Choice Trees

Representing Nondeterministic, Recursive, and Impure Programs in Coq

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

Penn
University of Pennsylvania

~~Choice~~ Trees
Interaction

Representing
~~Nondeterministic~~, Recursive, and Impure
Programs in Coq

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

3 years ago, in New Orleans...

Modeling Computations in a Proof Assistant

Four core desiderata:

- Reusable components
- Compositional, whenever possible
- Executable (allows for testing)
- Supporting termination sensitive refinements

In a dependently typed theory

In the Coq Proof Assistant

A reusable library to define and reason about Monadic Interpreters

Interaction Trees, Summarily

At its core, two standard notions from the literature

The Free Monad [Swiestra 08, Kiselyov and Ishii 15, …]

The Delay Monad [Capretta 05]

Notion 1: The Free Monad

Effectful computations arise from their signature of operations

```
itree E X
```

My computation is a glorified piece of syntax

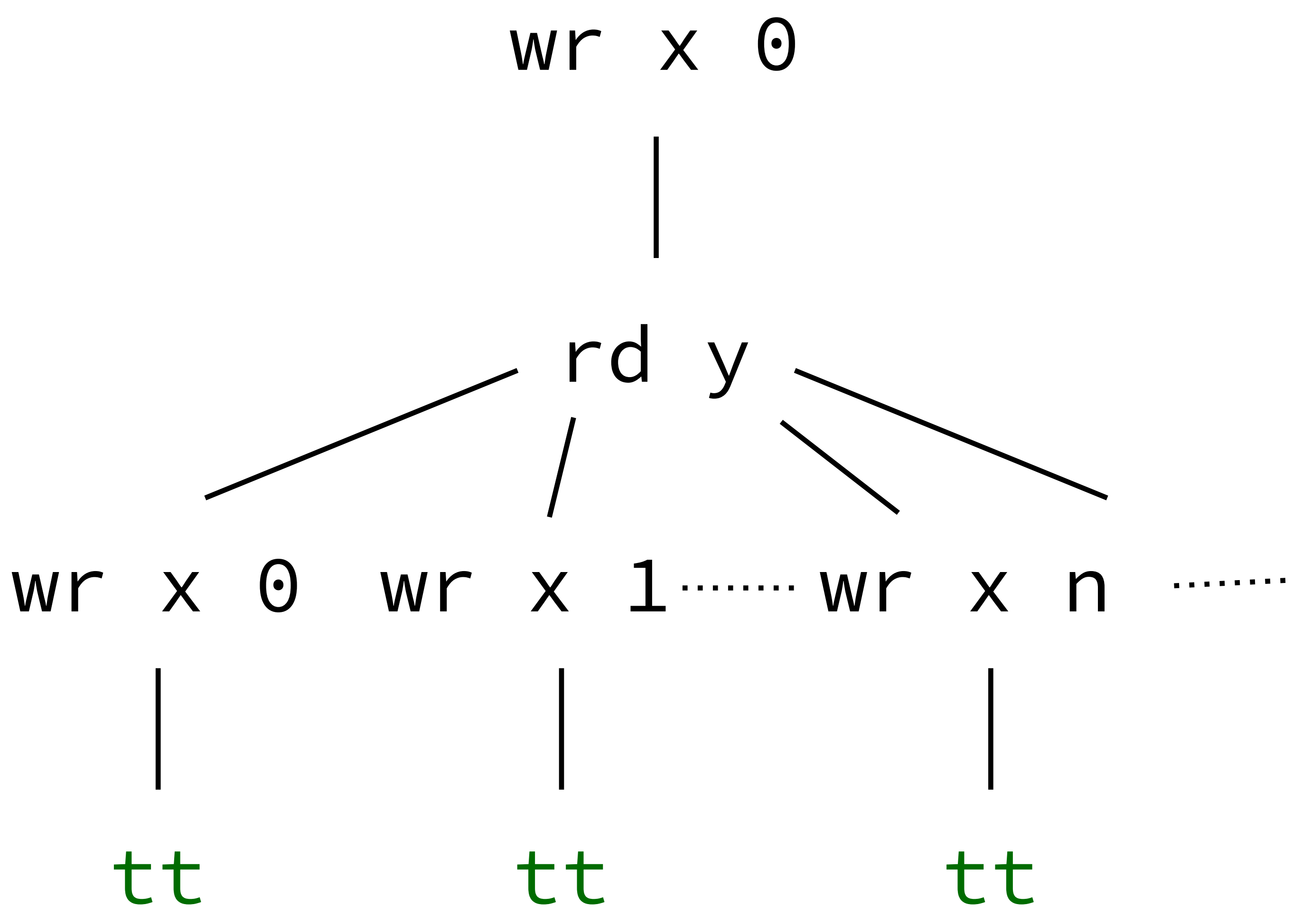
able to perform operations specified in E

