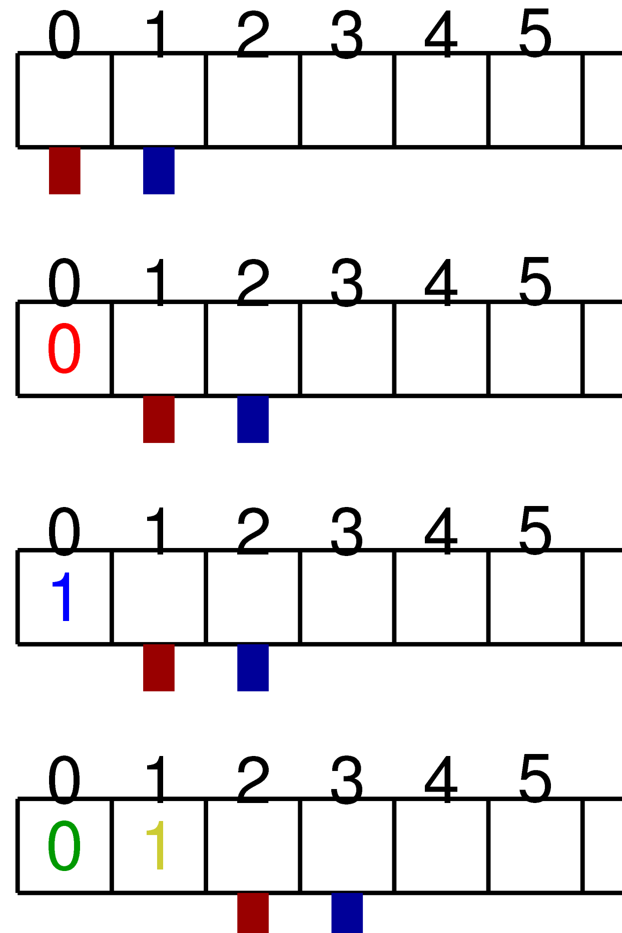




How can we output Gray-code?

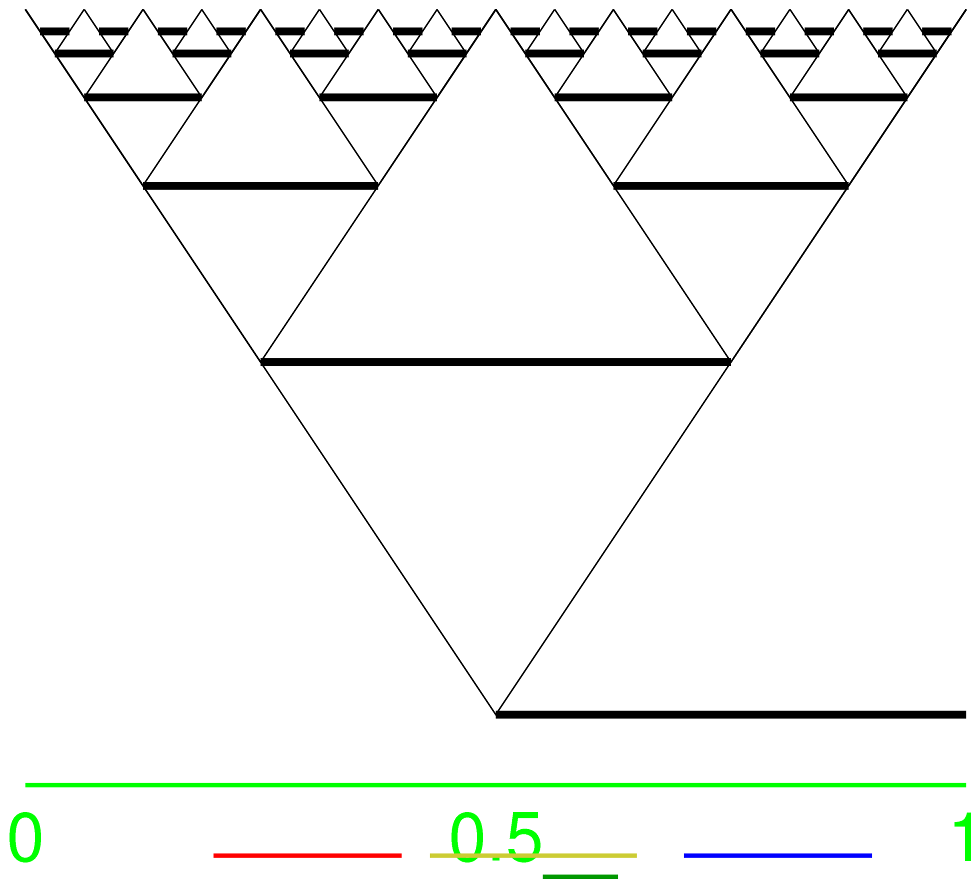
Real number computation as a limit of approximations (shrinking open intervals).

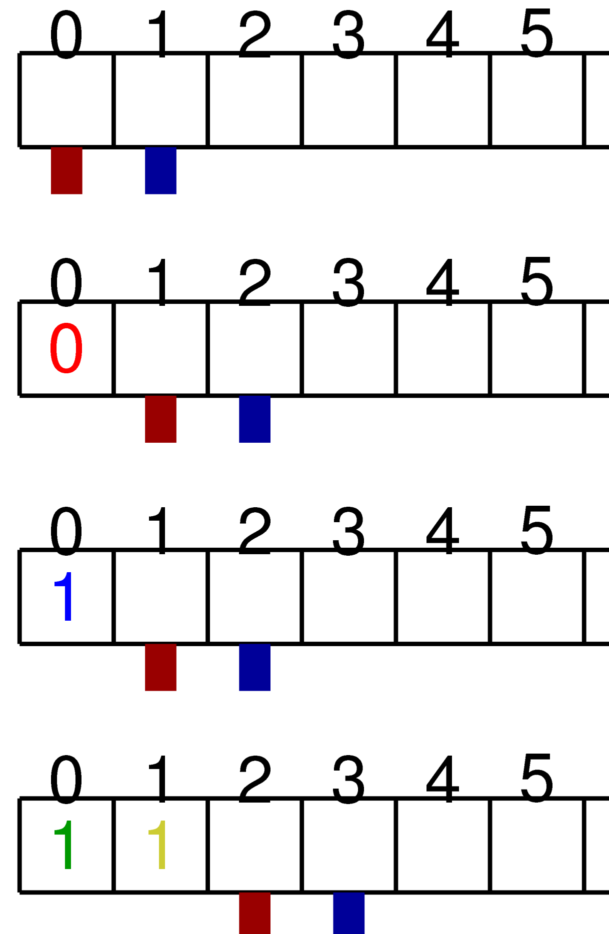




How can we output Gray-code?

