Representing Nondeterministic, Recursive, and Impure Programs in Coq

Nicolas Chappe, Paul He, Ludovic Henrio,
Steve Zdancewic and Yannick Zakowski

~~Choice~~ Trees

Interaction

Representing ~~Nondeterministic~~, Recursive, and Impure Programs in Coq (POPL'20)

Li-yao Xia, Yannick Zakowski, Paul He, Gregory Malecha,
Chung-Kil Hur, Benjamin Pierce, Steve Zdancewic

Introduction: Monadic Definitional Interpreters

Prior Work: Interaction Trees

Choice Trees: Tackling Non-Determinism

CTrees, LTSs and Bisimulations

Conclusion

Introduction: Monadic Definitional Interpreters

Modelling computations in a proof assistant

Why?

Many interesting properties:

- Does a program respect its specification?
- Are two syntactically different programs equivalent?
- Does a compiler respect the meaning of its input programs?

→ Notions of equivalence and refinement

How?

$$\frac{c_1 \mid \sigma \to c_1' \mid \sigma'}{c_1; c_2 \mid \sigma \to c_1'; c_2 \mid \sigma'}$$ Small-step

$$\frac{c_1 \mid \sigma \downarrow \sigma' \quad c_2 \mid \sigma' \downarrow \sigma''}{c_1; c_2 \mid \sigma \downarrow \sigma''}$$ Big-step

$$[c_2] \circ [c_1]$$ Denotational

composition of continuous functions over a CPO

Modelling computations in a proof assistant

Why?

Many interesting properties:

- Does a program respect its specification?
- Are two syntactically different programs equivalent?
- Does a compiler respect the meaning of its input programs?

→ Notions of equivalence and refinement

How?

$$\frac{c_1 \mid \sigma \to c_1' \mid \sigma'}{c_1; c_2 \mid \sigma \to c_1'; c_2 \mid \sigma'}$$ Small-step

$$\frac{c_1 \mid \sigma \downarrow \sigma' \qquad c_2 \mid \sigma' \downarrow \sigma''}{c_1; c_2 \mid \sigma \downarrow \sigma''}$$ Big-step

$$[c_2] \circ [c_1]$$ Denotational

composition of continuous functions over a CPO

The way we model impacts the ways we can reason

The Semantics Impacts the Reasoning

Compositionality: We can reason on parts of the program separately

→ Simplifies the proof technique

Modularity: The semantics is made of several independent parts

→ Improves maintainability

Executability: A complete reference interpreter can be derived from the semantics of a language

→ Helps with testing

Modelling, but how?

Let’s focus on executability

To model something as complex as C or LLVM IR, a reference interpreter is very valuable!

Modelling, but how?

Let’s focus on executability

To model something as complex as C or LLVM IR, a reference interpreter is very valuable!

ITrees take a simple route (back to the 70’s with Reynolds)

Definitional Interpreters

Describe the language to model
via an interpreter written in your host language

Modelling, but how?

Let’s focus on executability

To model something as complex as C or LLVM IR, a reference interpreter is very valuable!

ITrees take a simple route (back to the 70’s with Reynolds)

Definitional Monadic Interpreters

Describe the language to model
via an interpreter written in your host language

Interpreter for a Modest Language

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

Commands map an initial environment (memory) to a final environment

```
interp (c : com) (s : env) : env
```

We thread the state manually

```
interp (c1;c2) s1 ≜ let s2 := interp c1 s1
                     in interp c2 s2
```

Monadic Interpreter for a Modest Language

$$Imp \triangleq \bullet \mid x := e \mid c_1 ; c_2$$

Commands are stateful computations

```
interp (c : com) : state unit
```

```
state X ≜ env -> (env * X)
```

```
maybestate X ≜ env -> option (env * X)
```

Expressions can fail: does not leak into the definition of the sequence

The monad tells us how to thread computations

```
interp (c1;c2) ≜ interp c1 ;; interp c2
```

Interaction Trees

or

Representing
Recursive, and Impure
Programs in Coq

ITree Idea 1: the Free Monad

Stateful computations ~~map initial environments to final environments~~
are computations performing reads and writes

`free E X = X + E X + E E X + ...`

My computation is a piece of syntax

able to perform operations specified in `E`

in order to compute a value of type `X`

ITree Idea 1: the Free Monad

Stateful computations ~~map initial environments to final environments~~

are computations performing reads and writes

```
interp (c : com) : free Rd_Wr unit
```

```
free E X  = X + E X + E E X + ...
```

My computation is a piece of syntax

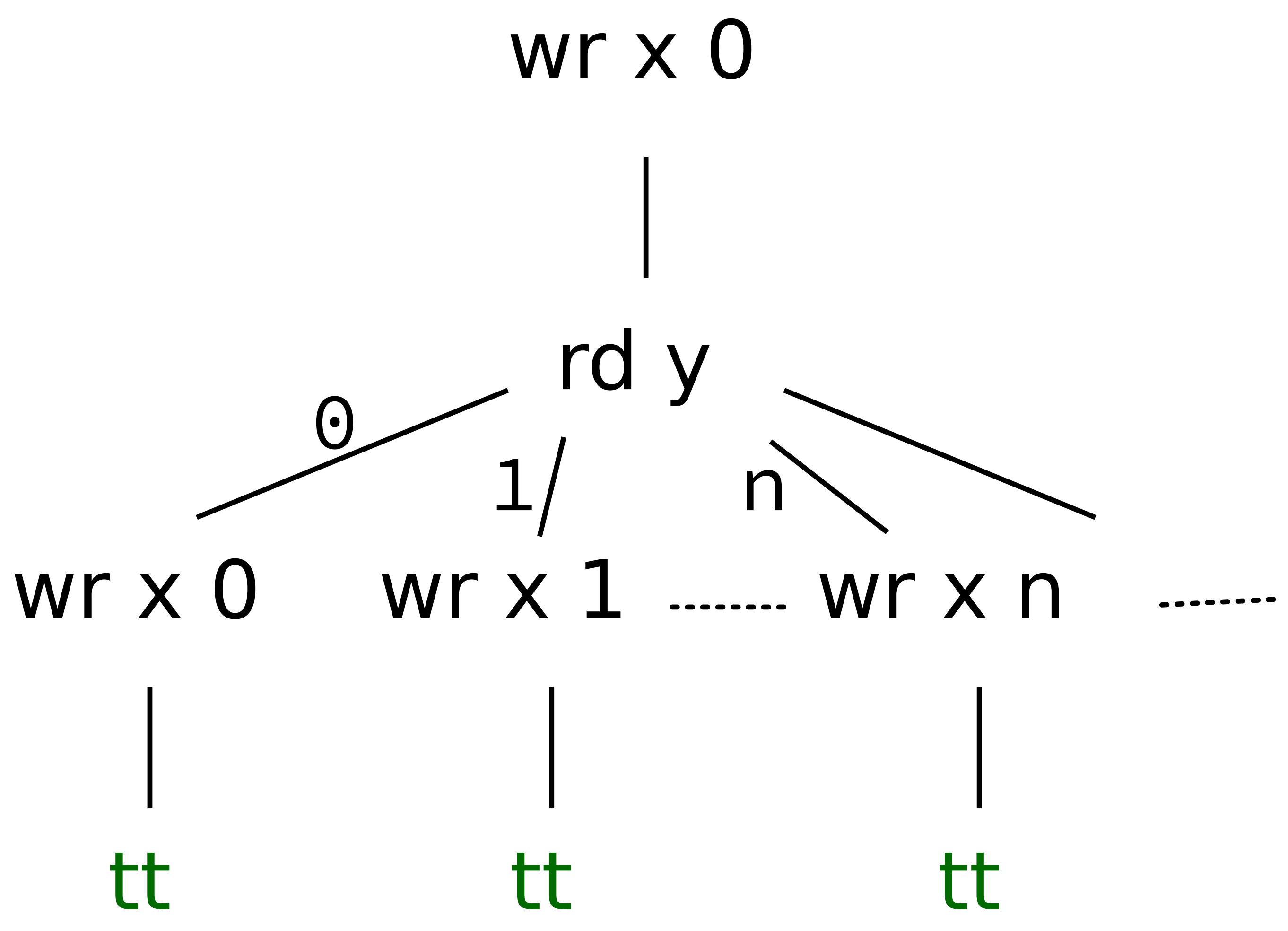
able to perform operations specified in `E`