in order to compute a value of type X

Programs as Trees

Imp programs are computations performing reads and writes

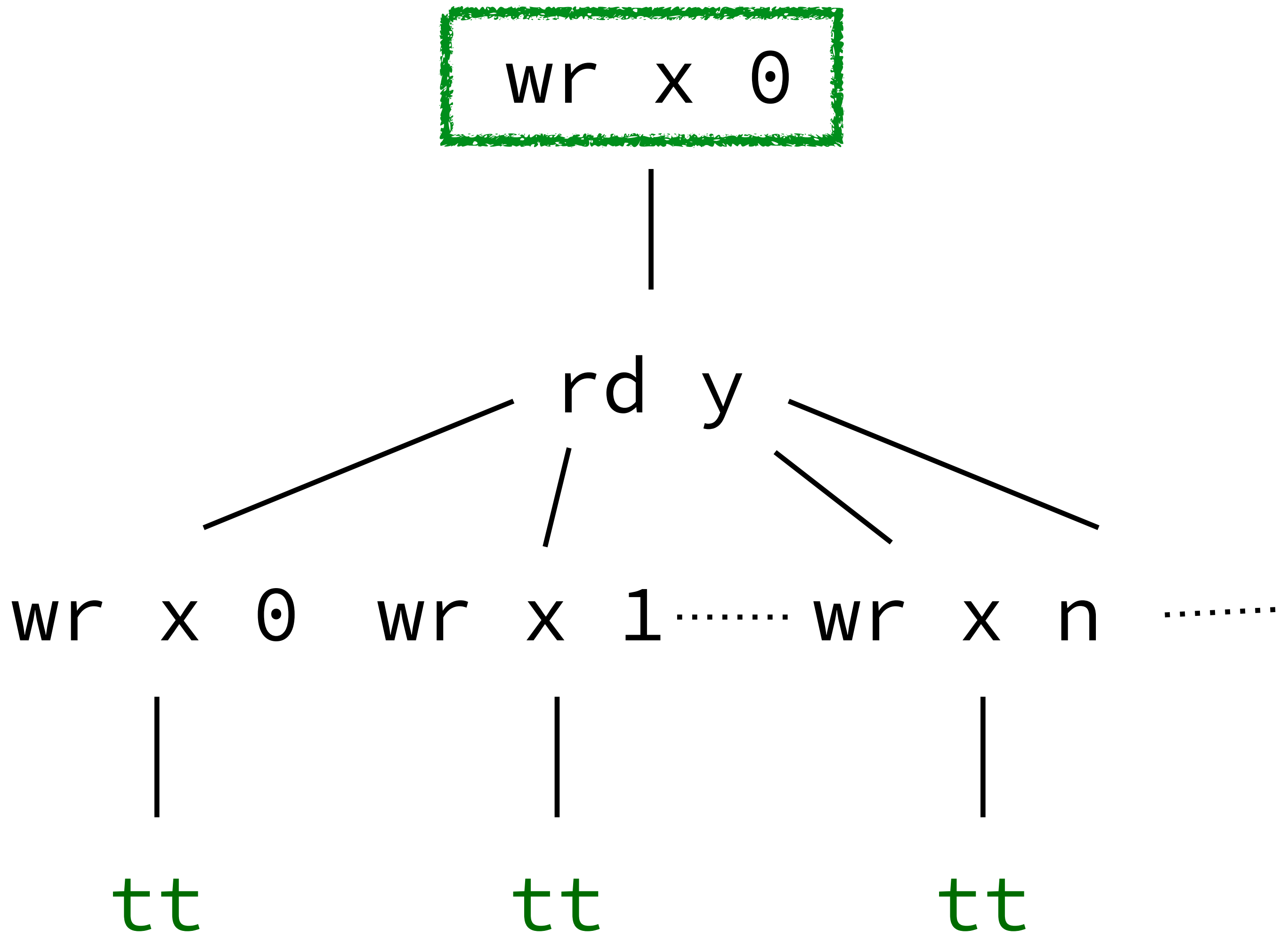
$$p \triangleq x := 0;\ x := y$$



Programs as Trees

Imp programs are computations performing reads and writes

$$p \triangleq \boxed{x := 0;}\ x := y$$



wr x 0

rd y

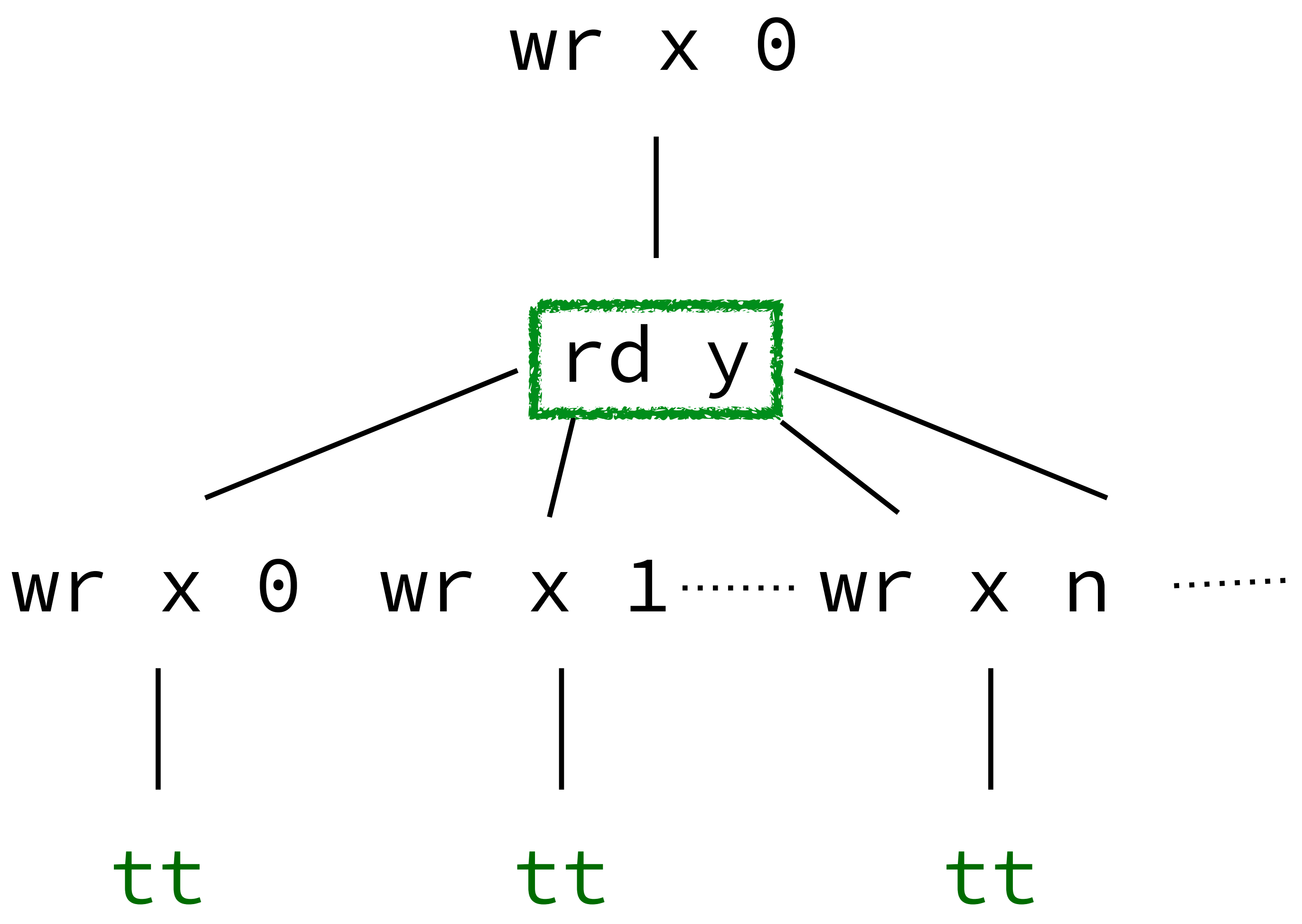
wr x 0 wr x 1 .... wr x n ....

tt tt tt

Programs as Trees

Imp programs are computations performing reads and writes

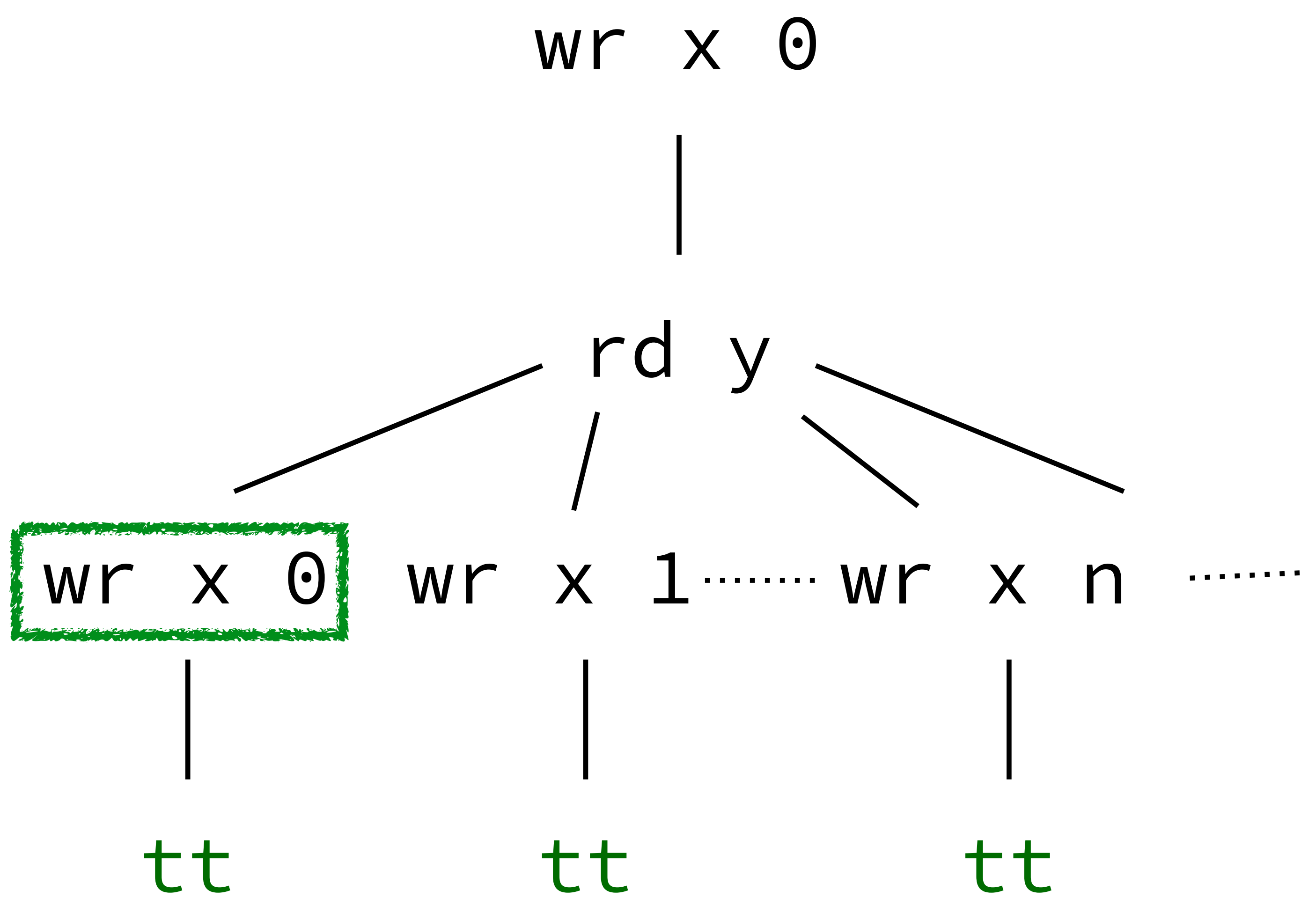
$$p \triangleq x := 0;\; x := \boxed{y}$$