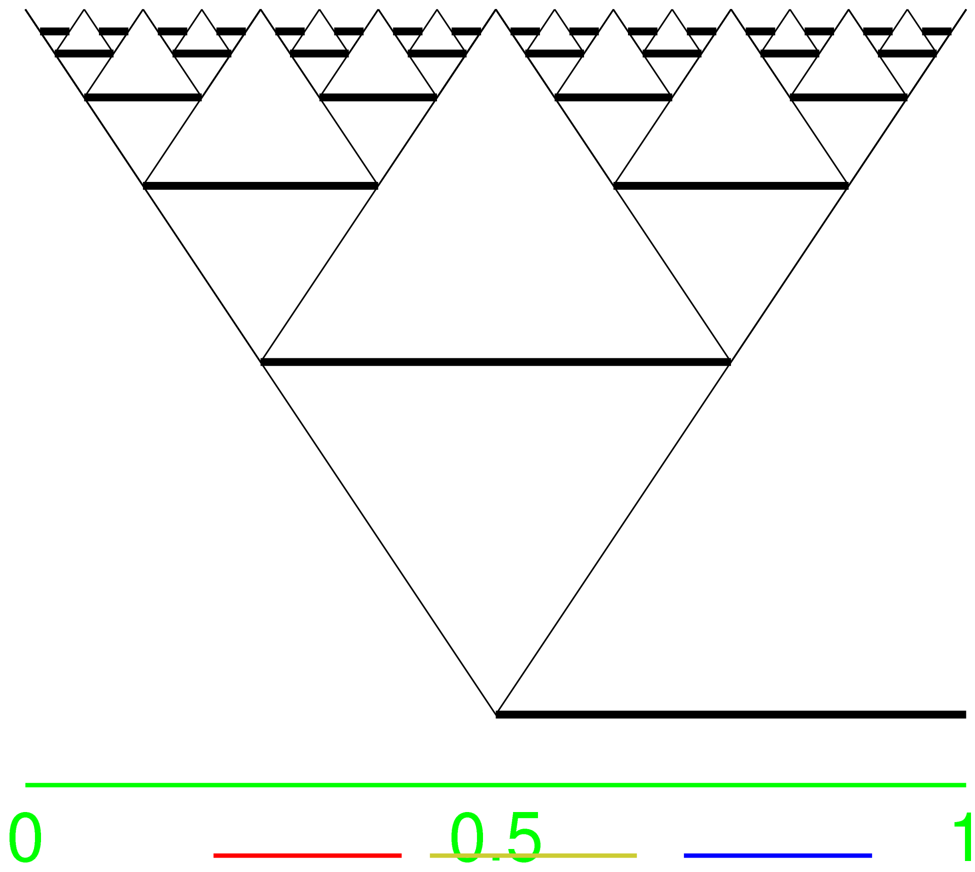
Real number computation as a limit of approximations (shrinking open intervals).

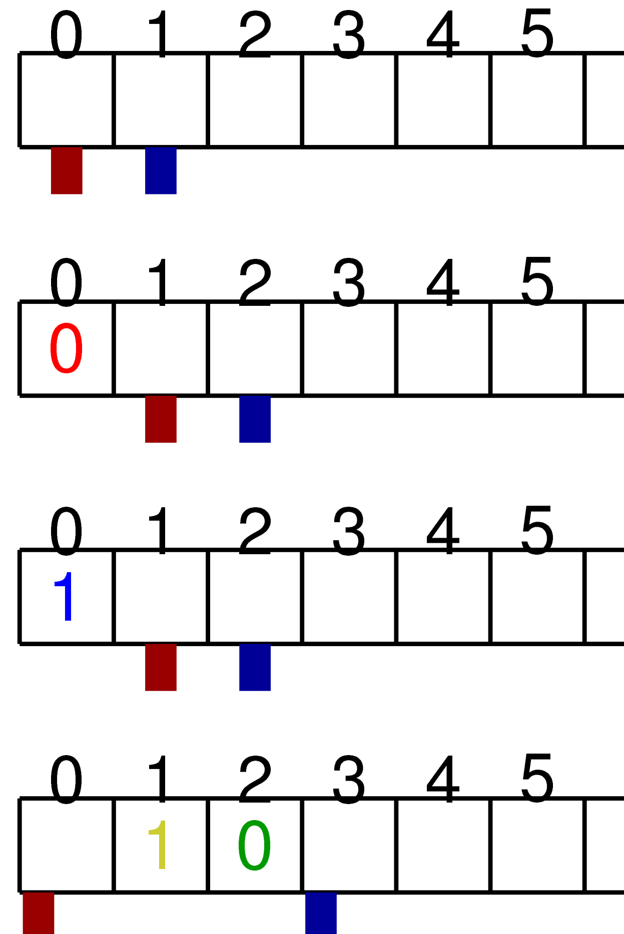




How can we output Gray-code?

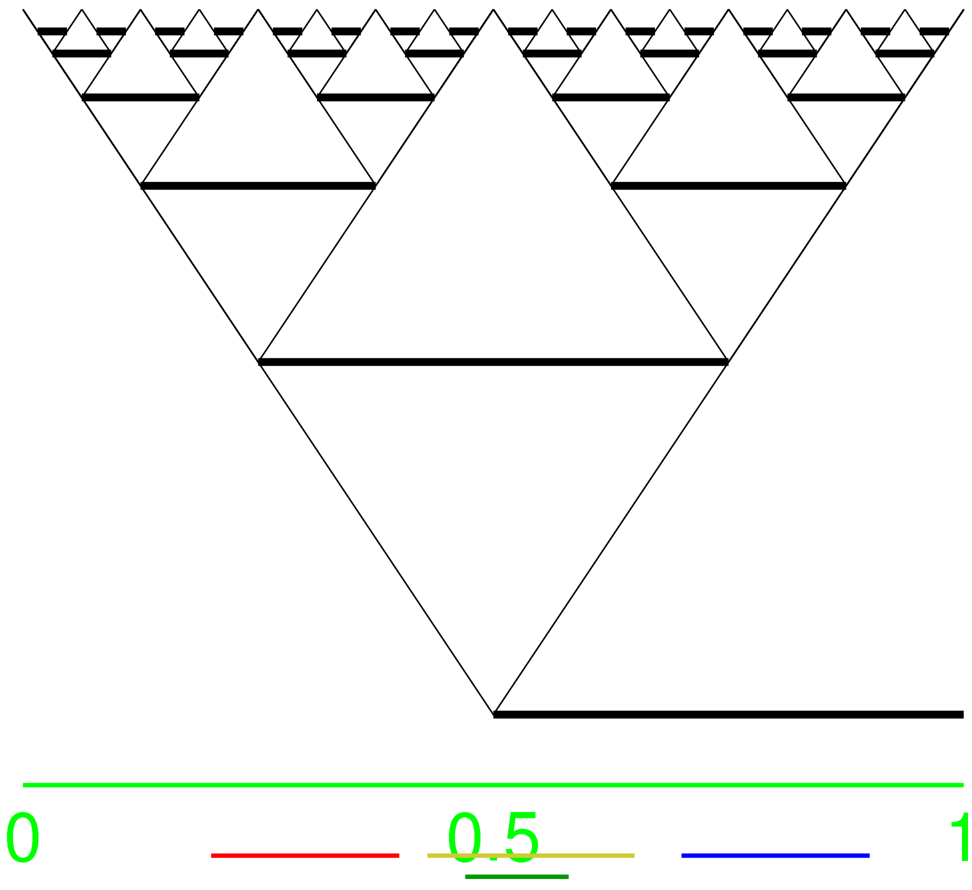
Real number computation as a limit of approximations (shrinking open intervals).

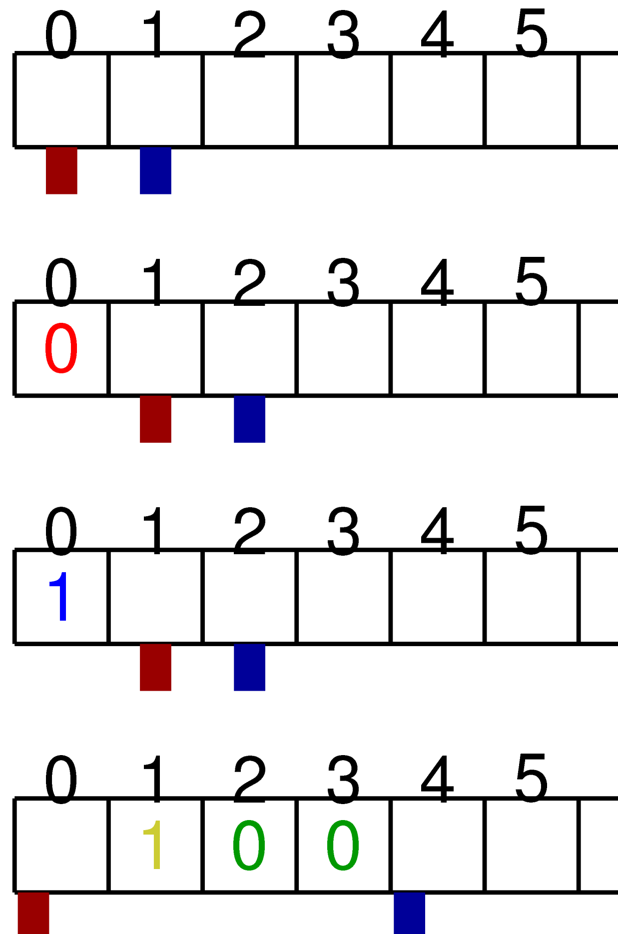




How can we output Gray-code?