in order to compute a value of type `X`

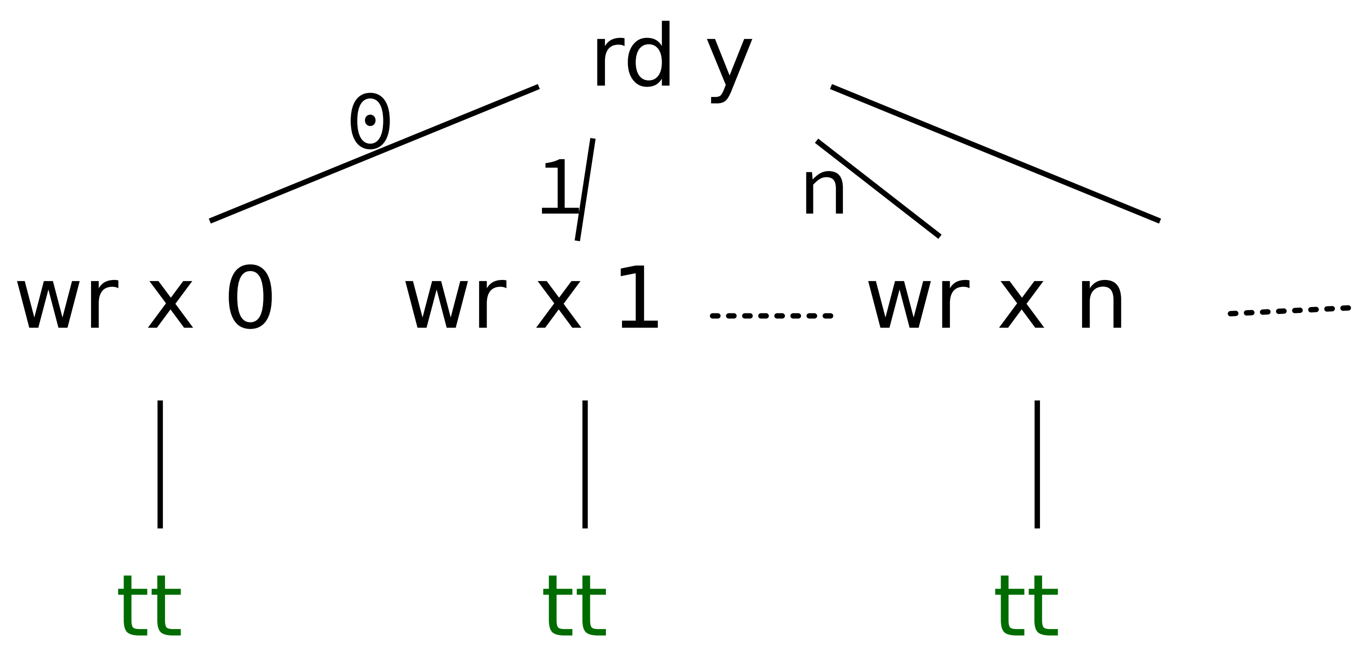
Programs as Trees

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

$$p_2 \triangleq x := 0; x := y$$



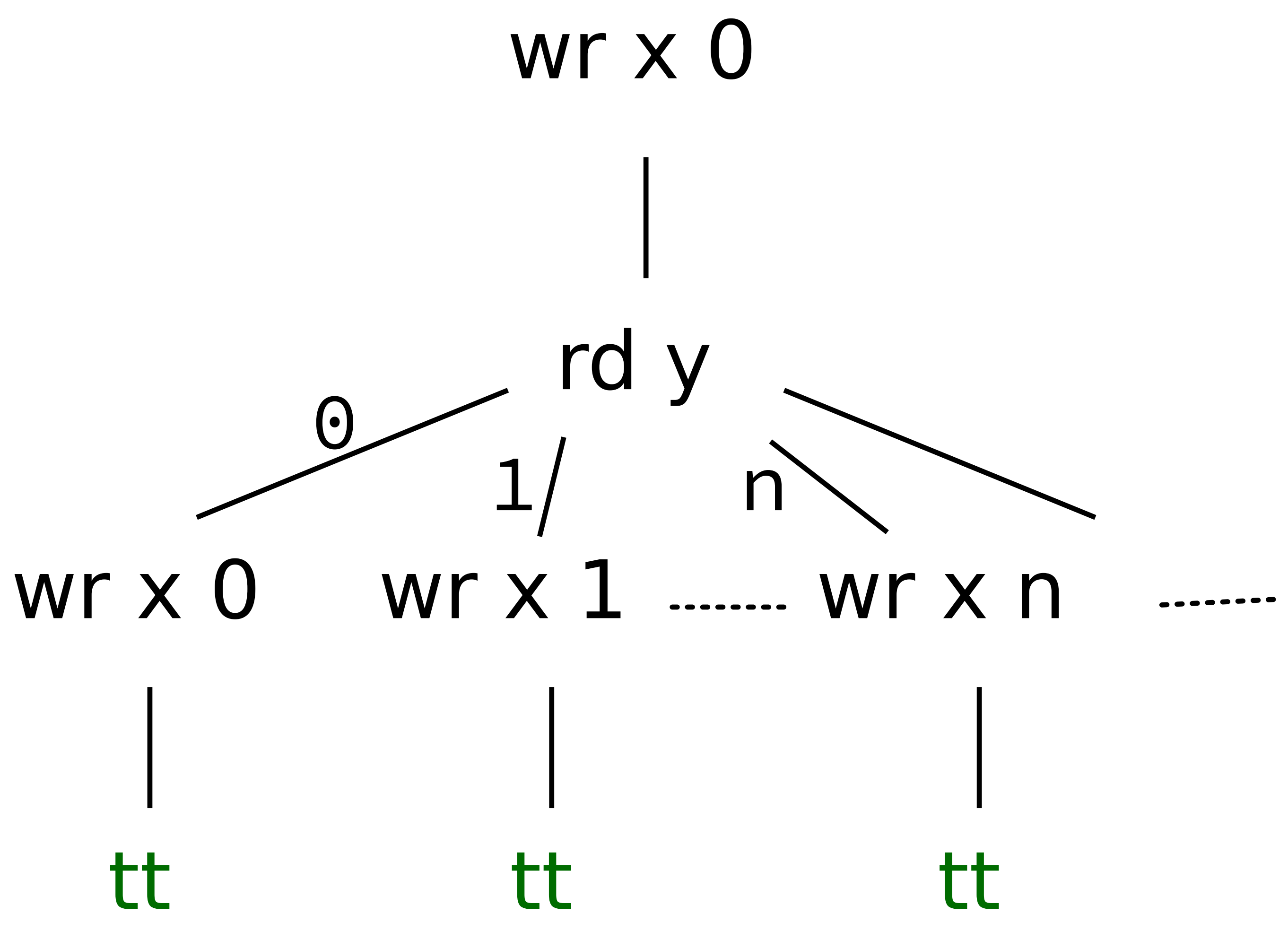
$$p_3 \triangleq x := y$$

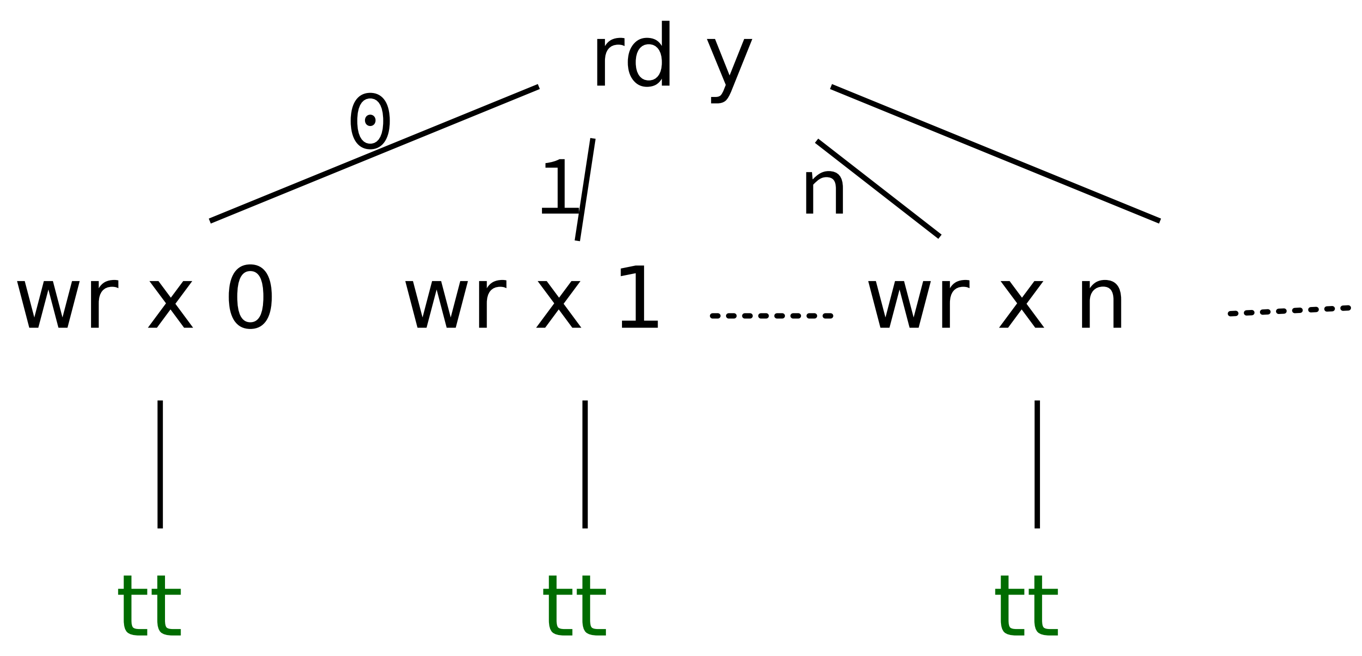


Programs as Trees

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

$p_2 \triangleq x := 0; x := y$ are the same $p_3 \triangleq x := y$

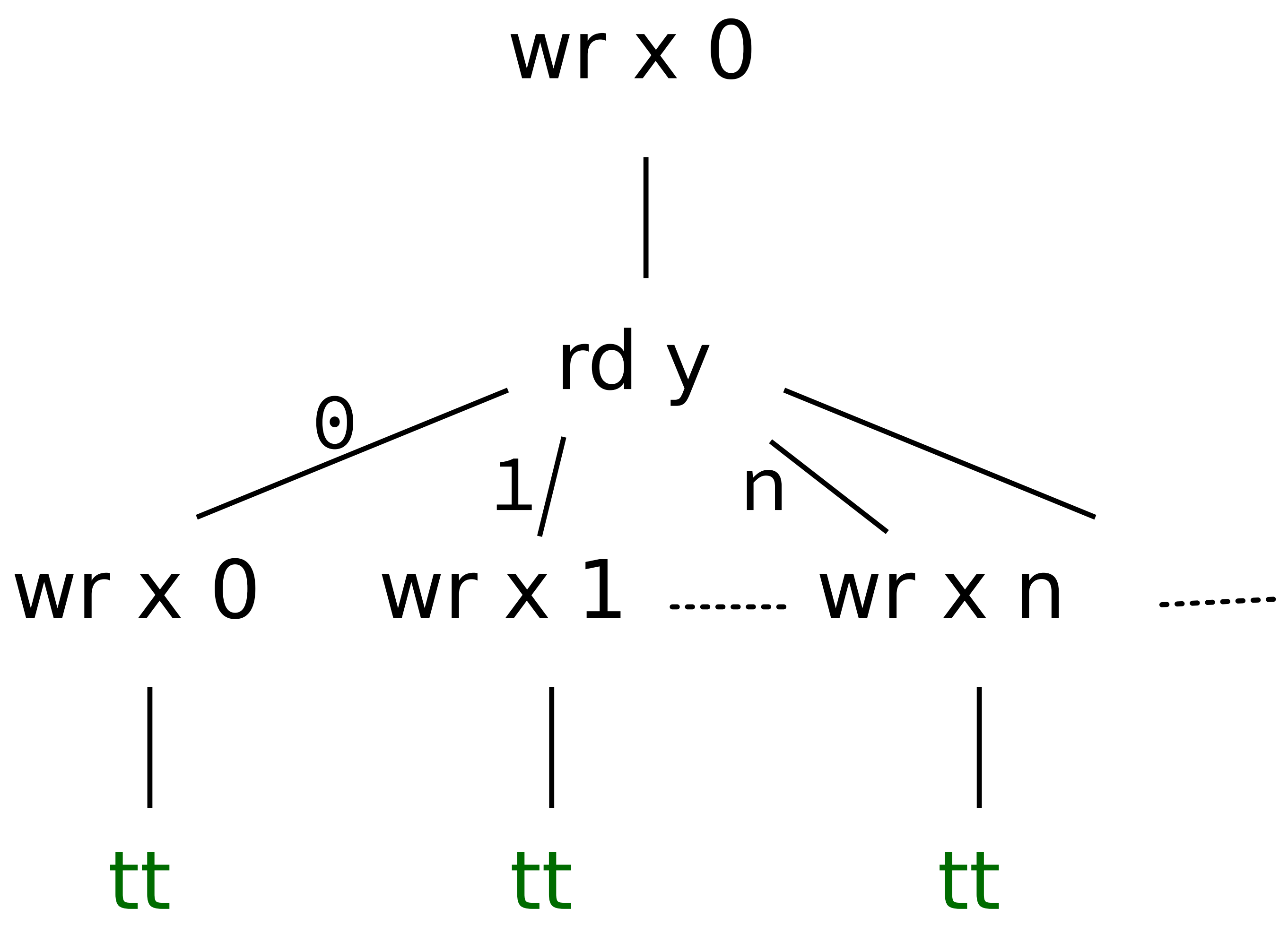




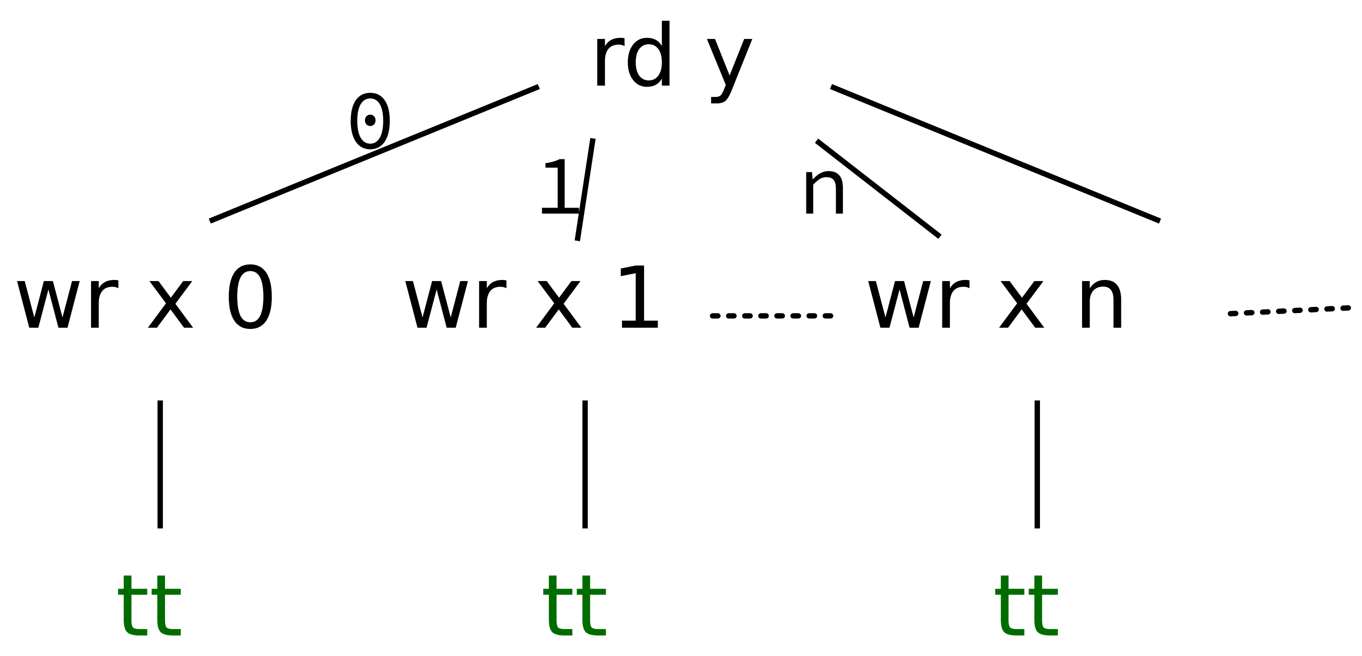
Programs as Trees

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