Programs as Trees

Imp programs are computations performing reads and writes

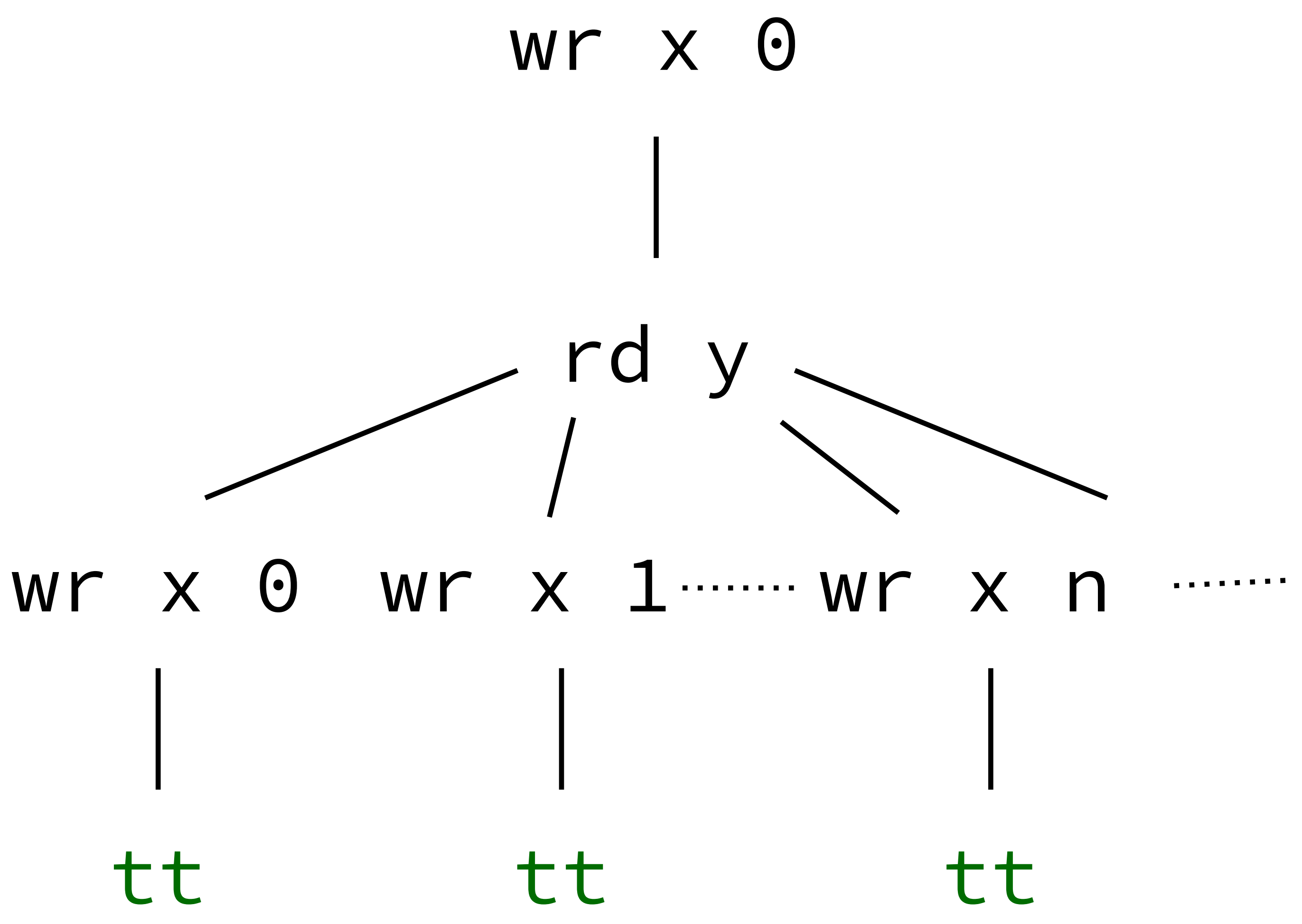
$p \triangleq x := 0; \boxed{x :=} y$

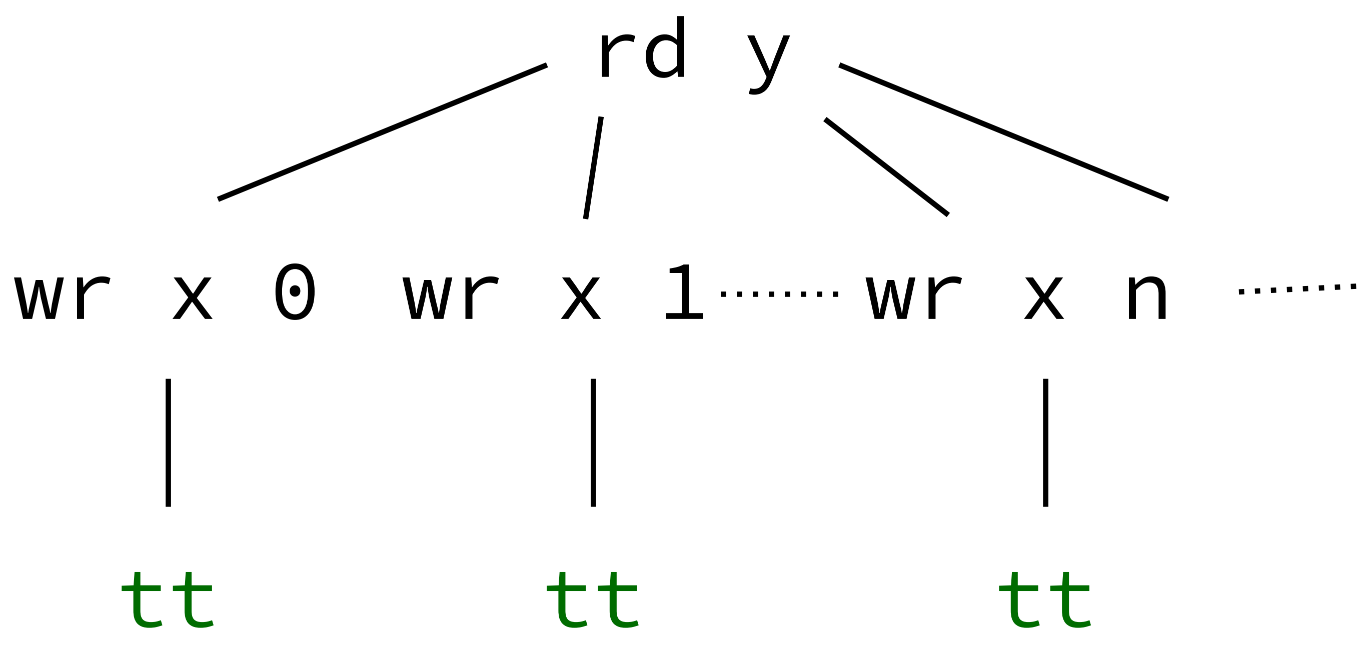


Programs as Trees

Imp programs are computations performing reads and writes

$p \triangleq x := 0;\ x := y$ $q \triangleq x := y$

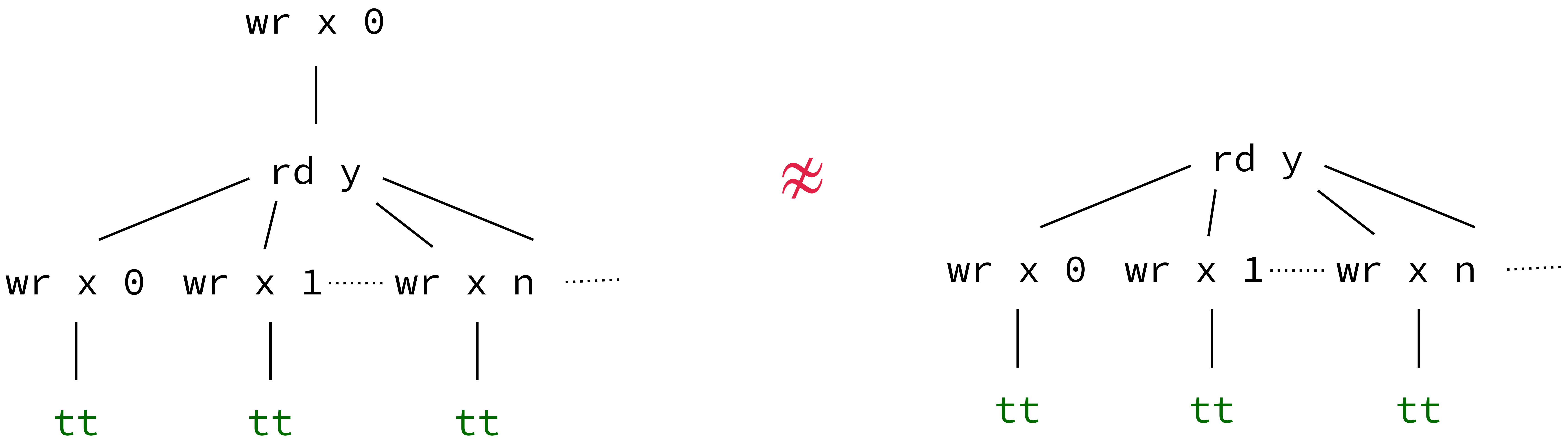




Programs as Trees