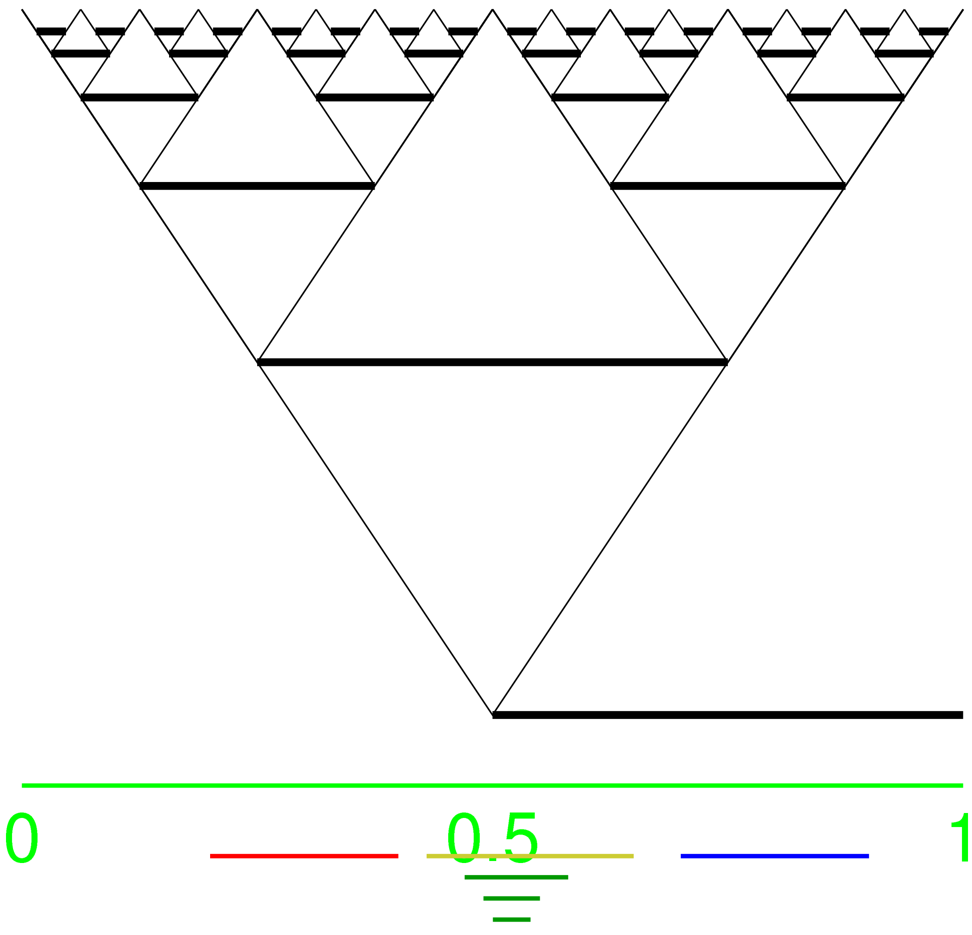
Real number computation as a limit of approximations (shrinking open intervals).

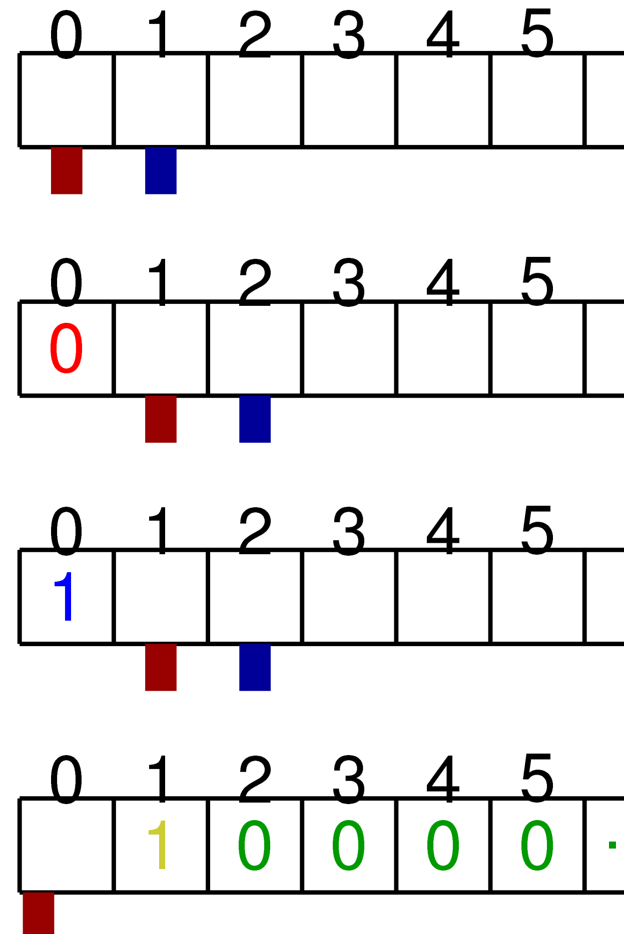




How can we output Gray-code?

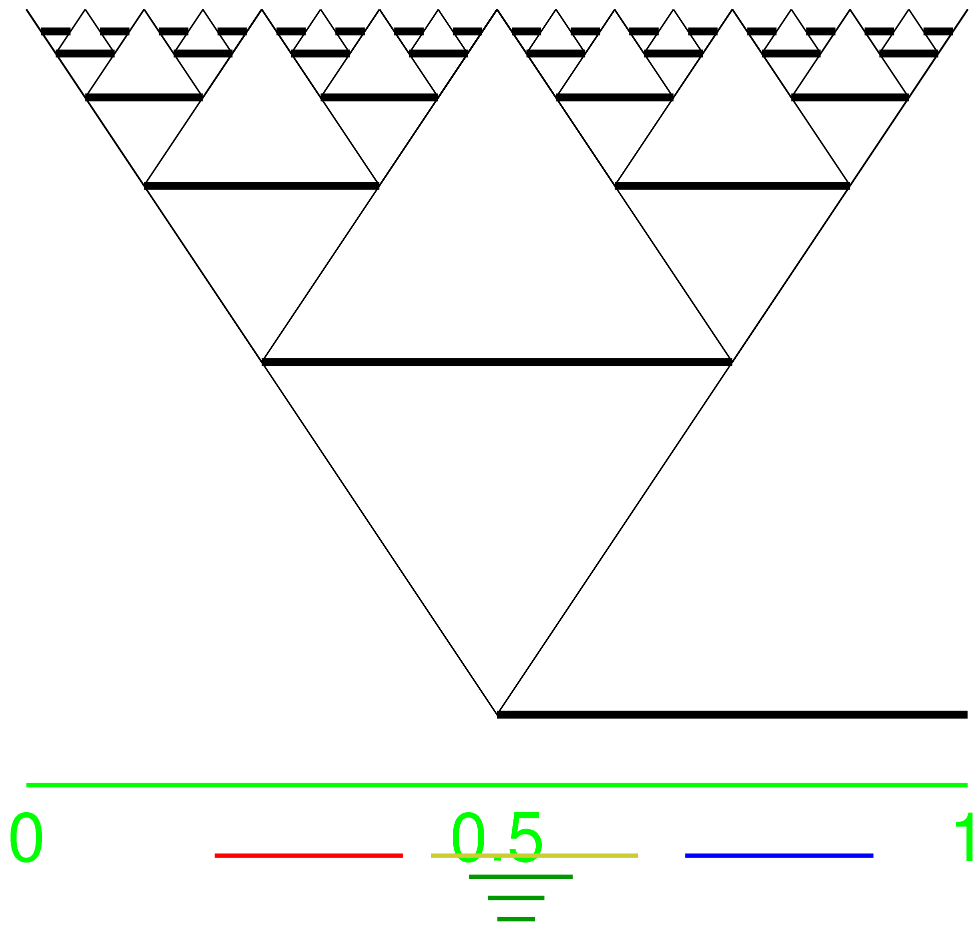
Real number computation as a limit of approximations
(shrinking open intervals).