$p_2 \triangleq x := 0; x := y$ are the same $p_3 \triangleq x := y$



$\not\approx$

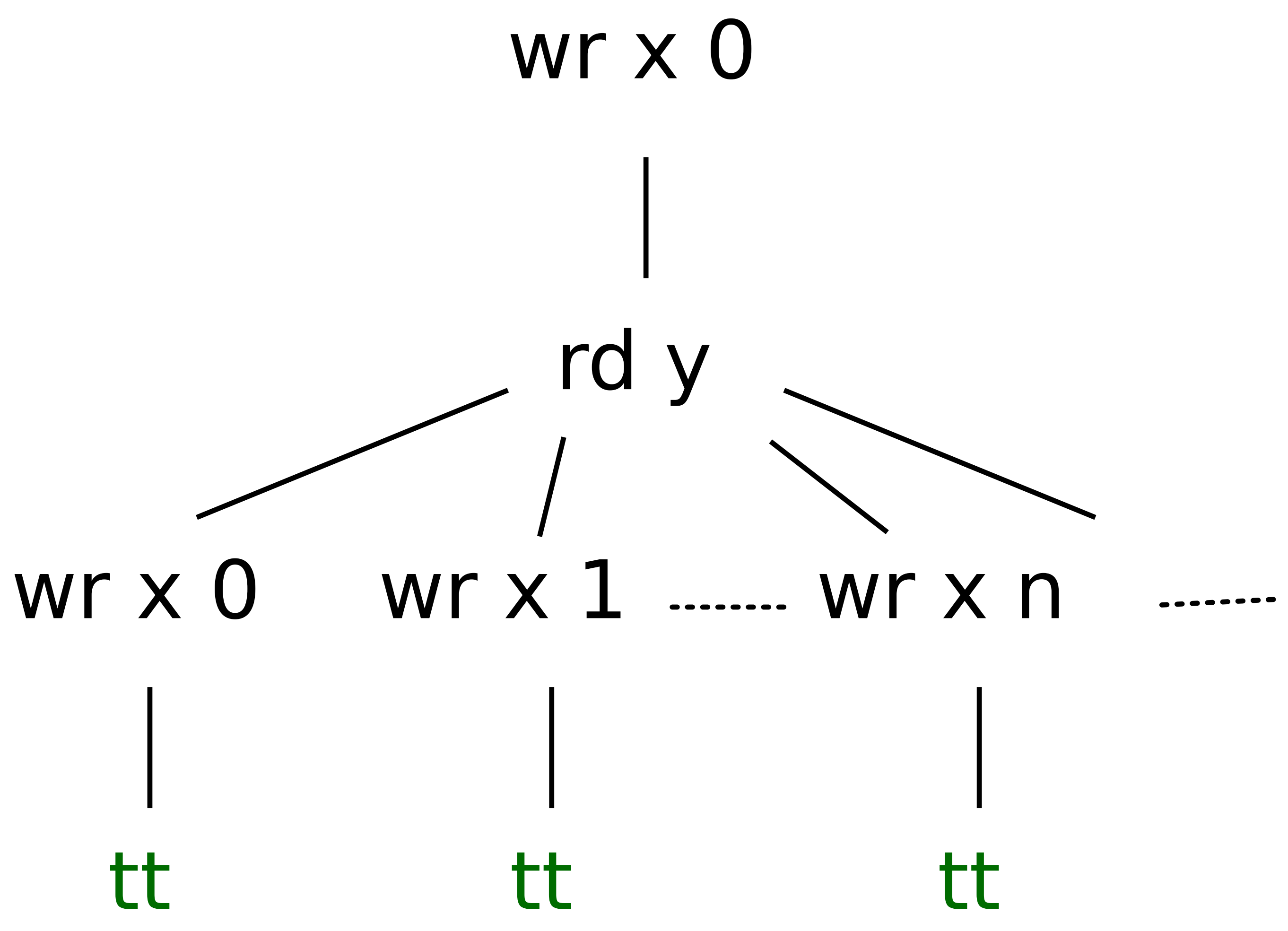


Programs as Trees

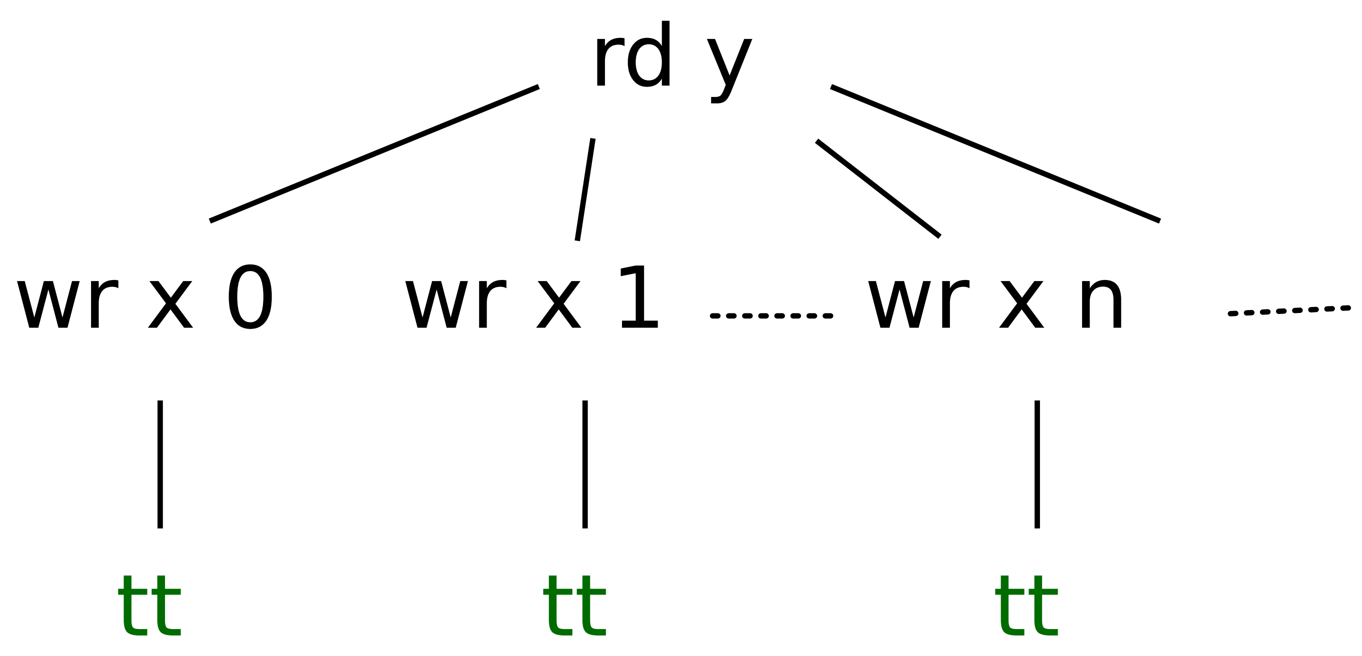
$$Imp \triangleq \bullet \mid x := e \mid c_1 ; c_2$$

Indeed, they are not the same syntax
We fold over the tree to bring in the semantics

$p_2 \triangleq x := 0; x := y$ are the same $p_3 \triangleq x := y$



$\not\approx$

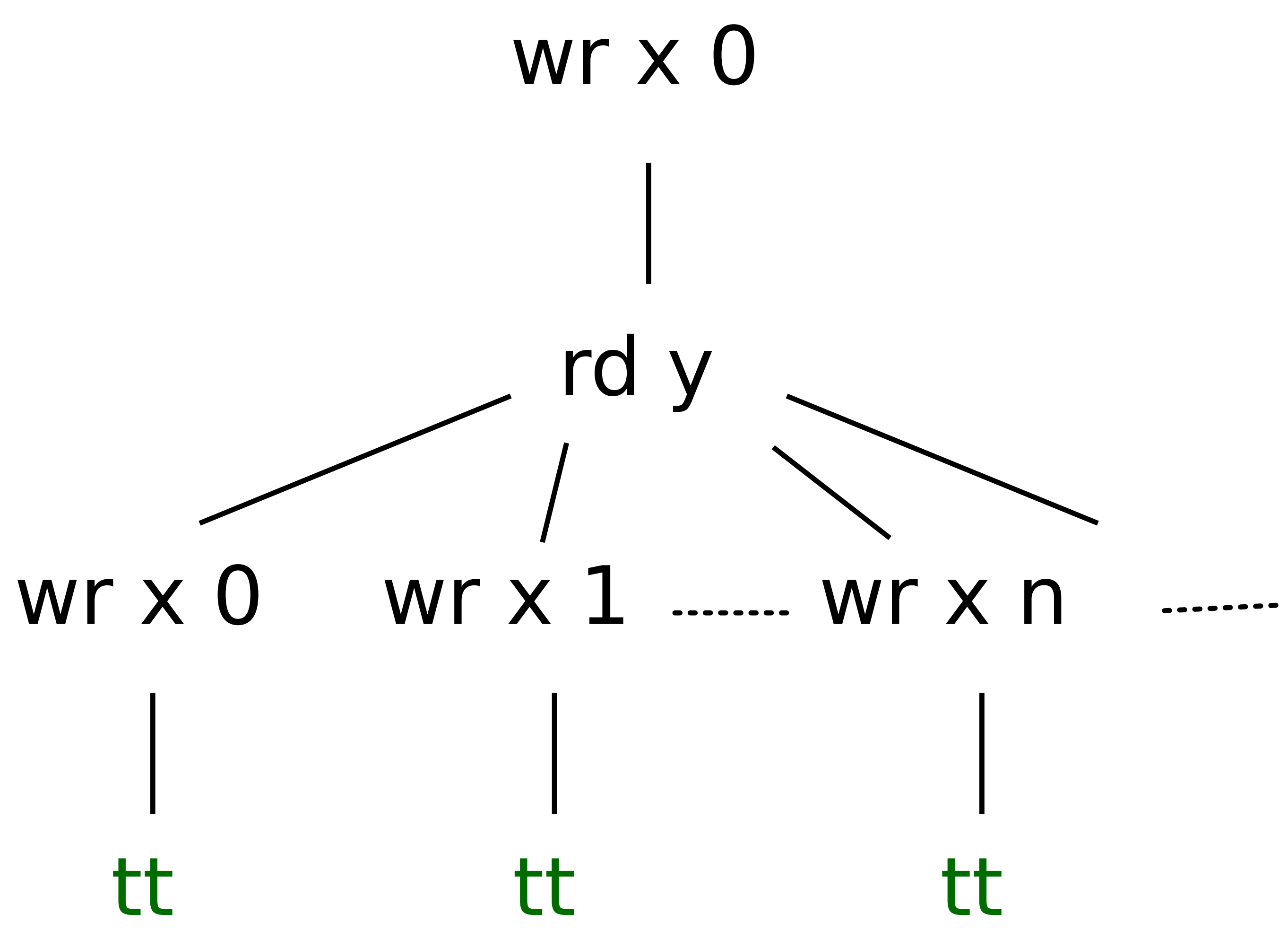


Programs as Trees

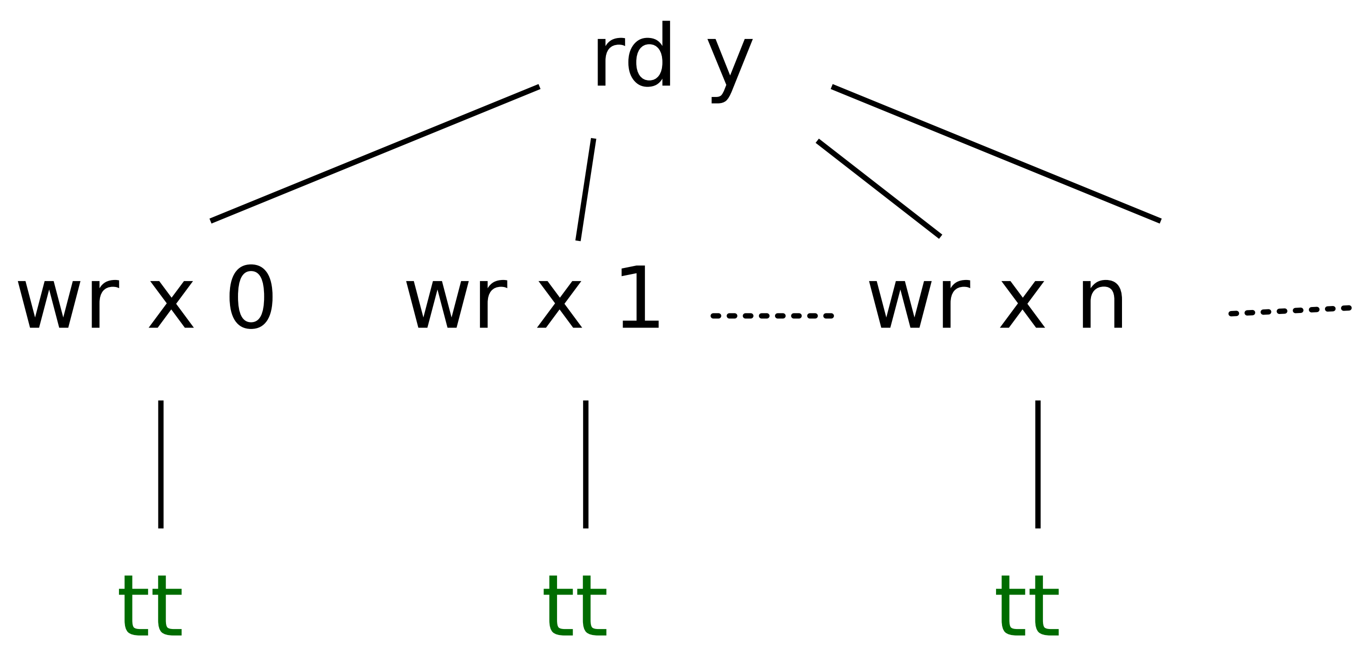
$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