Imp programs are computations performing reads and writes

$p \triangleq x := 0;\ x := y$ semantically equivalent $q \triangleq x := y$

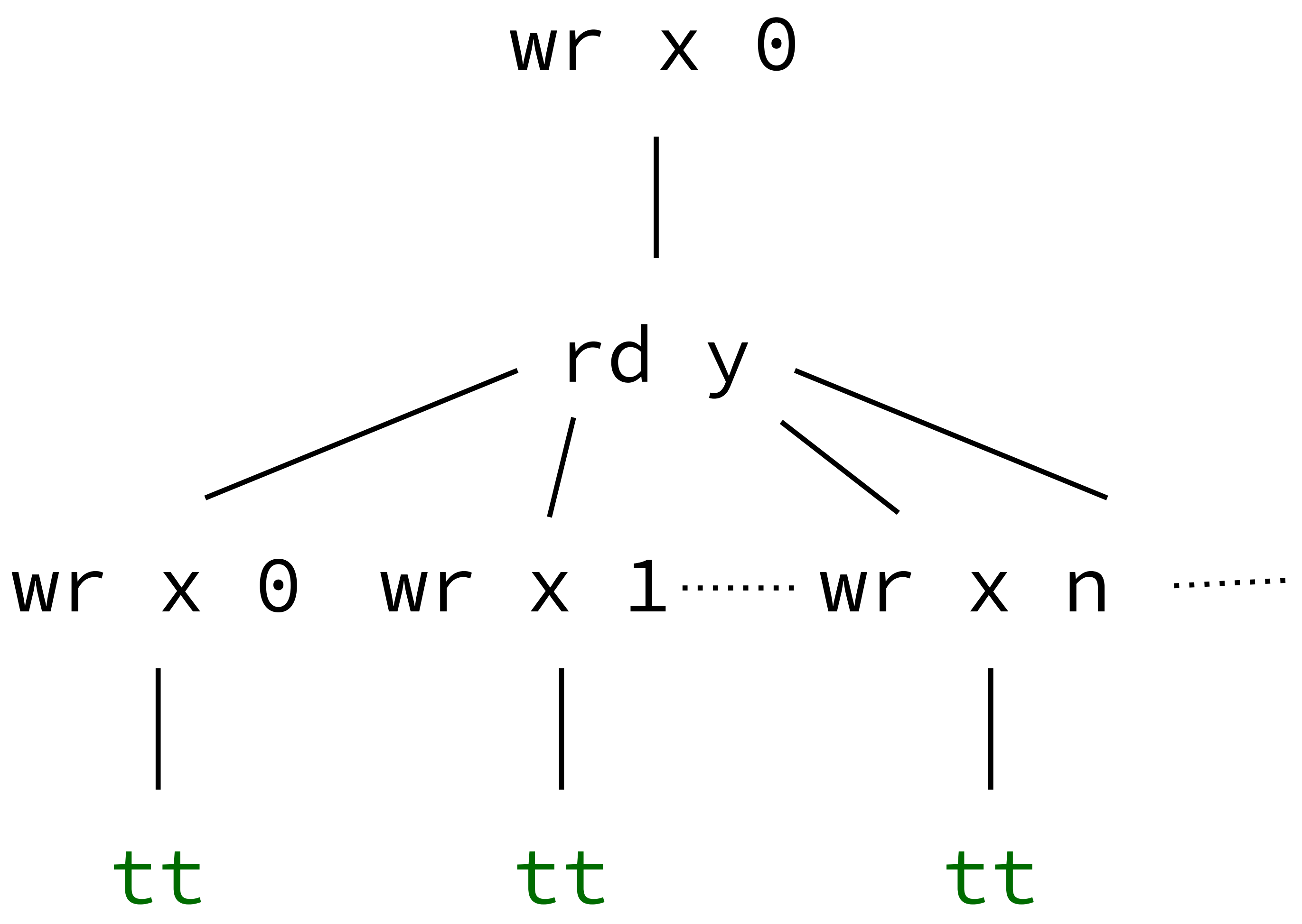


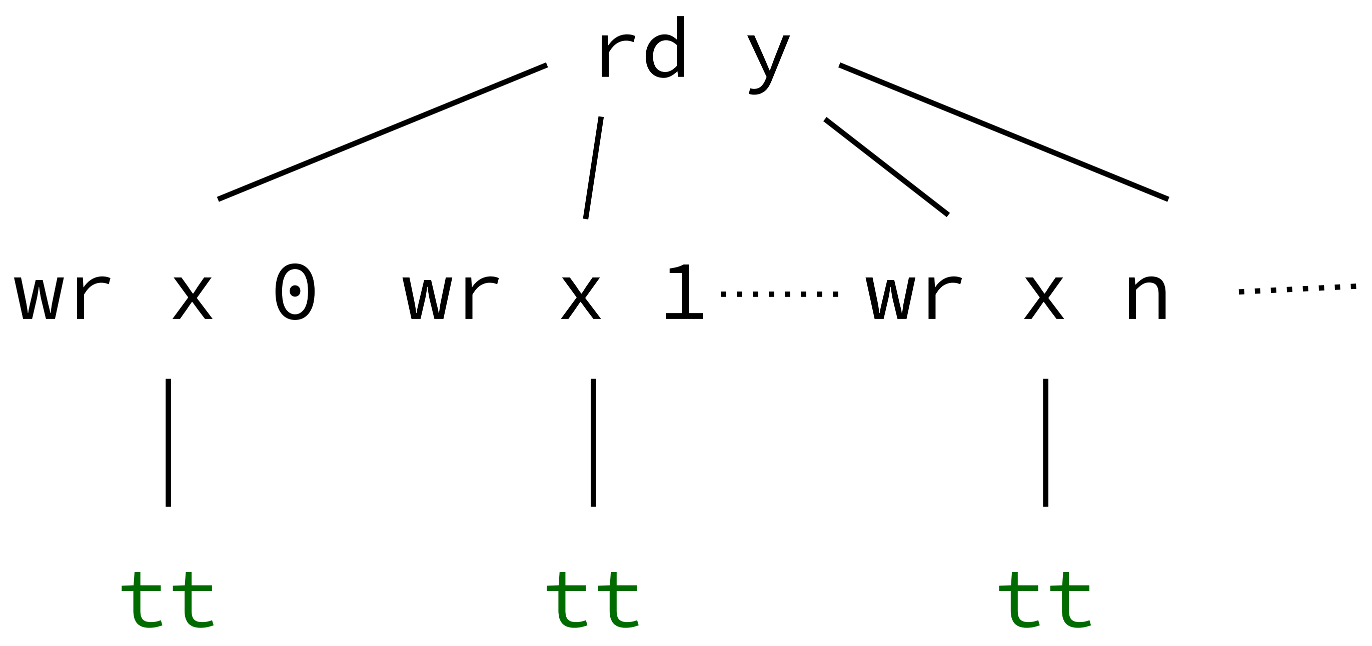
Programs as Stateful Trees

Imp programs are stateful computations

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

$q \triangleq x := y$



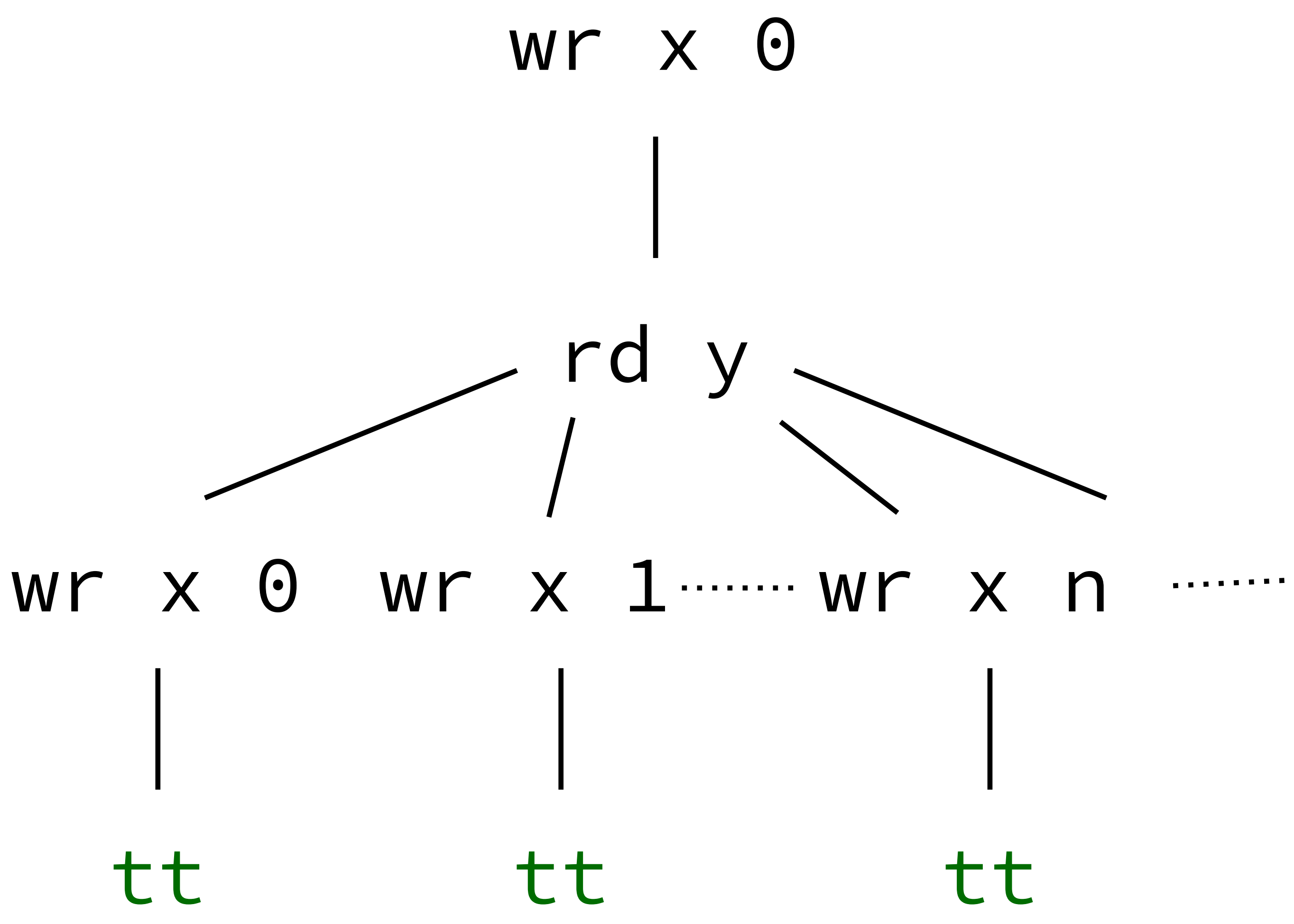


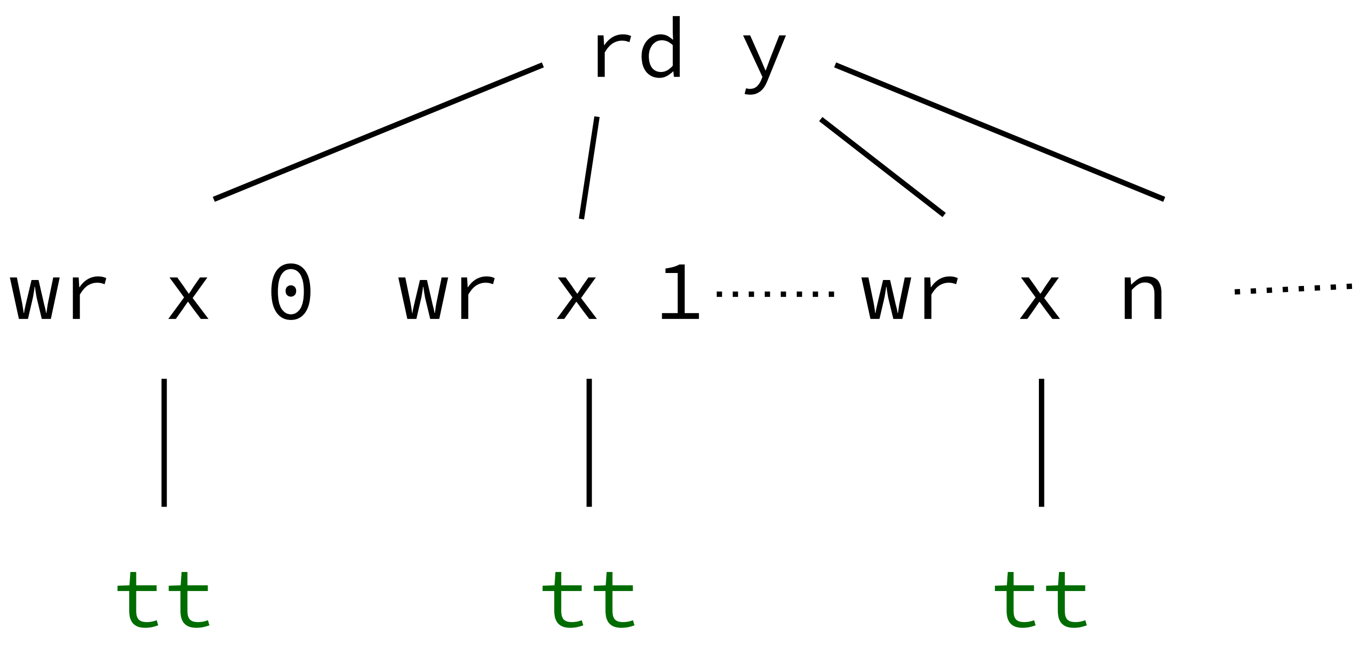