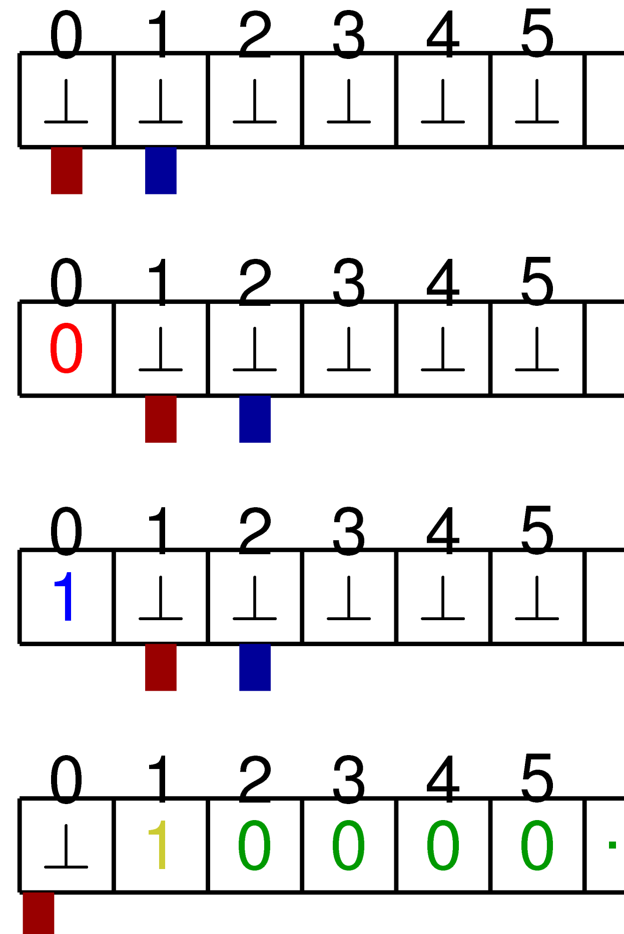




How can we output Gray-code?

Real number computation as a limit of approximations (shrinking open intervals).



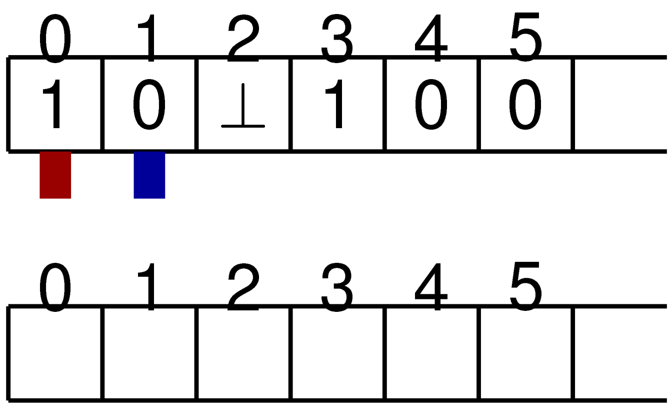


How can we input Gray-code?

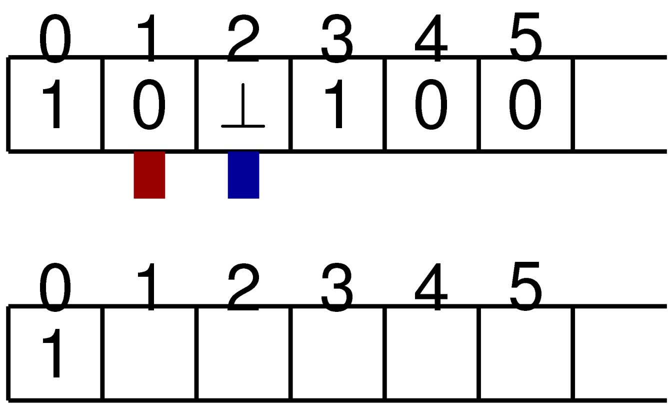
0	1	2	3	4	5	
1	0	$\perp$	1	0	0	

0	1	2	3	4	5	

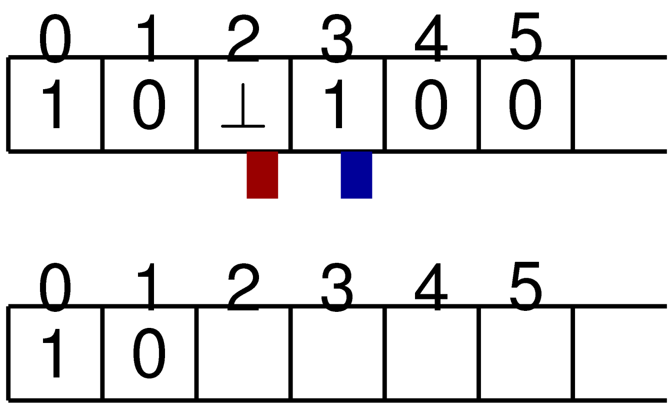
How can we input Gray-code?



How can we input Gray-code?



How can we input Gray-code?

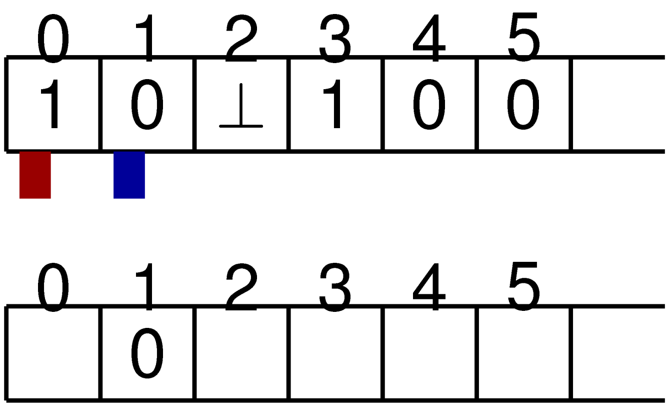


How can we input Gray-code?