Indeed, they are not the same syntax
We fold over the tree to bring in the semantics

$p_2 \triangleq x := 0; x := y$



$p_3 \triangleq x := y$

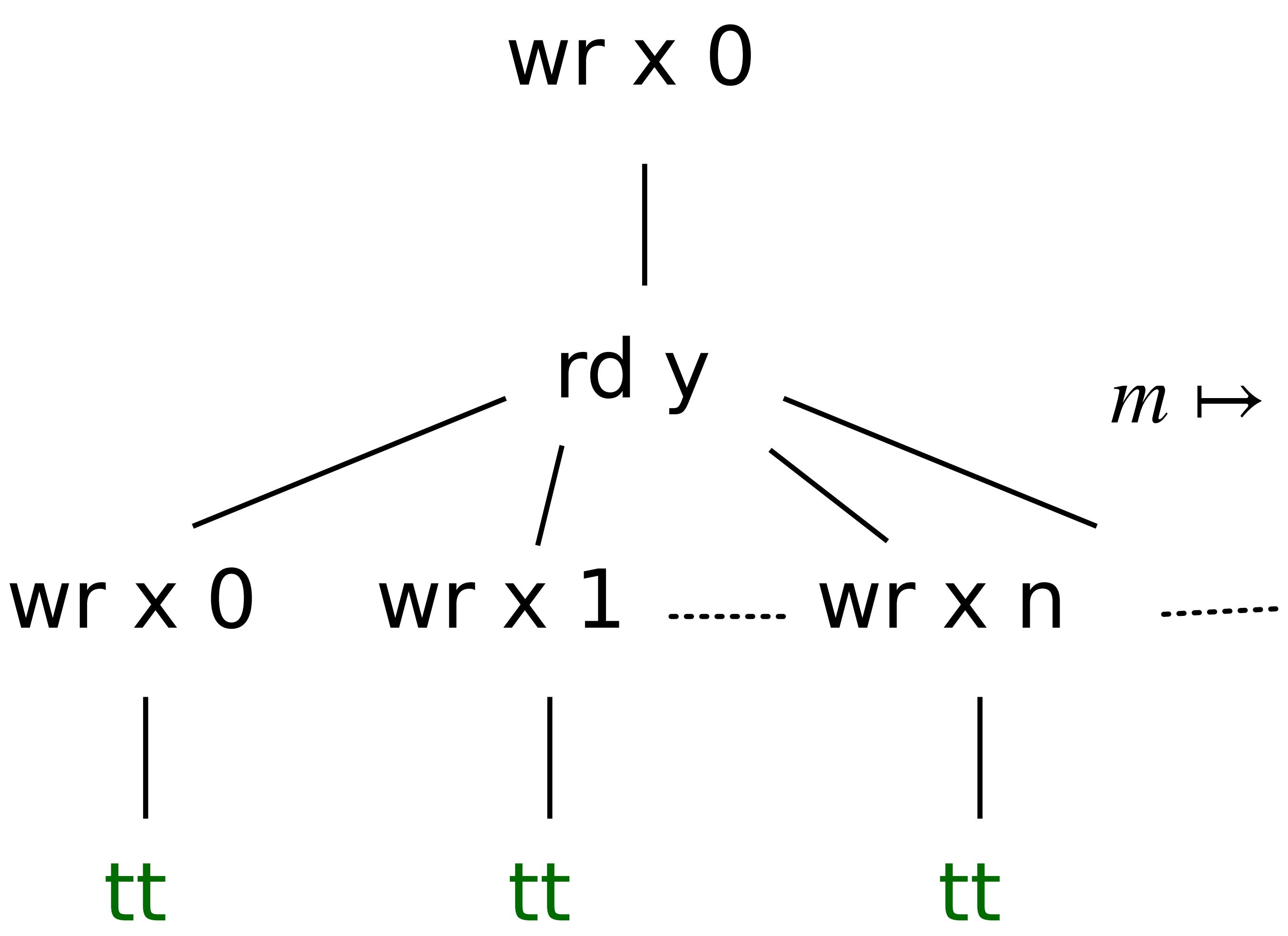


Programs as Trees

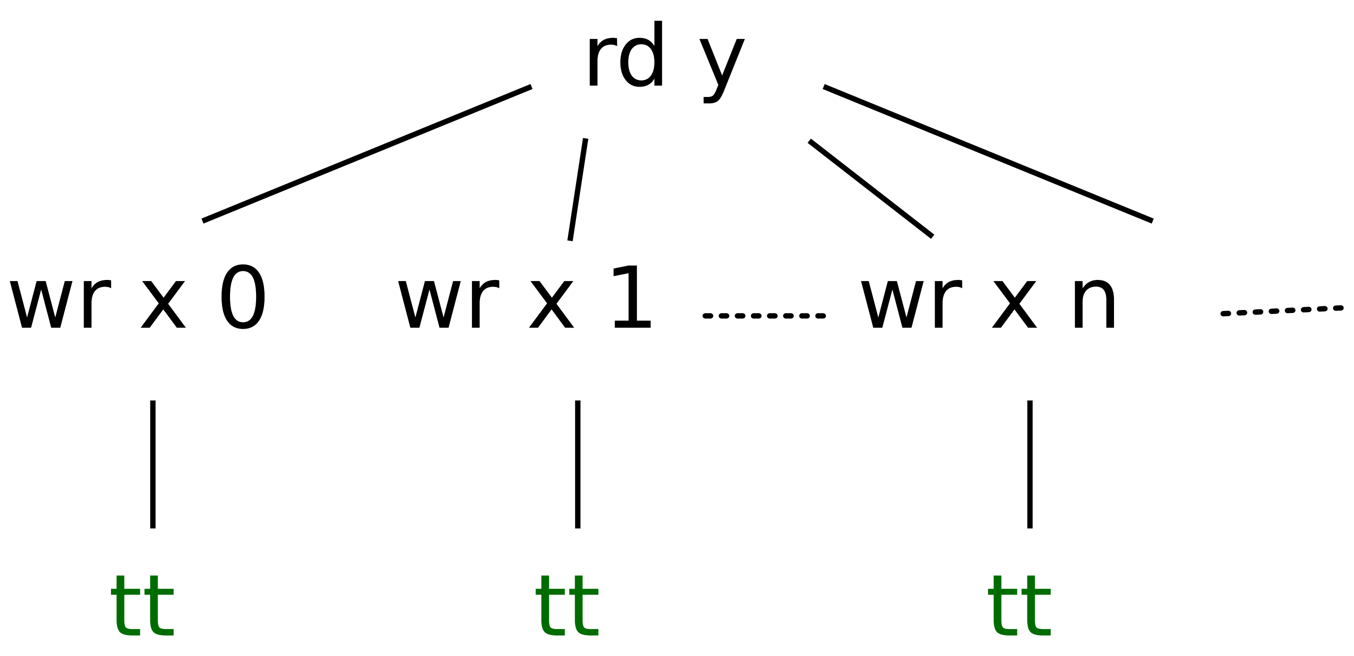
$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

Indeed, they are not the same syntax
We fold over the tree to bring in the semantics

$p_2 \triangleq x := 0; x := y$ are the same $p_3 \triangleq x := y$



$$m \mapsto m\{x \leftarrow 0\}\{x \leftarrow m(y)\} \approx m \mapsto m\{x \leftarrow m(y)\}$$



But What About Loops?

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c$$

$$p_1 \triangleq while\ true\ do\ \bullet$$

What tree should we associate to p_1?

ITree Idea 2: Capretta's Delay Monad

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c$$

$$p_1 \triangleq while\ true\ do\ \bullet$$

`later`

`later`

`later`

Something happened internally
Here, the re-entrance of the loop

ITree Idea 2: Capretta's Delay Monad

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c$$

$$p_1 \triangleq while\ true\ do\ \bullet$$

`later`

`later`

`later`

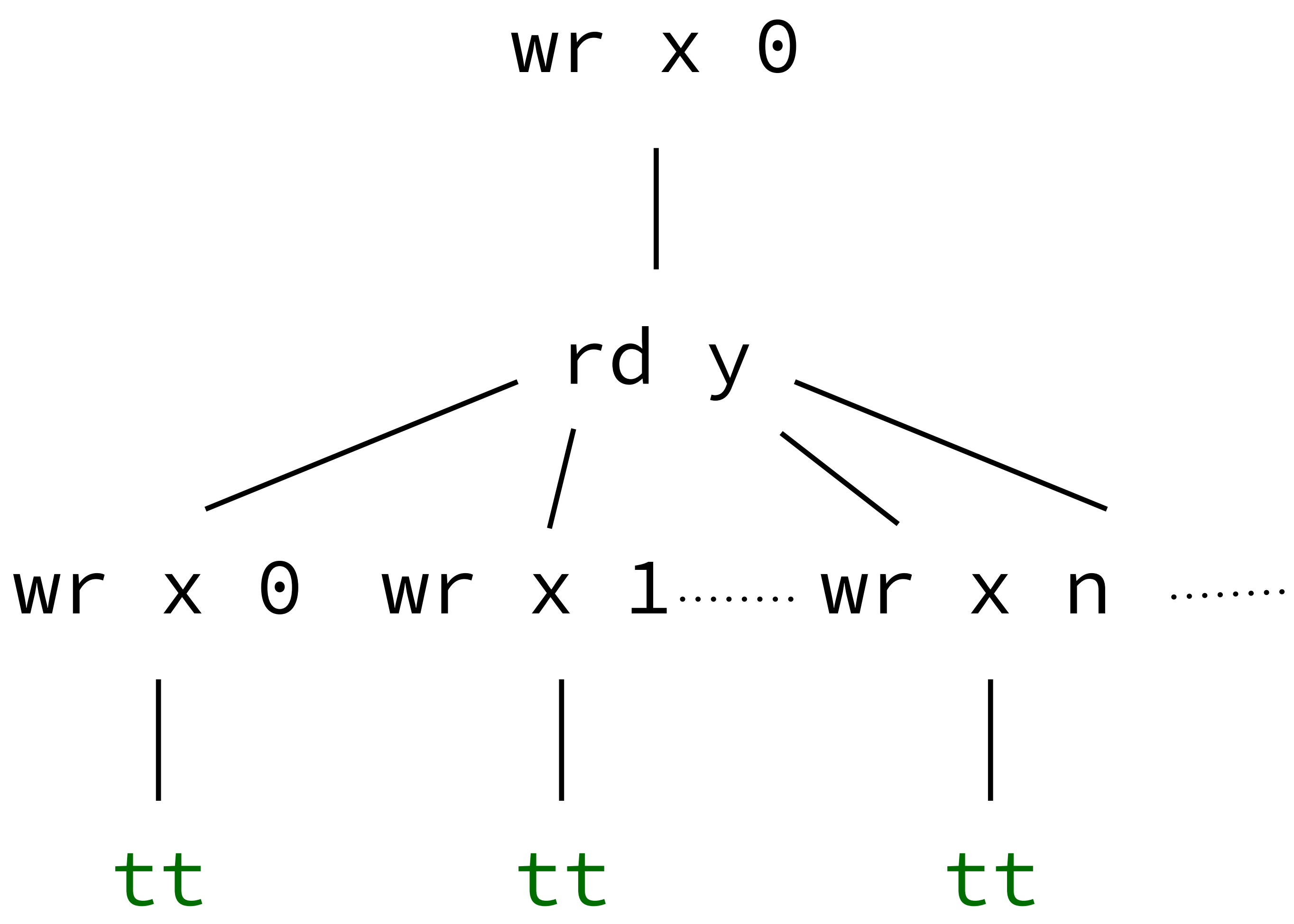
Something happened internally
Here, the re-entrance of the loop