Programs as Stateful Trees

Imp programs are stateful computations

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

$q \triangleq x := y$



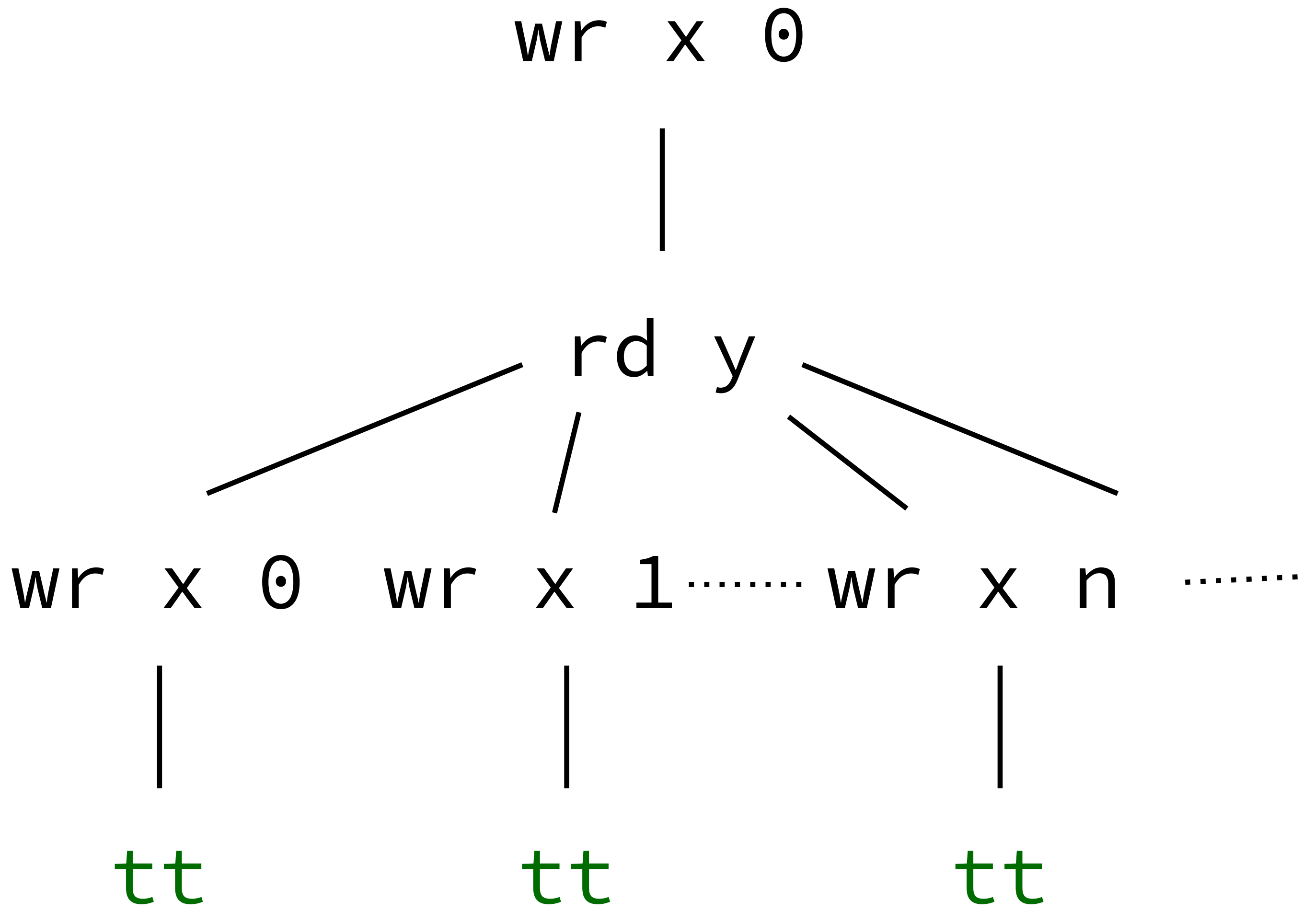


Programs as Stateful Trees

Imp programs are stateful computations

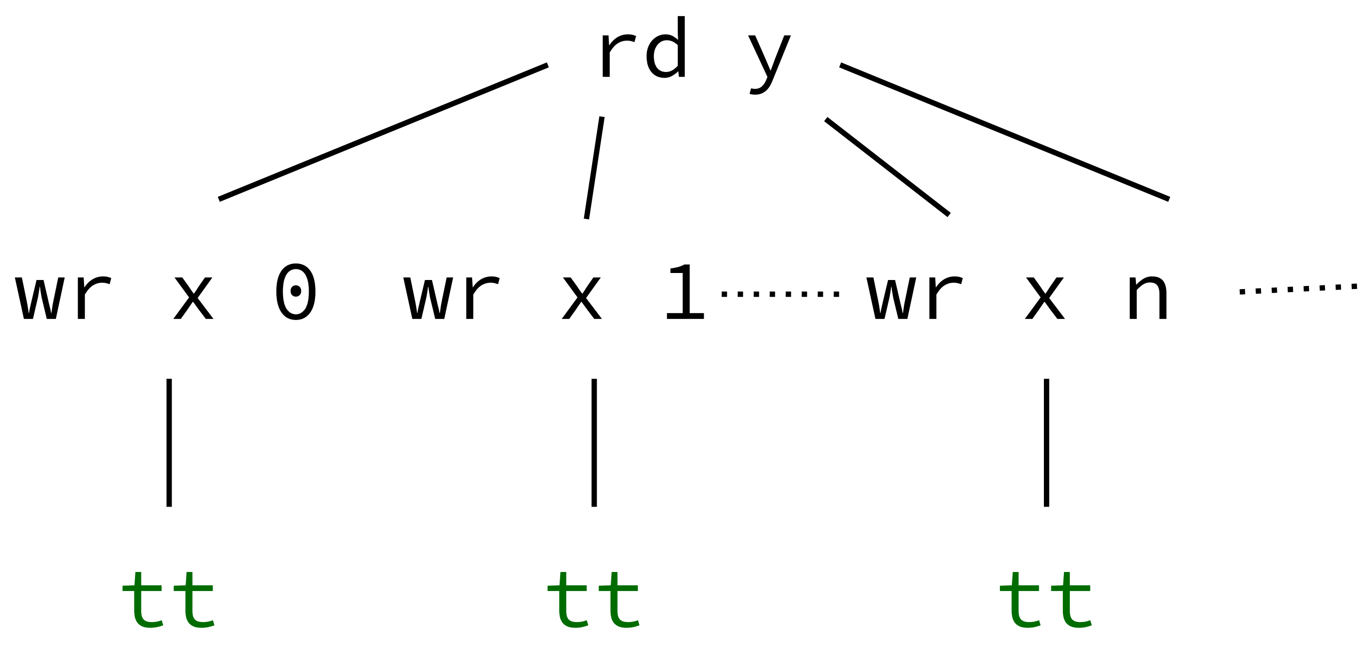
$p \triangleq x := 0; x := y$

$q \triangleq x := y$



$m \mapsto$

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