0	1	2	3	4	5	
1	0	$\perp$	1	0	0	

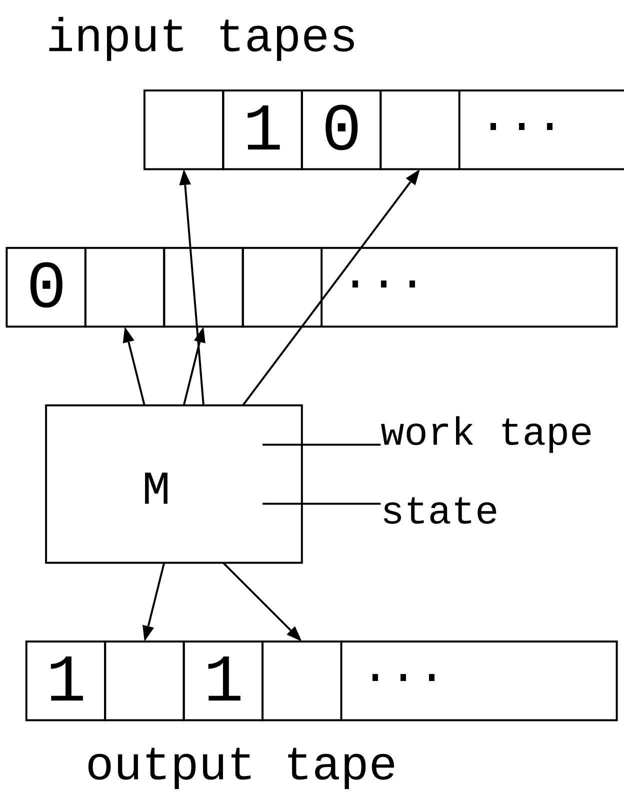
0	1	2	3	4	5	
1	0		1			

How can we input Gray-code?



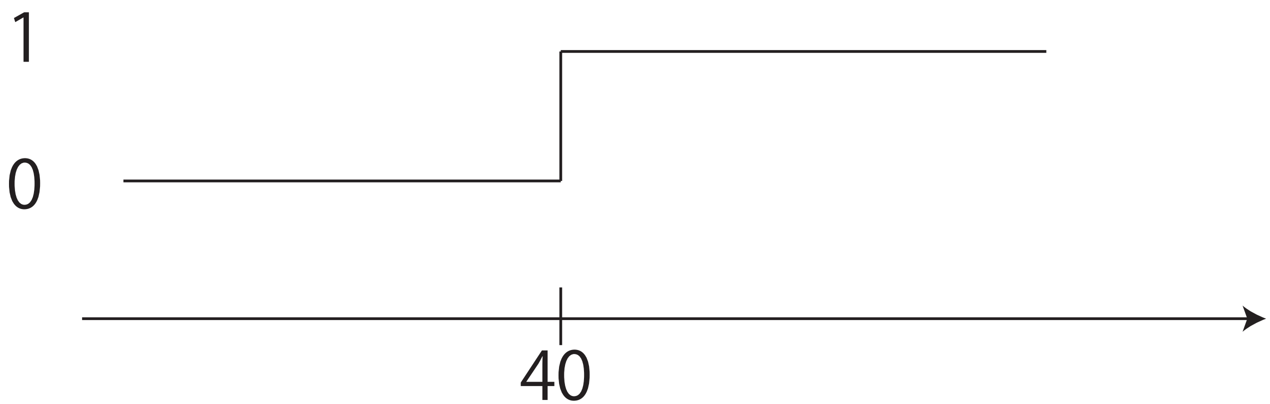
Two possible inputs as the first character.

IM2-machine



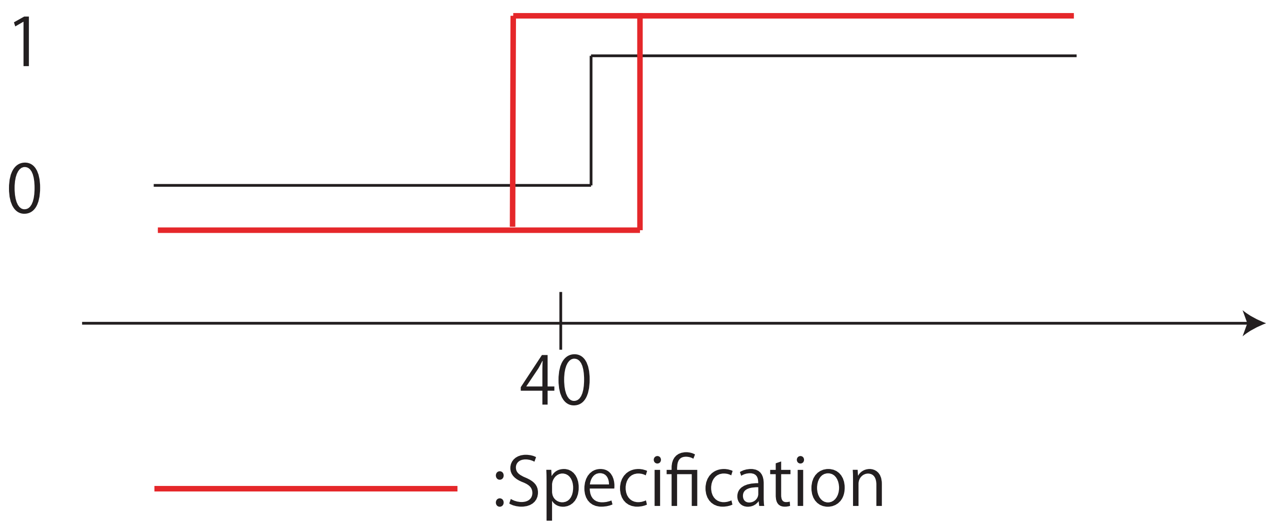
- Generalization of Type-2 machine with 2-heads input/output access.
- Indeterministic (i.e. nondeterminist behavior depending on the head used to input.
  $\rightarrow$ defines a multi-valued function.
  note: Multi-valuedness is natural real number computation)

Multi-valuedness



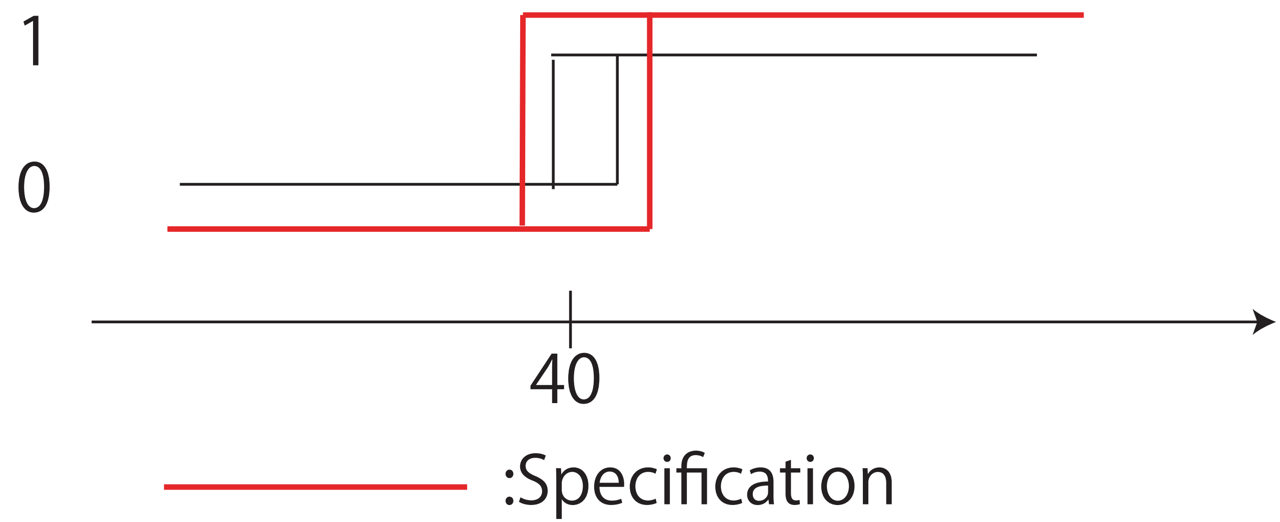
- Consider a thermometer which will make an alarm if it is hotter than 40 degree. Is it possible?