We move onto a coinductive datatype, p_1 is an infinite tree

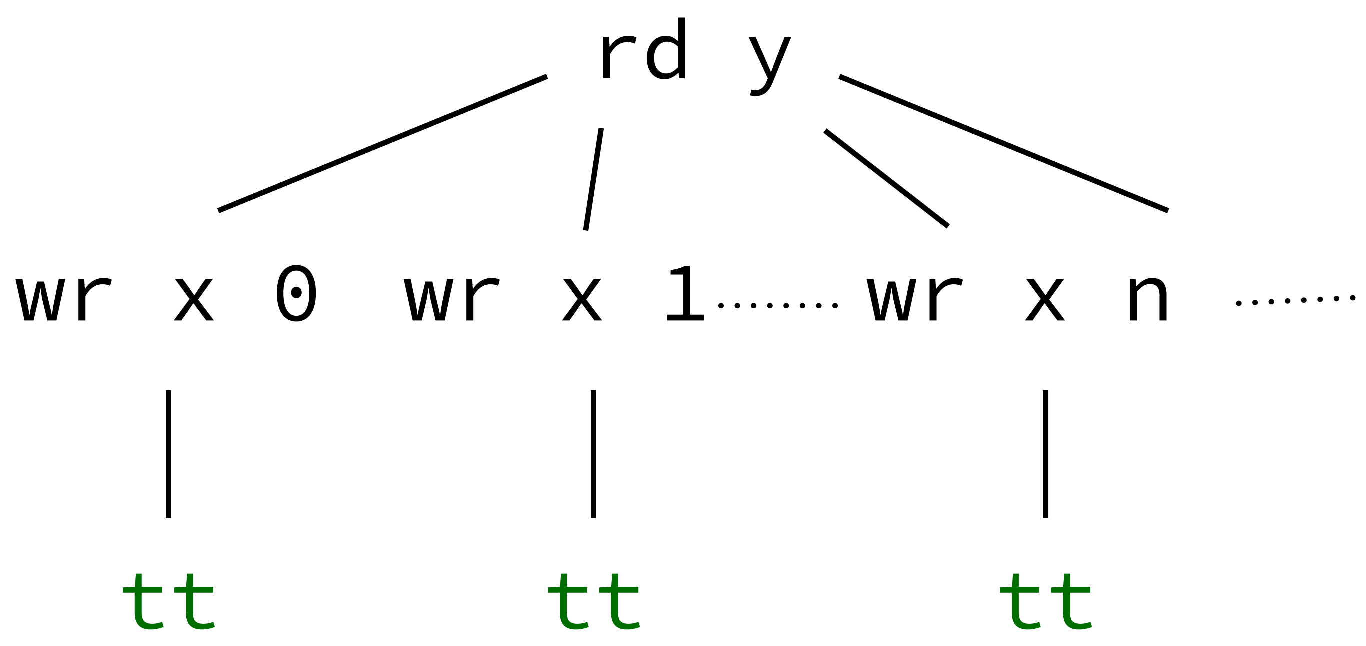
Programs as Stateful Infinite Trees

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c$$

$$p_2 \triangleq x := 0; x := y$$



$$p_3 \triangleq x := y$$

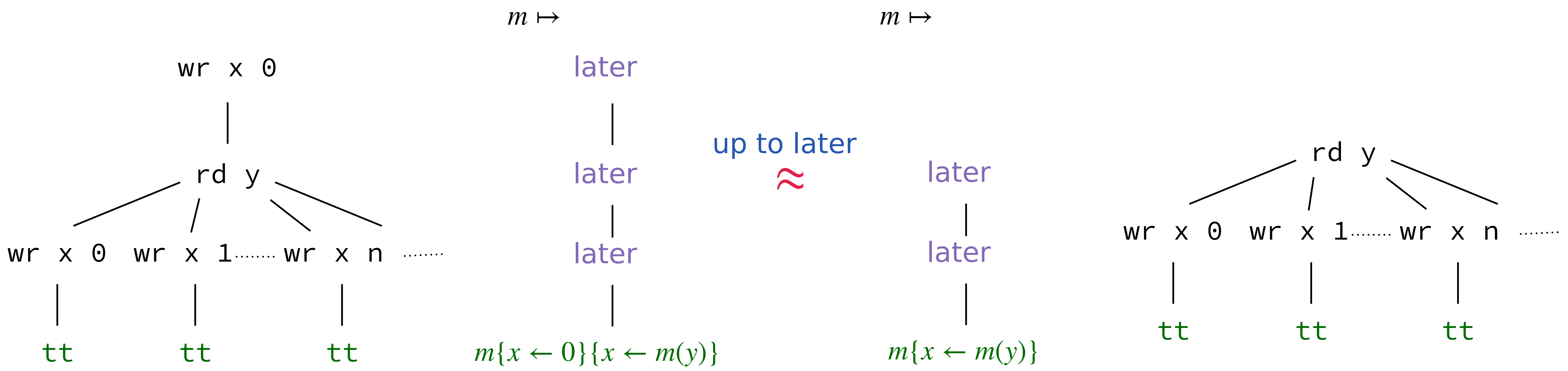


Programs as Stateful Infinite Trees

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c$$

$p_2 \triangleq x := 0; x := y$

$p_3 \triangleq x := y$



Interaction Trees

A domain of computations shallow embedded in Coq

```
CoInductive itree (E: Type -> Type) (R: Type): Type :=
| Ret (r: R)
| Later (t: itree E R)
| Vis {X: Type} (e: E X) (k: X -> itree E R).
```

A value of the datatype (itree E R) represents:

- a potentially diverging computation,
- which may return a value of type R,
- while emitting during its execution visible events from the interface E.

Choice Trees

or

Representing
Nondeterministic, Recursive, and Impure
Programs in Coq

Nondeterministic branching

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c \mid br\ c_1\ or\ c_2 \mid stuck \mid print$$

$br\ c_1\ or\ c_2$: either branch can be executed

Sounds quite easy to model as an itree: let's have a (toss : E bool) event

$$[br\ c_1\ or\ c_2] \triangleq \begin{tikzcd} & \texttt{toss} \arrow[dl, "\texttt{true}"'] \arrow[dr, "\texttt{false}"] & \\ {[\texttt{c1}]} & & {[\texttt{c2}]} \end{tikzcd}$$

Nondeterministic branching

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c \mid br\ c_1\ or\ c_2 \mid stuck \mid print$$

$br\ c_1\ or\ c_2$: either branch can be executed

Sounds quite easy to model as an itree: let's have a (toss : E bool) event

$[br\ c_1\ or\ c_2] \triangleq$

```
      toss
true /    \ false
[c1]       [c2]
```

$\not\approx$

```
      toss
true /    \ false
[c2]       [c1]
```

At this stage, toss is not commutative

nor idempotent, nor associative

Question: what is the structure into which we should interpret toss?

Nondeterministic branching

Question: what is the structure into which we should interpret toss?

An idea from Vellvm: *sets* of trees?

$$\mathcal{I}([br\ c_1\ or\ c_2]) \triangleq [c_1] \cup [c_2]$$ (In Coq: `itree E X -> Prop`)

❌ We lose executability, monadic laws, everything becomes harder...

Nondeterministic branching

Question: what is the structure into which we should interpret toss?

An idea from Vellvm: *sets* of trees?

$\mathcal{I}([br\ c_1\ or\ c_2]) \triangleq [c_1] \cup [c_2]$ (In Coq: `itree E X -> Prop`)

We lose executability, monadic laws, everything becomes harder...

This work: ctrees, what we believe to be the right structure

Nondeterministic branching: but what do we mean?

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2 \mid while\ b\ do\ c \mid br\ c_1\ or\ c_2 \mid stuck \mid print$$

$$p \triangleq br\ (while\ true\ do\ print)\ or\ stuck$$

Can the above program p be stuck?

Case 1:

$$\frac{}{br\ c_1\ or\ c_2 \to c_1}$$

$p \to stuck$ is possible

Case 2:

$$\frac{c_1 \to c_1'}{br\ c_1\ or\ c_2 \to c_1'}$$

$p \to stuck$ is not possible

The system may **become** either branch

The system may **take a transition** offered by either branch

$p \triangleq br\ (while\ true\ do\ print)\ or\ stuck$

Let's take the perspective of an LTS

Case 0 (itree):

$$\frac{}{br\ c_1\ or\ c_2 \xrightarrow{true} c_1}$$

$p \xrightarrow{true} stuck$ possible

Case 1:

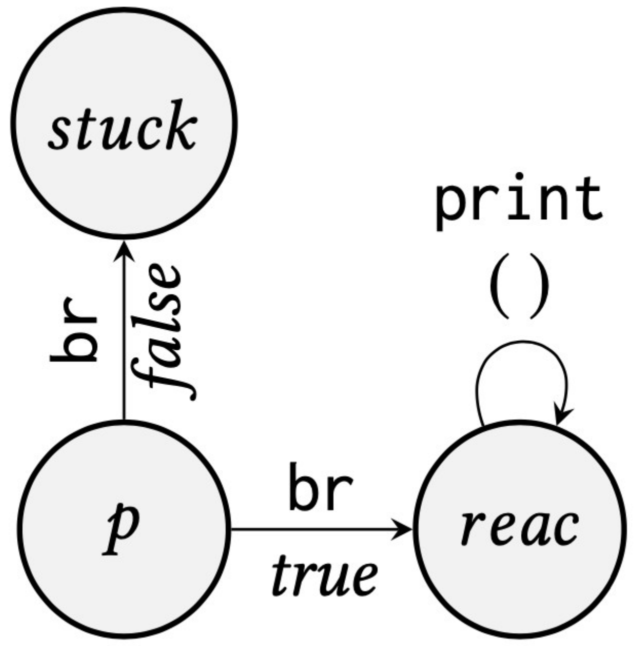
$$\frac{}{br\ c_1\ or\ c_2 \to c_1}$$

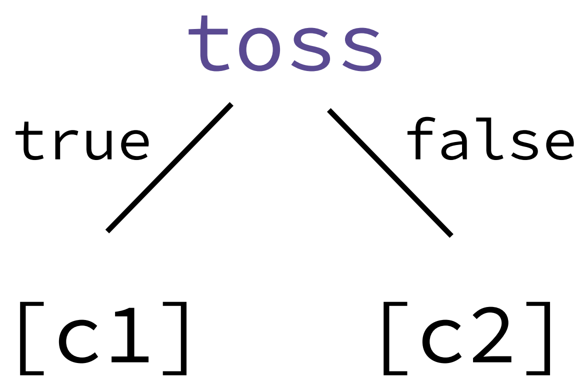
$p \to stuck$ possible

Case 2:

$$\frac{c_1 \to c_1'}{br\ c_1\ or\ c_2 \to c_1'}$$

$p \to stuck$ not possible





External event,
we observe which event happened,
what branch we took

$p \triangleq br\ (while\ true\ do\ print)\ or\ stuck$

Let's take the perspective of an LTS

Case 0 (itree):

$$\frac{}{br\ c_1\ or\ c_2 \xrightarrow{true} c_1}$$

$p \xrightarrow{true} stuck$ possible

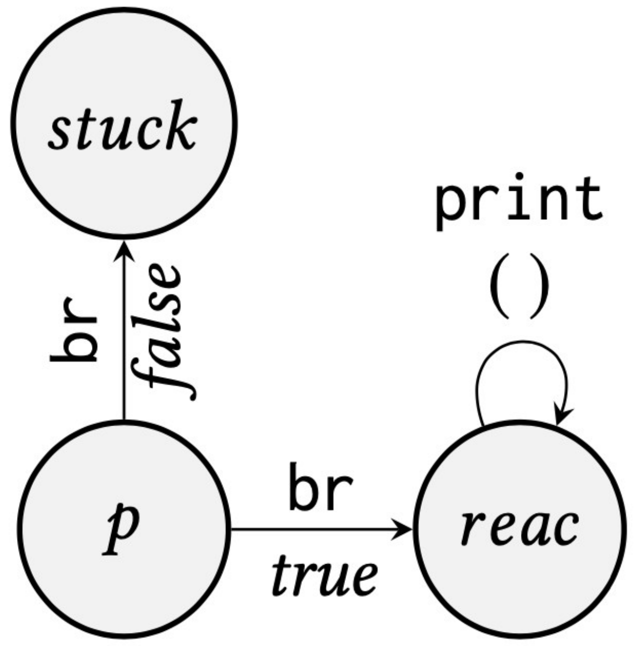
Case 1:

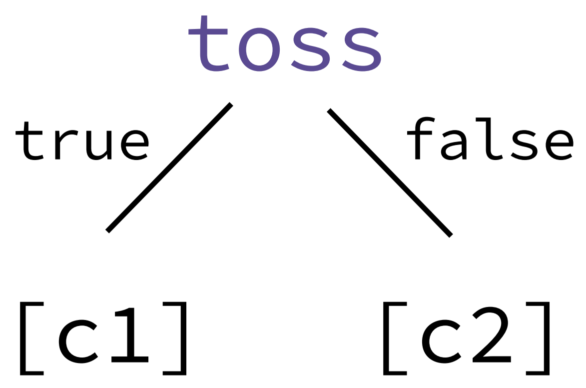
$$\frac{}{br\ c_1\ or\ c_2 \rightarrow c_1}$$

$p \rightarrow stuck$ possible

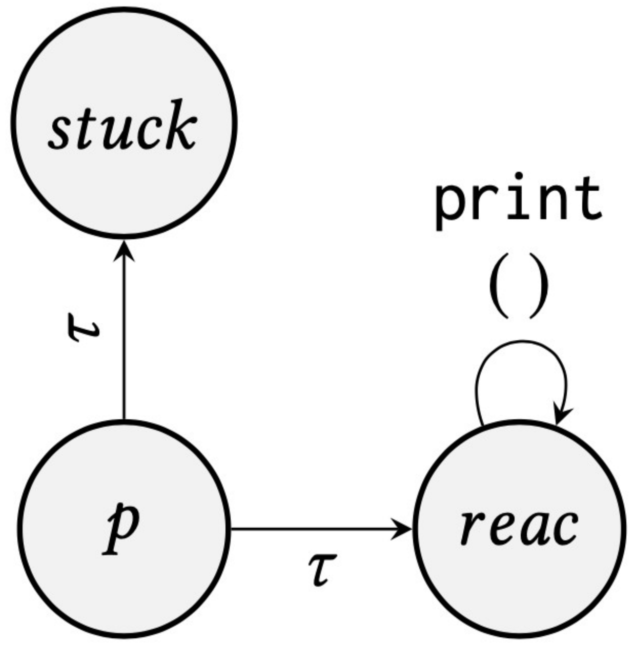
Case 2: $$\frac{c_1 \rightarrow c_1'}{br\ c_1\ or\ c_2 \rightarrow c_1'}$$

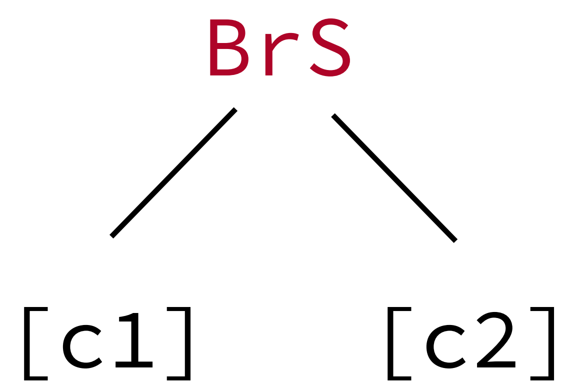
$p \rightarrow stuck$ not possible





External event,
we observe which event happened,
what branch we took





Stepping branch,
we observe that a branch
has been taken

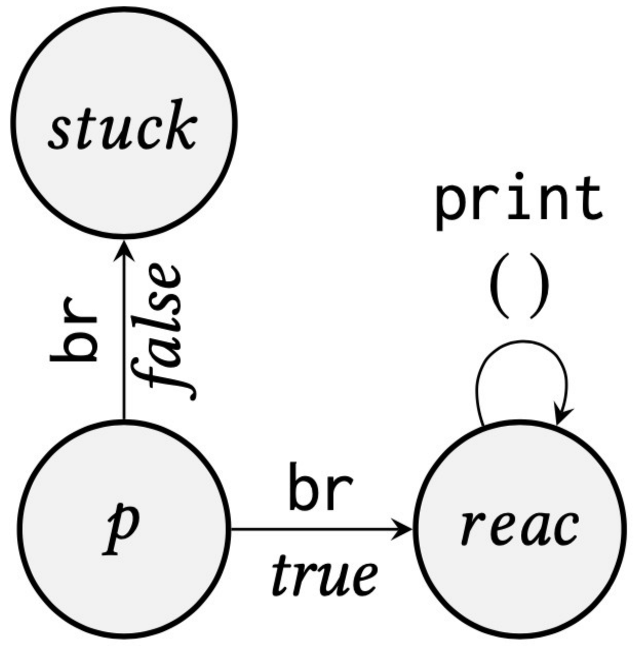
$p \triangleq br\ (while\ true\ do\ print)\ or\ stuck$

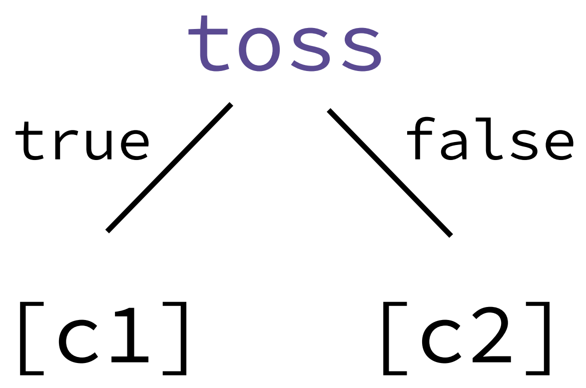
Let’s take the perspective of an LTS

Case 0 (itree):

$$\frac{}{br\ c_1\ or\ c_2 \xrightarrow{true} c_1}$$

$p \xrightarrow{true} stuck$ possible



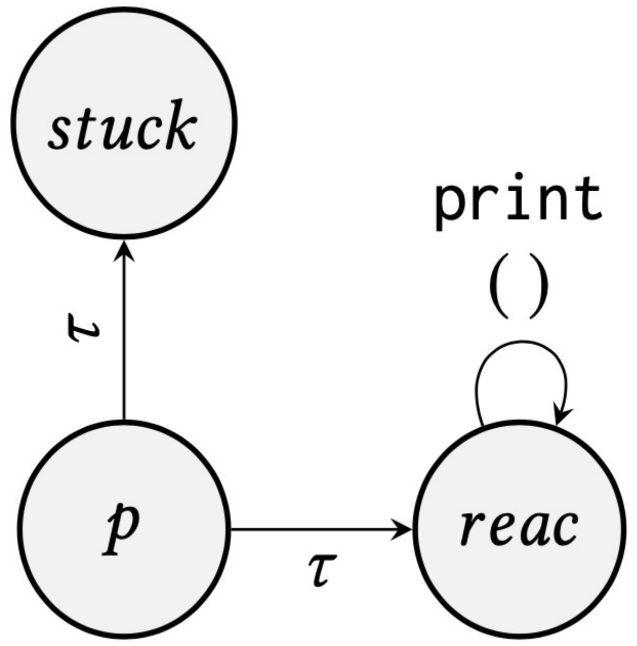


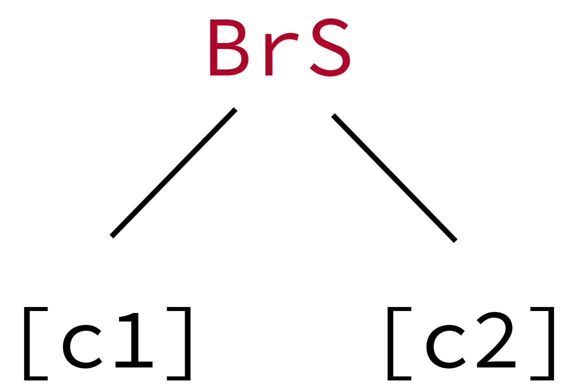
External event,
we observe which event happened,
what branch we took

Case 1:

$$\frac{}{br\ c_1\ or\ c_2 \to c_1}$$

$p \to stuck$ possible



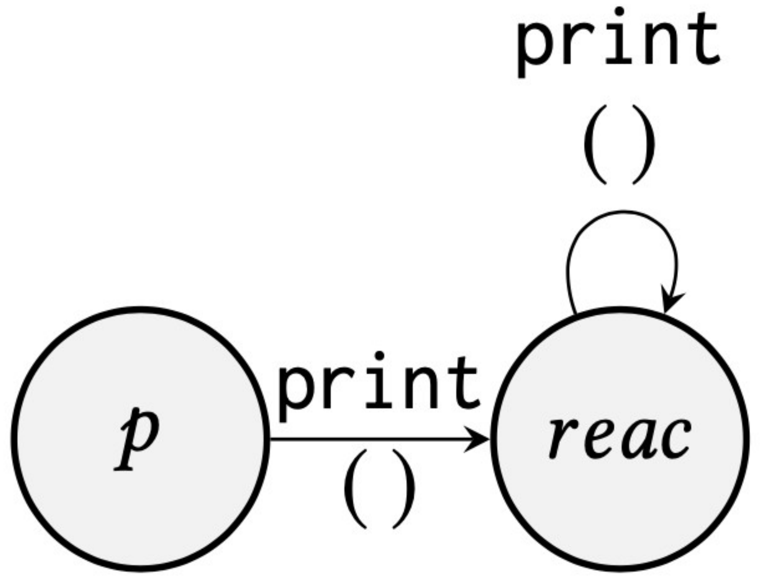


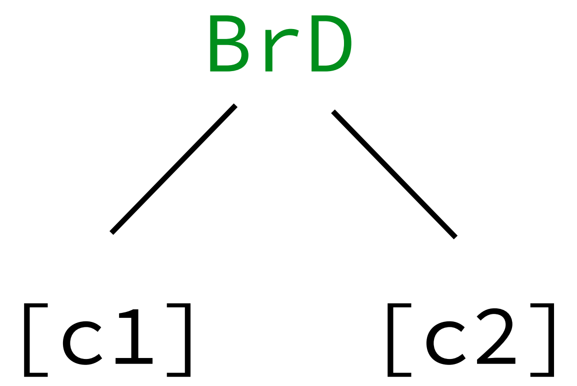
Stepping branch,
we observe that a branch
has been taken

Case 2:

$$\frac{c_1 \to c_1'}{br\ c_1\ or\ c_2 \to c_1'}$$

$p \to stuck$ not possible

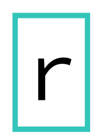




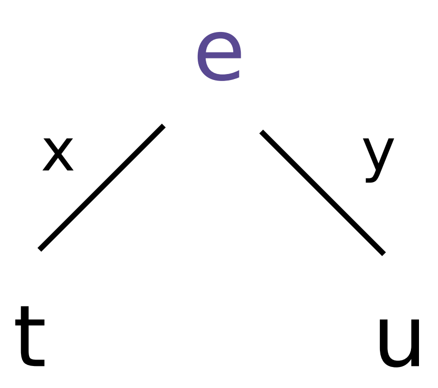
Delayed branch,
there’s a branch,
but we don’t observe it

Choice trees

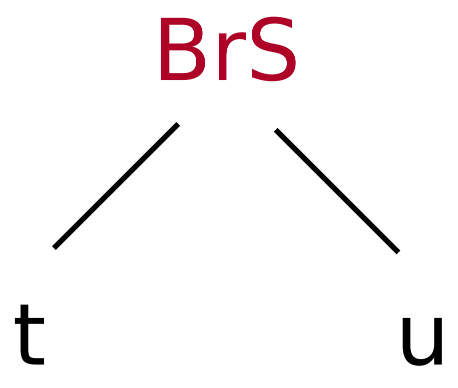
A *ctree E R* models a computation as a potentially infinite tree made of:



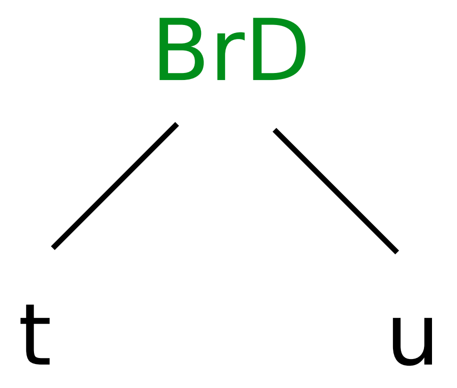
Leaves,
pure computations
(of type R)



External events,
interaction with an environment
(as described by E)



Stepping branches,
an internal choice which may
be observed



Delayed branches,
an internal choice that
only allows to try reaching
an observable action

```
CoInductive ctree (E: Type -> Type) (R: Type): Type :=
| Ret (r: R)
| Vis {X: Type} (e: E X) (k: X -> ctree E R)
| BrS {n: nat}          (k: fin n -> ctree E R)
| BrD {n: nat}          (k: fin n -> ctree E R)
```

CTrees, LTSs and Bisimulations

Bisimulation over ctrees

$\boxed{\text{Question: when should two ctrees be deemed equivalent?}}$

There has already been a lot of work on equivalence of LTSs,

Let’s build LTSs from ctrees!