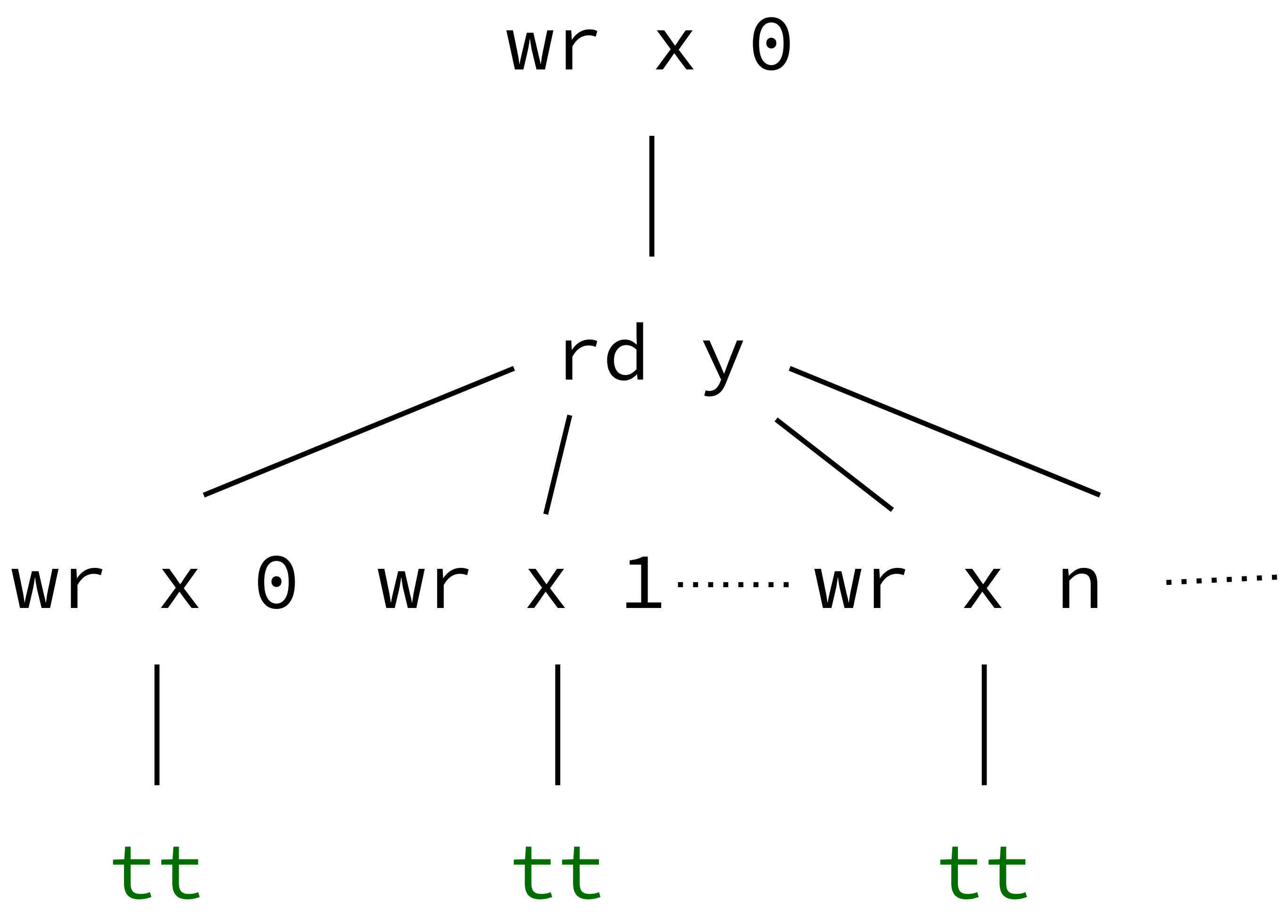


Programs as Stateful Trees

Imp programs are stateful computations

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

$q \triangleq x := y$

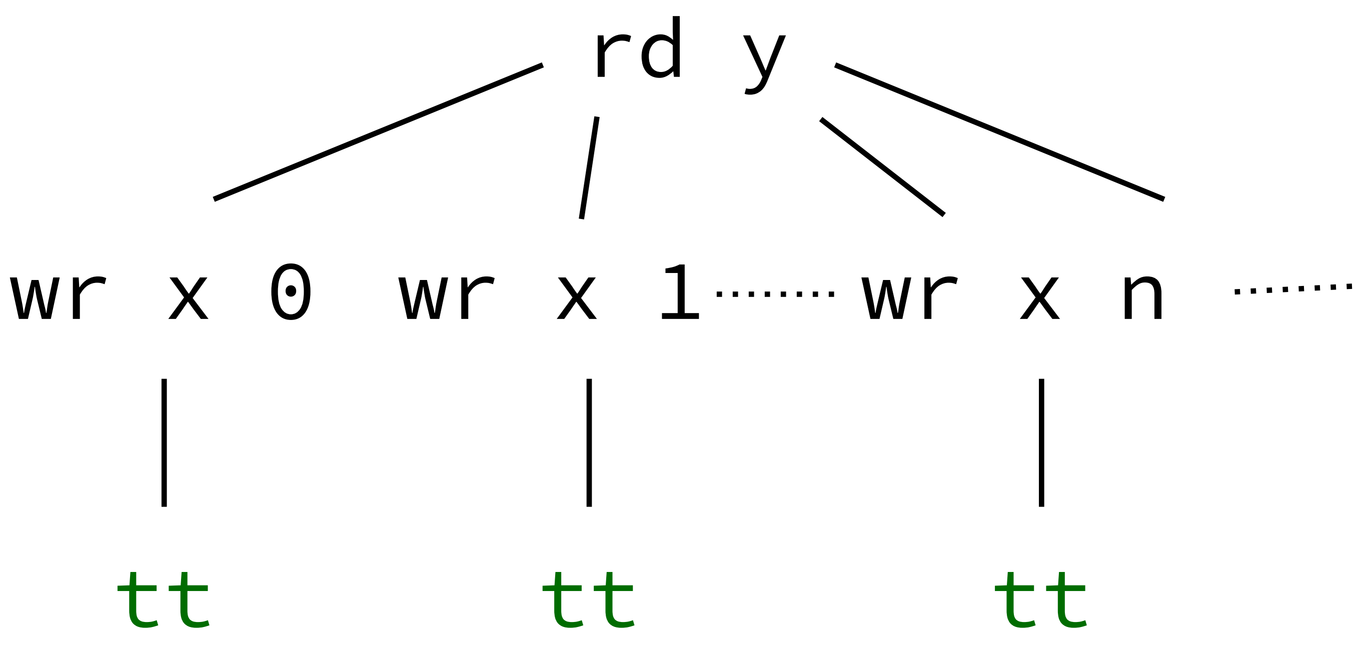


$m \mapsto$

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

$m \mapsto$

$m\{x \leftarrow m(y)\}$

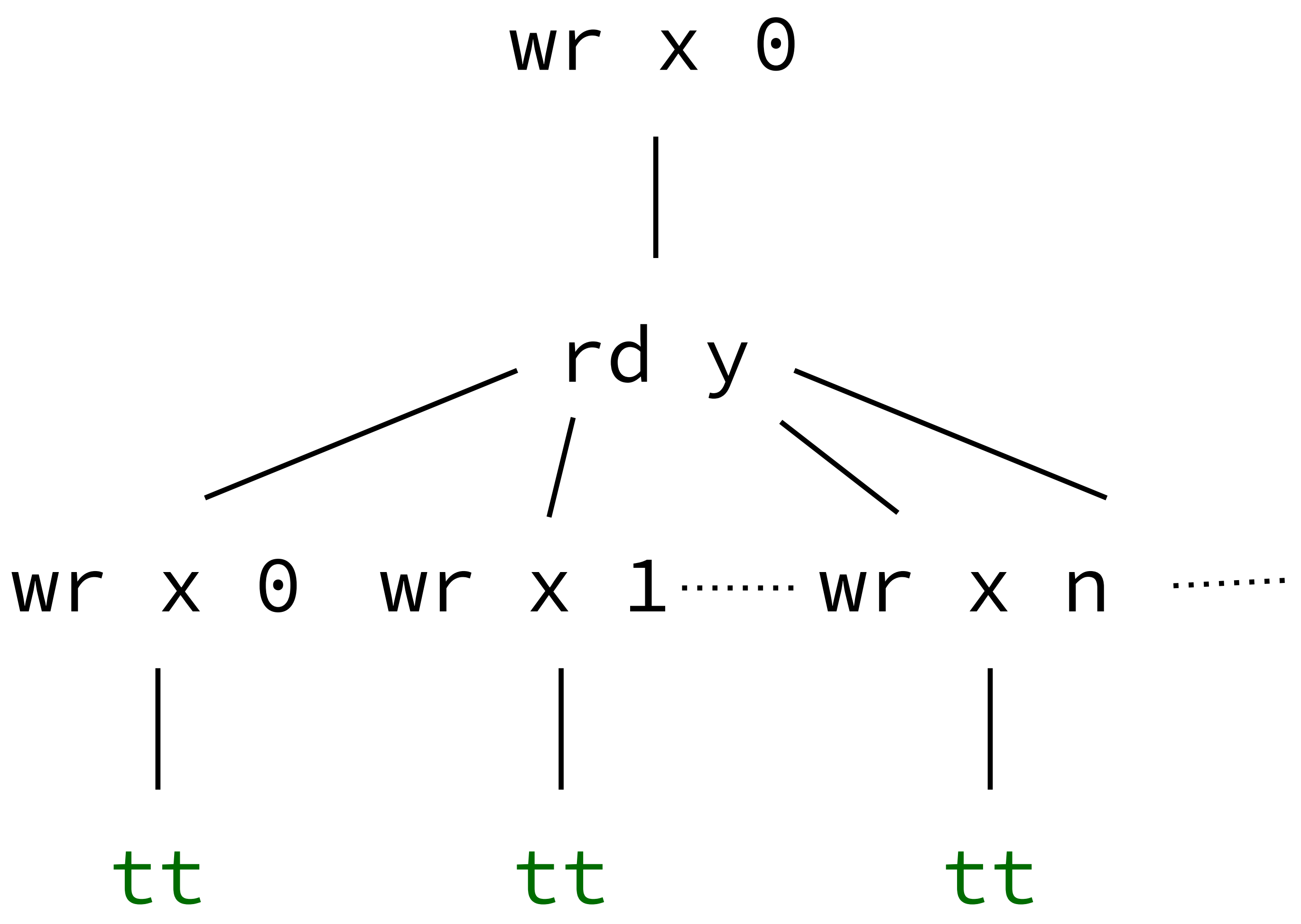


Programs as Stateful Trees