Multi-valuedness



- Consider a thermometer which will make an alarm if it is hotter than 40 degree. Is it possible?
- Physically, it should be around 40 with a specification from $40 - \epsilon$ to $40 + \epsilon$.

Multi-valuedness



- Consider a thermometer which will make an alarm if it is hotter than 40 degree. Is it possible?
- Physically, it should be around 40 with a specification from $40 - \epsilon$ to $40 + \epsilon$.
- Physical implementation should also be multi-valued, depending on how it approaches.

Computability on Real Number

- There are many ways of defining computability on real numbers.
  - TTE (Type two theory of effectivity) by Grzegorczyk, Weihrauch, Hertling, Brattka,...)
  - Pour-El and Richards approach.
  - Many approaches to Exact Real Number Computation. [Boehm, Edalat,Potts, Gianantonimo, Vuillemin,...]
  - Blum-Shub-Smale machine.

Computability on Real Number

- There are many ways of defining computability on real numbers.
  - TTE (Type two theory of effectivity) by Grzegorczyk, Weihrauch, Hertling, Brattka,...)
  - Pour-El and Richards approach.
  - Many approaches to Exact Real Number Computation. [Boehm, Edalat,Potts, Gianantonimo, Vuillemin,...]
  - Blum-Shub-Smale machine.
- Gray-code and IM2-machine computability can be extended to real numbers.
- It is equal to TTE approach with admissible representation.

Implementation in programming languages

How to implement IM2-machines in 'REAL' programming languages?

- It is possible, with logic programming languages with guarded clauses and committed choice, such as Concurrent Prolog, PARLOG, and GHC (Guarded Horn Clauses)
  Direct translation from rules of an IM2-machine to GHC.

Implementation in programming languages

How to implement IM2-machines in 'REAL' programming languages?

- It is possible, with logic programming languages with guarded clauses and committed choice, such as Concurrent Prolog, PARLOG, and GHC (Guarded Horn Clauses)
  Direct translation from rules of an IM2-machine to GHC.
- Impossible, with sequential functional languages like Haskell.
- Extension of Haskell with gamb (sequential partial realization of amb). Implemented 2005.

Examples:

Conversions from/to the signed digit representation.

Signed-digit representation: an expansion of $[-1, 1]$ as an infinite sequence of $\Gamma = \{0, 1, \overline{1}(= -1)\}$, defined as

$$\delta_s(a_1 a_2 \ldots) = \Sigma_{i=1}^{\infty}\{a_i \cdot 2^{-i}\}.$$

Here, we fix $a_1 = 1$ and discard this from the representation.

$$\delta_s([0, 0, ...]) = 1/2.$$

Output (SD $\to$ Gray)

Output: possible in a functional language.

```
stog(1:xs)  = 1:nh(stog xs)
stog(1̄:xs)  = 0:stog xs
stog(0:xs)  = c:1:nh ds where c:ds= stog xs

not 0 = 1
not 1 = 0
nh c:a = not c:a
```

- `stog([0,0..])` has no output (`[⊥,1,0,0..]`)
- `tail(stog([0,0..]))` produces `[1,0,0,0...]` infinitely.

Input (Gray → SD)

Input: impossible...

```
gtos(0:xs)   = 1̄:(gtos (b:xs))
gtos(1:xs)   = 1:(gtos (nh (b:xs)))
gtos(a:1:xs) = 0:(gtos (a:(nh xs)))
```

- Correct Haskell syntax, but different meaning!!
- The third line never used for `⊥:1:[0,0..]`.
- `gtos(stog([0,0..]))` has no output.
  (we expect the output `[0,0..]`).

Real number computation in GHC

```
main              :- pinf(ZZ),gtos(YY,ZZ),stog(XX,YY),inf0(X
stog([-1|X],YY) :- YY=[0|Y],  stog(X,Y).
stog([1|X],YY)  :- YY=[1|Y],nh(Z,Y),stog(X,Z).
stog([0|X],YY)  :- YY=[C,1|Y],nh(Z,Y),stog(X,[C|Z]).
gtos([0|Y],XX)  :- XX = [-1|X],gtos(Y,X).
gtos([1|Y],XX)  :- XX=[1|X],nh(Y,Z),gtos(Z,X).
gtos([C,1|Y],XX):- XX=[0|X],nh(Y,Z),gtos([C|Z],X).
inf0(XX)        :- XX = [0|X], inf0(X).
pinf([X|Y])     :- io:outstream([print(X),flush]),pinf
nh(X,XX)        :- X=[X0|X1],not(X0,Z),XX=[Z|X1].
not(0,X)        :- X = 1.
not(1,X)        :- X = 0.
```

Logic vs. Functional

- Logic programming languages: bottom up
- Functional programming languages: top down

McCarthy's 'amb' operator

As an extension of Haskell, consider the amb operator

```
amb:  a -> a -> Amb a
```

where the datatype `Amb a` is defined as

```
data Amb a = Right a | Left a
```

- `amb` M N: evaluate M and N in parallel. It is reduced to `Left` V when M is reduced to V, `Right` V' when N is reduced to V'.
- Its computation does not terminate only when both M and N do not have normal forms.
- When both M and N have normal forms, we have two possibilities and thus it is a nondeterministic multi-valued operator.

Implementation in Haskell + amb

We can implement IM2-machines in Haskell + amb.

```
gtos(a:b:xs) = case (amb a b) of
   Left 0 -> 1̄:(gtos (b:xs))
   Left 1 -> 1:(gtos (nh (b:xs)))
   Right 1 -> 0:(gtos (a:(nh xs)))
   Right 0 -> case a of 0->1̄:1̄:(gtos xs)
                        1->1:1:(gtos(nh xs))
```

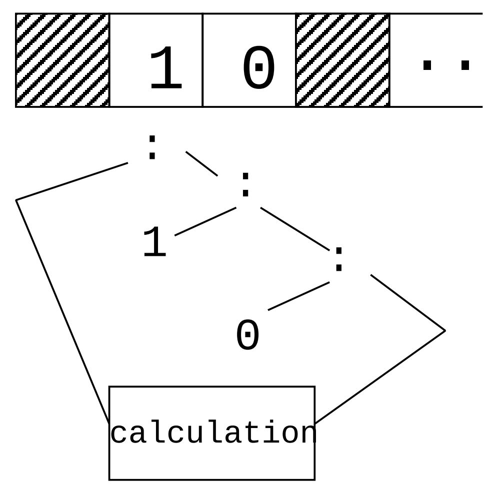
Implementation in Haskell + amb

We can implement IM2-machines in Haskell + amb.

```
gtos(a:b:xs) = case (amb a b) of
   Left 0 -> 1̄:(gtos (b:xs))
   Left 1 -> 1:(gtos (nh (b:xs)))
   Right 1 -> 0:(gtos (a:(nh xs)))
   Right 0 -> case a of 0->1̄:1̄:(gtos xs)
                        1->1:1:(gtos(nh xs))
```

Question:

- Is parallelism really required?
- The output of one sequential computation given at one of the two locations.



gamb:

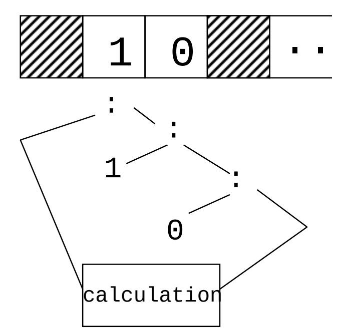
Partial sequential realization of the amb operator.

```
gamb:  Bool -> Bool -> Amb Bool.
```

- Based on graph reduction.
- `gamb` $M\,N$ works only when M and N share the common redex reachable through operator nodes `cons, head, tail, nh, not`.
- For them, apply the following graph-reductions.