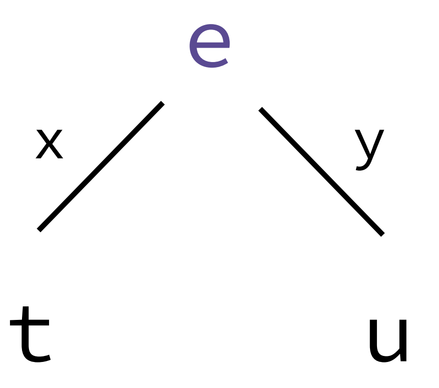
Bisimulation over ctrees

Question: when should two ctrees be deemed equivalent?

$$label ::= val\ x \mid obs\ e\ x \mid \tau$$

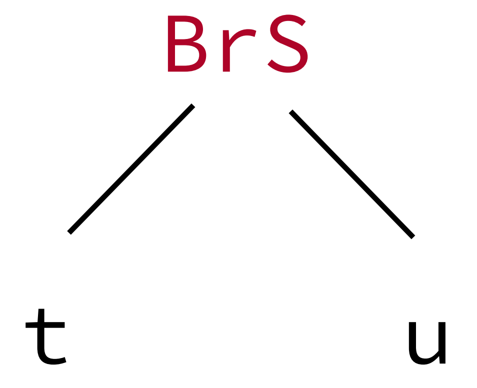
$\boxed{\texttt{r}} \xrightarrow{val\ r} \varnothing$

Leaves,
pure computations
(of type R)



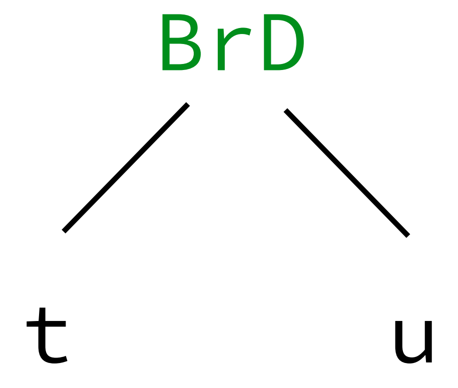
$\xrightarrow{obs\ e\ x} t$

External events,
interaction with an environment
(as described by E)



$\xrightarrow{\tau} t$

Stepping branches,
an internal choice which may
be observed



$\xrightarrow{l} t'$

$if\ t \xrightarrow{l} t'$

(inductively)

Delayed branches,
an internal choice that
only allows to try reaching
an observable action

Bisimulation over LTSs

Question: when should two ctrees be deemed equivalent?

Let $(\mathcal{S}, \xrightarrow{l})$ be a LTS, $\mathscr{R}$ a relation on $\mathcal{S}$ is a simulation if:

$$\begin{tikzcd} P \arrow[r, dash, "\mathscr{R}"] \arrow[d, "l"'] & Q \\ P' & \end{tikzcd}$$

Bisimulation over LTSs

$\boxed{\text{Question: when should two ctrees be deemed equivalent?}}$

Let $(\mathcal{S}, \xrightarrow{l})$ be a LTS, $\mathcal{R}$ a relation on $\mathcal{S}$ is a simulation if:

$$\begin{tikzcd} P \arrow[r, dash, "\mathcal{R}"] \arrow[d, "l"'] & Q \arrow[d, dotted, "l"] \\ P' \arrow[r, dotted, dash, "\mathcal{R}"] & Q' \end{tikzcd}$$

Similarity is then defined as the largest simulation

A whole zoo have been studied: weak, complete, branching, ...

Bisimulation over ctrees

Question: when should two ctrees be deemed equivalent?

Answer: if their underlying LTSs are bisimilar!

$$sb\ \mathscr{R}\ s\ t \triangleq$$

$$\forall l, t, s, s', s \xrightarrow{l} s' \Rightarrow \exists t', s' \mathscr{R} t' \wedge t \xrightarrow{l} t'$$

and

$$\forall l, s, t, t', t \xrightarrow{l} t' \Rightarrow \exists s', s' \mathscr{R} t' \wedge s \xrightarrow{l} s'$$

$$\begin{tikzcd} P \arrow[r, dash, "\mathscr{R}"] \arrow[d, "l"'] & Q \arrow[d, dotted] \\ P' \arrow[r, dotted, dash, "\mathscr{R}"] & Q' \end{tikzcd}$$

For Coq enthusiasts

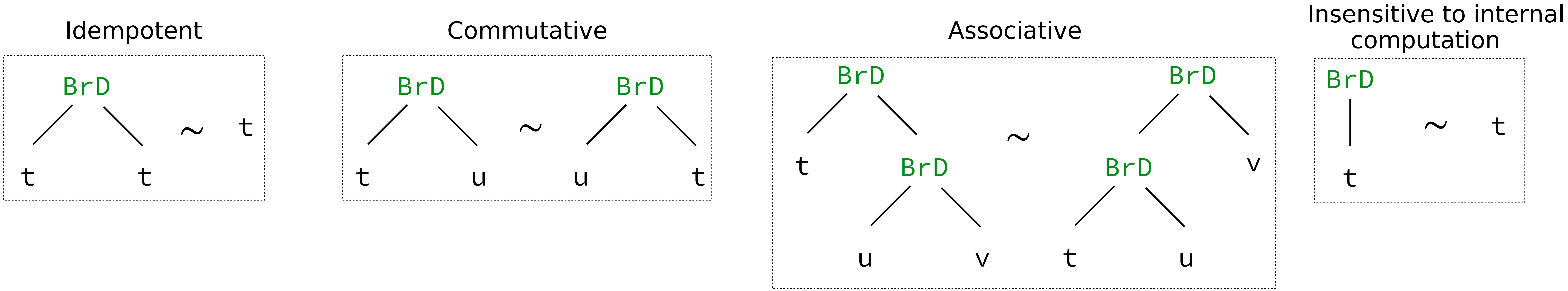
We tie the coinductive knot using Pous's coinduction library

Bisimulation over ctrees

Question: when should two ctrees be deemed equivalent?

Answer: if their underlying LTSs are bisimilar!

We recover the right algebraic laws for non-determinism



Bisimulation over ctrees

Insensitive to internal computation

$$\begin{array}{c} \texttt{BrD} \\ | \\ \texttt{t} \end{array} \sim \texttt{t}$$

Do we have the same with `BrS`?

Insensitive to internal computation (?)

$$\begin{array}{c} \texttt{BrS} \\ | \\ \texttt{t} \end{array} \sim \texttt{t}$$

Bisimulation over ctrees

Insensitive to internal computation

BrD
| $\sim$ t
t

Do we have the same with BrS?

Insensitive to internal computation (?)

BrS
| $\cancel{\sim}$ $\approx$ t
t

Three main equivalences over ctrees

(Coinductive) structural equality

Strong bisimilarity ($\sim$)

Weak bisimilarity ($\approx$)

And trace equivalence, simulations, and potentially all their variants

CTrees and Interpretation

- CTrees are an adequate *target* monad into which one can interpret toss

`h(toss) ≜ BrD 2`

`interp h : itree (Toss + E) ~> ctree E`

$$\boxed{t \approx u \longrightarrow \texttt{interp h } t \sim \texttt{interp h } u}$$

- They of course themselves still support interpretation

  (targets must explain how they internalise branching nodes)

- Branching nodes can be « interpreted » as well

  ⇝ low level notion of scheduler
  ⇝ formal refinements (complete simulations) in Coq
  ⇝ practical testing in OCaml

Calculus of Communicating Systems [Milner, 1980]

$$P ::= 0 \mid l \cdot P \mid P \oplus Q \mid P \parallel Q \mid \nu c \cdot P \mid !P$$

Communication

Internal choice

Parallel composition

Channel restriction

Replication

Goal: build a computable model of ccs using ctrees

Calculus of Communicating Systems [Milner, 1980]

$$P ::= \ 0 \ \mid \ l \cdot P \ \mid \ P \oplus Q \ \mid \ P \parallel Q \ \mid \ \nu c \cdot P \ \mid !P$$

- We establish ccs's traditional equational theory w.r.t. ~ on our model
- We prove an adequacy result against ccs's operational semantics

$$[P] \sim [Q] \text{ iff } P \sim_{op} Q$$

- Our model is computable: we can execute by extraction

  ⇝ With a caveat: restriction kills branches,
  one needs to avoid these dead branches

Cooperative scheduling

$$com ::= \bullet \mid x := e \mid c_1; c_2 \mid \mathit{while}\ b\ \mathit{do}\ c \mid \mathit{fork}\ c_1\ c_2 \mid \mathit{yield}$$

- Two layered computable model:
  - compositional construction with explicit fork and yield events
  - top-level interleaving combinator
- Combination of non-determinism with stateful computations
- Selected set of algebraic equations (further work needed there)

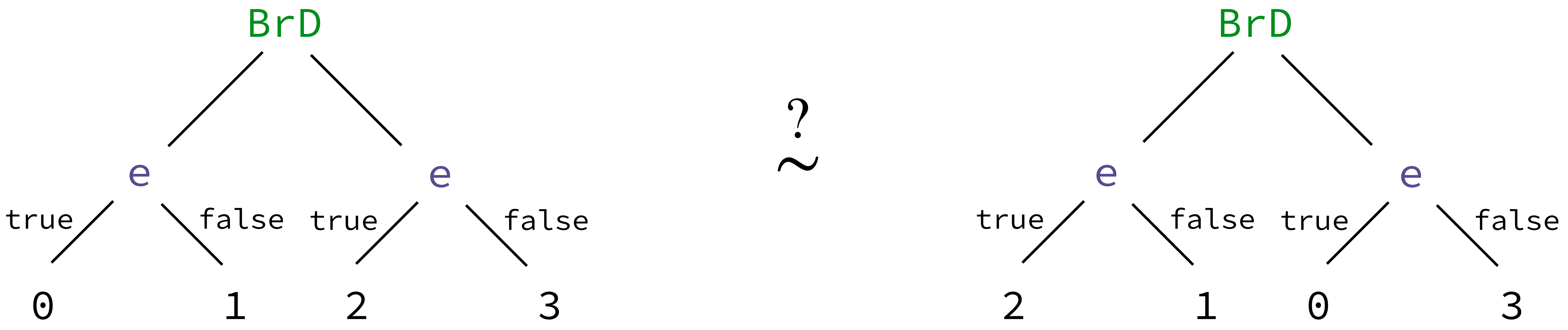
Ctrees Open Question 1: BrD or BrS?