Imp programs are stateful computations

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

$q \triangleq x := y$



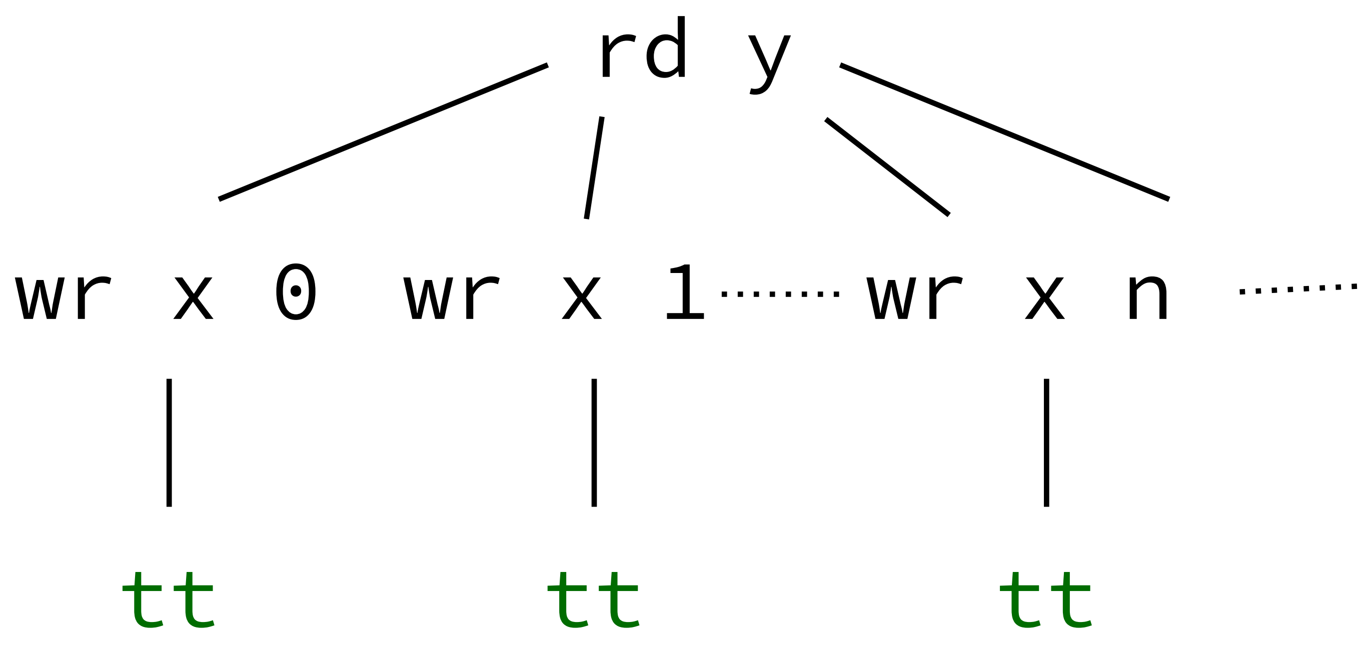
$m \mapsto$

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

$\approx$

$m \mapsto$

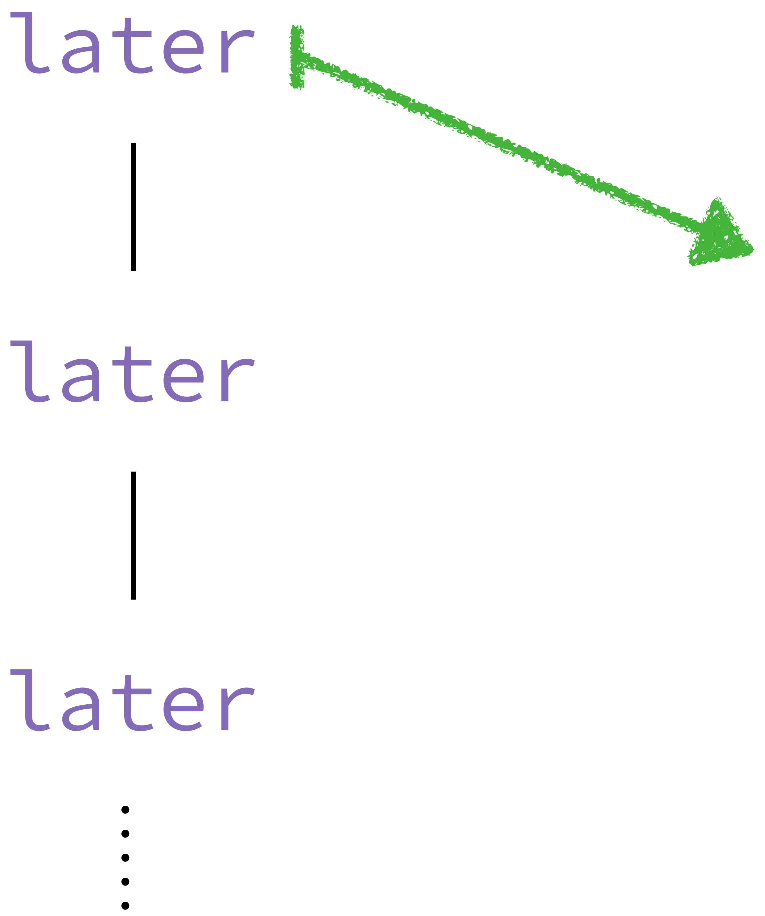
$m\{x \leftarrow m(y)\}$



ITree Second Notion: Capretta’s Delay Monad

Should recursion be an operation? We hardcode a model for it