Reduction rules (I):

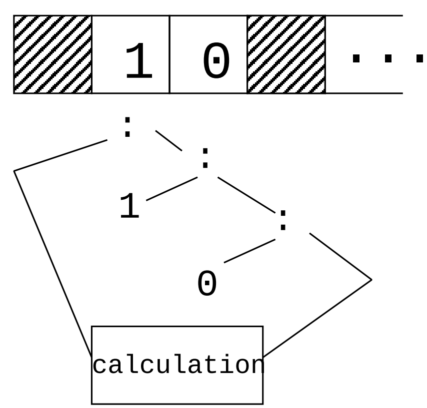
$\texttt{head}\ (B : L)$	$\to B$	(R-head)
$\texttt{tail}\ (B : L)$	$\to L$	(R-tail)
$\texttt{not}\ 0$	$\to 1$	(R-not0)
$\texttt{not}\ 1$	$\to 0$	(R-not1)
$\texttt{nh}\ (B : L)$	$\to (\texttt{not}\ B) : L$	(R-nh)



Reduction of $\mathtt{gamb}\, M\, N$:

1. M and N are reduced with the rules in (I).
2. Return `Right` c or `Left` c if one of them become a normal form (0 or 1),
3. Compare the leftmost outermost redexes of M and N. If they are different node, raise a runtime error.
4. Reduce the shared redex to a weak head normal form.
5. Repeat 1. to 4. until it returns in 2. or it raises an error in 3..

Not for terms, but for term-graphs.



Properties of gamb

- In the form of

```
gtos(a:b:xs) = case (gamb a b) of...
```

  one can express the behavior of an IM2-machine.
- gamb is a single-valued deterministic function at the level of graph reduction, and a multi-valued function at the functional level.
- Nondeterminism not as the result of parallelism, but depending on the intensional representation.
- “Real Number Expression” should be a graph. rather than a term. Containing some information how we come to this state, and depending on it the computation proceeds.

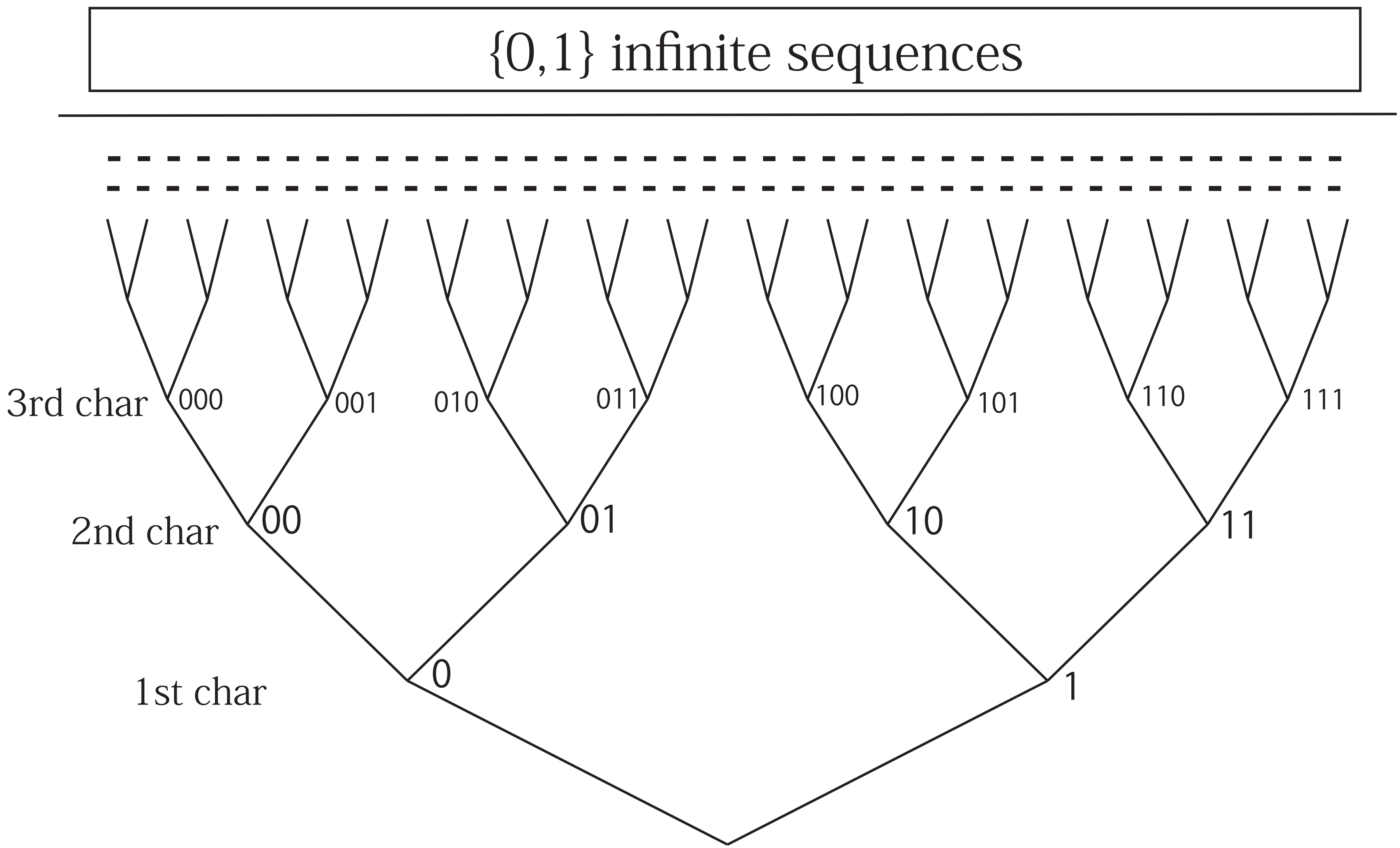
Addition

```
pl (a1:a2:as) (b1:b2:bs) =
 case (gamb a1 a2) of
  Left 0 -> case (gamb b1 b2) of
    Left 0 -> 0:(pl (a2:as) (b2:bs))
    Left 1 -> (head d):1:(nh (tail d))
        where d = pl (a2:as) (b2:bs)
    Right 1 -> case (gamb a2 (head as)) of
      Left 0 -> 0:(pl (1:(nh as)) (b1:1:(tail bs)))
      Left 1 -> (head d):1:(nh (tail d))
        where d = pl (a1:(nh as)) (b1:(nh bs))
      Right 1 -> 0:1:(pl ((not a2):(nh (tail as))) ((not b1):(nh
(tail bs))))
  Left 1 -> case (gamb b1 b2) of
    Left 1 -> 1:(pl (a2:as) (b2:bs))
    Left 0 -> (head d):1:(nh (tail d))
        where d = pl (nh (a2:as)) (b2:bs)
    Right 1 -> case (gamb a2 (head as) of
      Left 0 -> 1:(pl (1:(nh as)) ((not b1):1:(tail bs)))
      Left 1 -> (head d):1:(nh (tail d))
        where d = pl (a1:(nh as)) (b1:(nh bs))
      Right 1 -> 1:1:(pl ((not a2):(nh (tail as))) (b1:(nh (tail
```