$p_1 \triangleq \textit{while true do} \bullet$

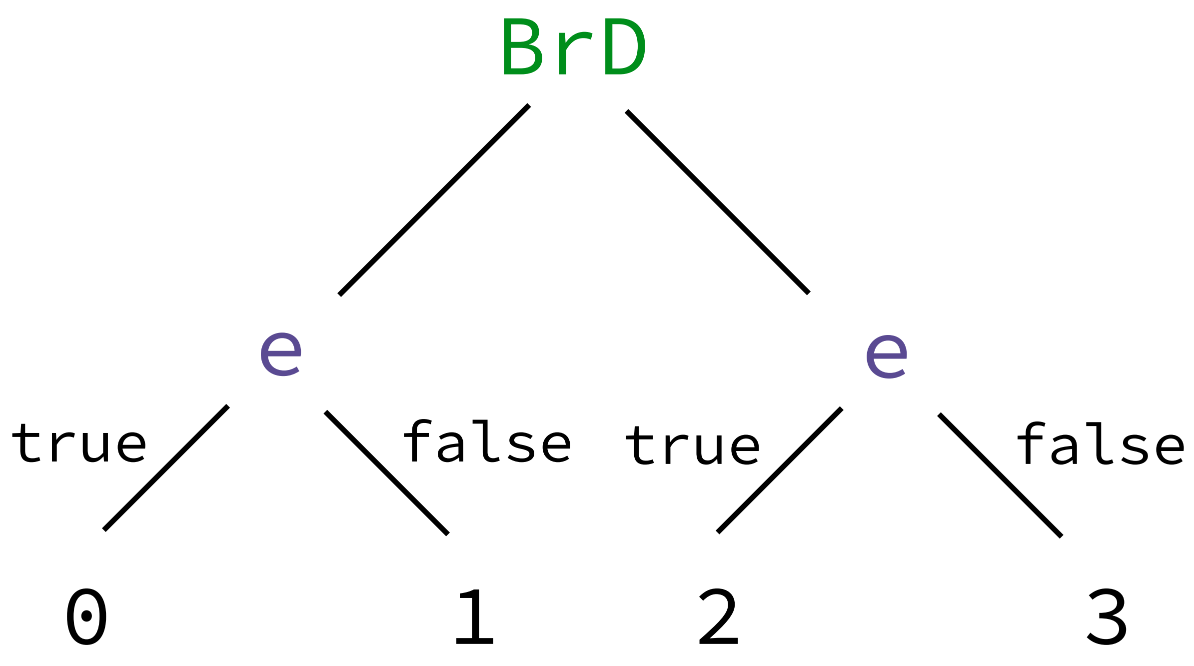
$$\begin{array}{c} \texttt{later} \\ | \\ \texttt{later} \\ | \\ \texttt{later} \\ \vdots \end{array} \quad \mapsto \quad \begin{array}{c} \texttt{BrD} \\ | \\ \texttt{BrD} \\ | \\ \texttt{BrD} \\ \vdots \end{array} \quad \texttt{or} \quad \begin{array}{c} \texttt{BrS} \\ | \\ \texttt{BrS} \\ | \\ \texttt{BrS} \\ \vdots \end{array} \quad ?$$

More generally: BrD and strong bisimulation or BrS and weak?

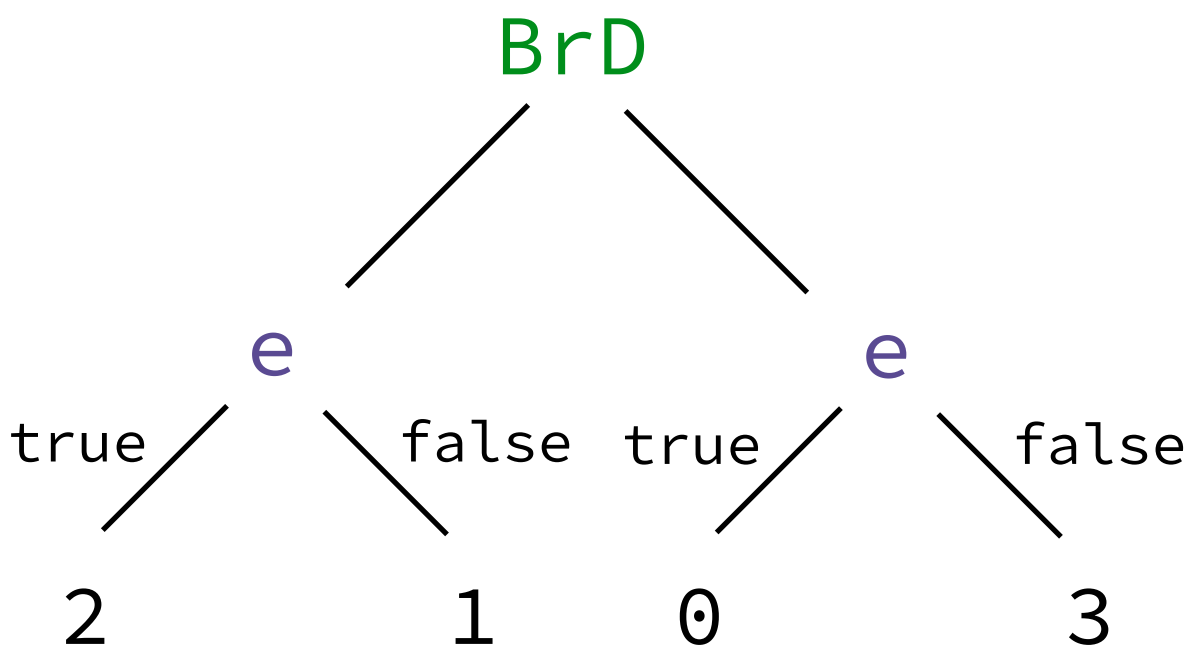
CTrees Open Question 2: Do we have the right LTS?



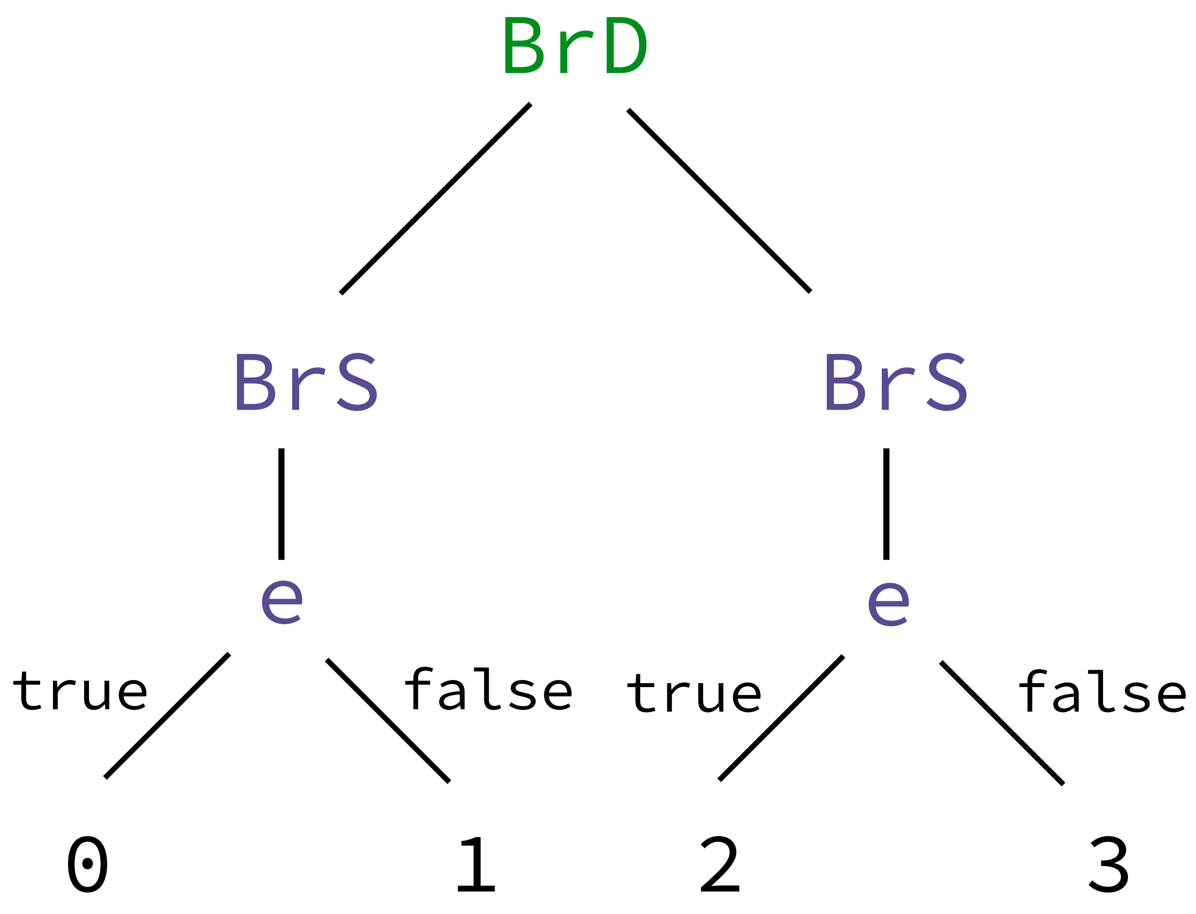
CTrees Open Question 2: Do we have the right LTS?



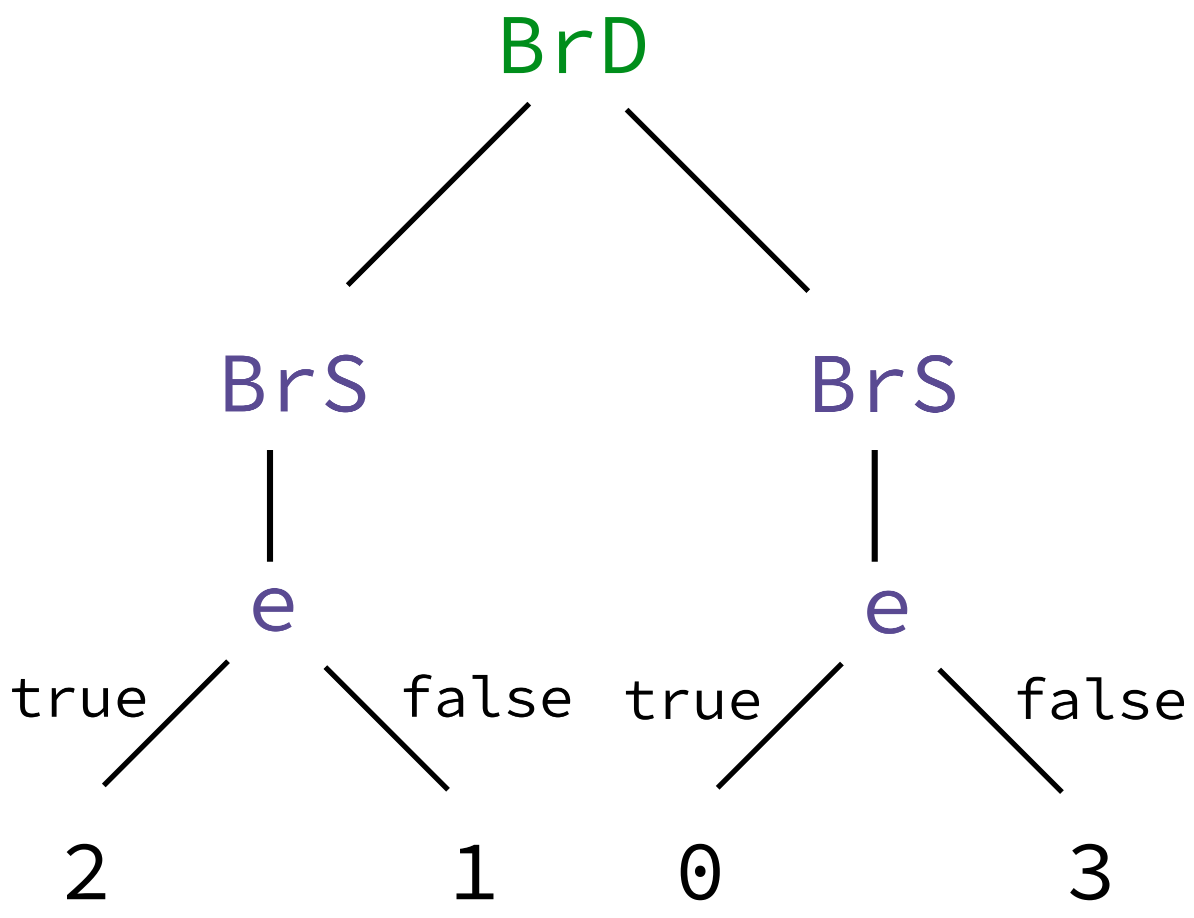
$\overset{?}{\sim}$



$$\frac{t \sim u}{interp\ h\ t \sim interp\ h\ u}$$



$\not\sim$



Conclusion

Choice Trees in a Nutshell

Modelling non-determinism and concurrency as monadic interpreters

- We stick to the tree structure, with two new kinds of branching nodes
- Looking at the tree as an LTS sheds light to reason on their equivalence: the tools from the process algebra literature can be brought in
- Case studies suggest that the approach is viable!
- The representation still feels too large: avenue for improvement?

Implemented as a Coq library: https://github.com/vellvm/ctrees/

Accepted at POPL’23:
https://perso.ens-lyon.fr/yannick.zakowski/papers/ctrees.pdf