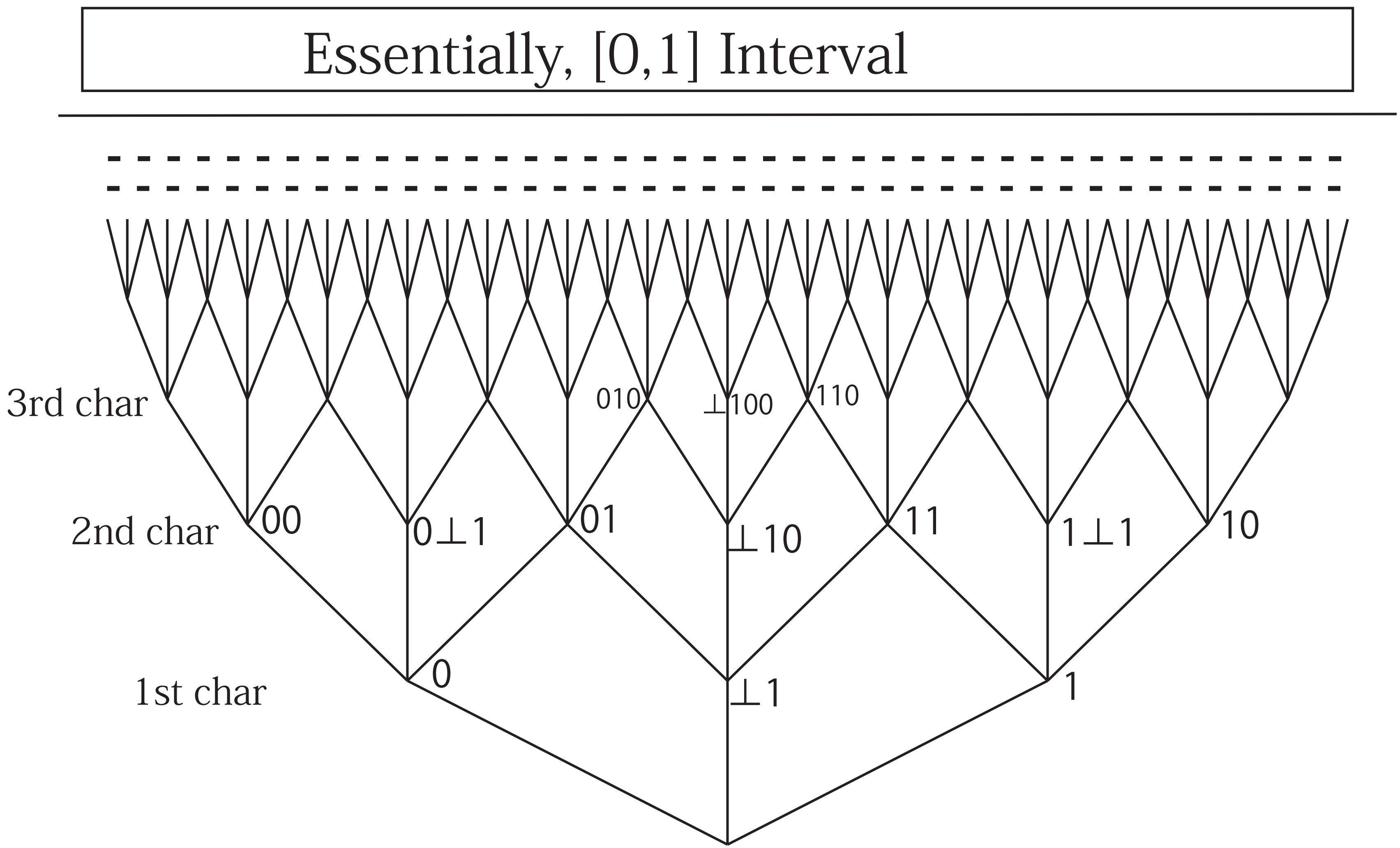
Part three:

Domain Representation of Reals and Topological Spaces.

Cantor Set and its finite approximations



Real Number as limit of Gray-code



$\Sigma^{\omega}_{\perp,n}$-Representation of Topological Spaces

As we have noted,

- The cantor space (0-dimensional) can be embedded in $\Sigma^{\omega}_{\perp,0}$.
- I (or R) (1-dimensional) can be embedded in $\Sigma^{\omega}_{\perp,1}$.
- I^2 (2-dimensional) can be embedded in $\Sigma^{\omega}_{\perp,2}$.

$\Sigma^\omega_{\perp,n}$-Representation of Topological Spaces

As we have noted,

- The cantor space (0-dimensional) can be embedded in $\Sigma^\omega_{\perp,0}$.
- I (or R) (1-dimensional) can be embedded in $\Sigma^\omega_{\perp,1}$.
- I^2 (2-dimensional) can be embedded in $\Sigma^\omega_{\perp,2}$.

Theorem: A separable metric space can be embedded in $\Sigma^\omega_{\perp,n}$ iff it is n-dimensional.

$\Sigma^{\omega}_{\perp,n}$-Representation of Topological Spaces

As we have noted,

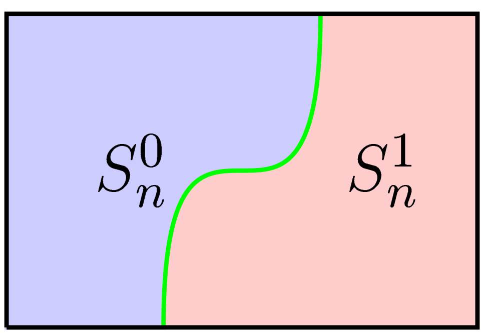
- The cantor space (0-dimensional) can be embedded in $\Sigma^{\omega}_{\perp,0}$.
- I (or R) (1-dimensional) can be embedded in $\Sigma^{\omega}_{\perp,1}$.
- I^2 (2-dimensional) can be embedded in $\Sigma^{\omega}_{\perp,2}$.

Theorem: A separable metric space can be embedded in $\Sigma^{\omega}_{\perp,n}$ iff it is n-dimensional.

- n-dimensional space can be accessed with $n+1$ heads.

Dyadic Subbase

- What is $\{0, 1, \perp\}^\omega$-representation, topologically?
- For each bit k, $\{x : \varphi(x)(k) = 0\}$ and $\{x : \varphi(x)(k) = 1\}$ are regular open sets and $\{x : \varphi(x)(k) = \perp\}$ is their common boundary.
- Representing topological space as an infinite product of this. [Ohta, T, Yamada]
- $\perp$ corresponds to the boundary, and is the keyword to understand the continuity of this world!



0	1	2	3	4	5	
0	$\perp$	1	1	$\perp$	0	...

Full-Flipping Maps

- Gray-embedding is based on dynamical system of the tent function.
- Generalization of this framework to other dynamical system on other topological spaces.
- Dynamical System is governed by “Symbolic Dynamical System”, which is the combinatorial study of $\{0,1\}^\omega$ sequences.
- Is it related to formal language theory and learning theory?

Conclusion:

Bottom and Continuity.

Bottom and Continuity

- The discovery of $\perp$ is the greatest contribution of computer science (and logic) to the world!.
  (Before that, we have only deterministic worldview with 0 and 1.)

Bottom and Continuity

- The discovery of $\perp$ is the greatest contribution of computer science (and logic) to the world!.
  (Before that, we have only deterministic worldview with 0 and 1.)
- I love Real number and continuous things more than $\{0, 1\}$-sequences.
- Continuous things can be coded with sequences with bottoms. Gray-code is a continuously changing code.
- 0 and 1 are connected through $\perp$.

Bottom and Continuity

- The discovery of $\perp$ is the greatest contribution of computer science (and logic) to the world!. (Before that, we have only deterministic worldview with 0 and 1.)
- I love Real number and continuous things more than $\{0,1\}$-sequences.
- Continuous things can be coded with sequences with bottoms. Gray-code is a continuously changing code.
- 0 and 1 are connected through $\perp$.
- Even if there are bottoms in data, we can proceed with the rest of the information, with IM2-machine. Therefore, continuous way of handling data is possible.
- Bottom is allowed to appear only once in the coding of reals.