$r \triangleq \mathit{while\ true\ do}\ \bullet$



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

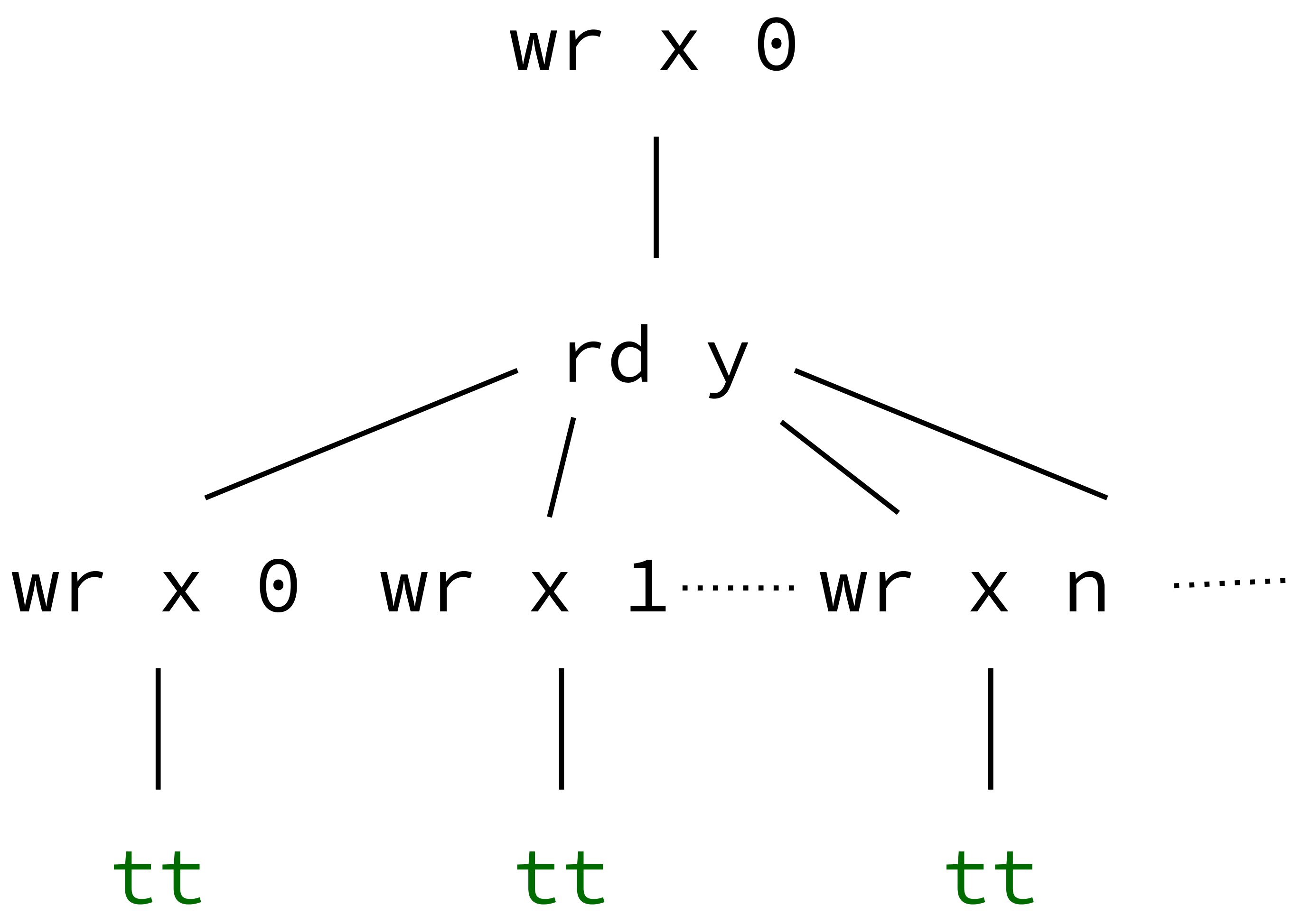
We move onto a coinductive datatype, r is an infinite tree

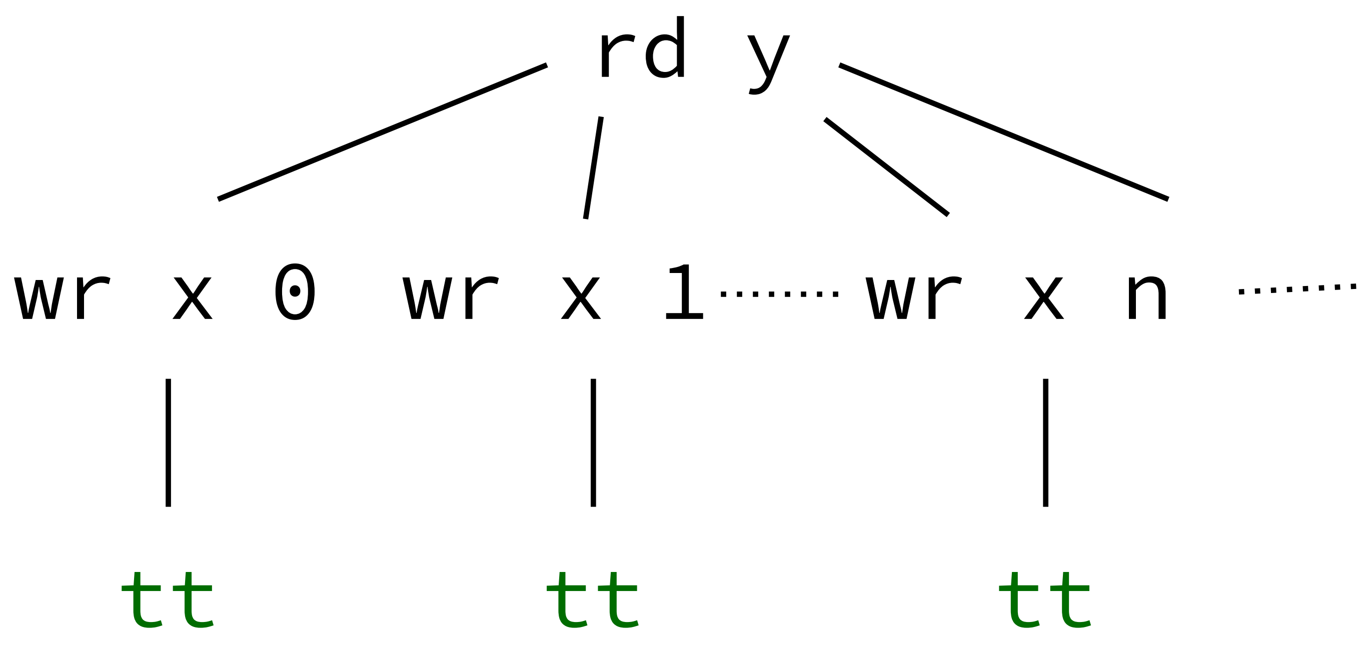
Programs as Stateful Potentially Infinite Trees

Imp programs are stateful delayed computations

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

$p_3 \triangleq x := y$

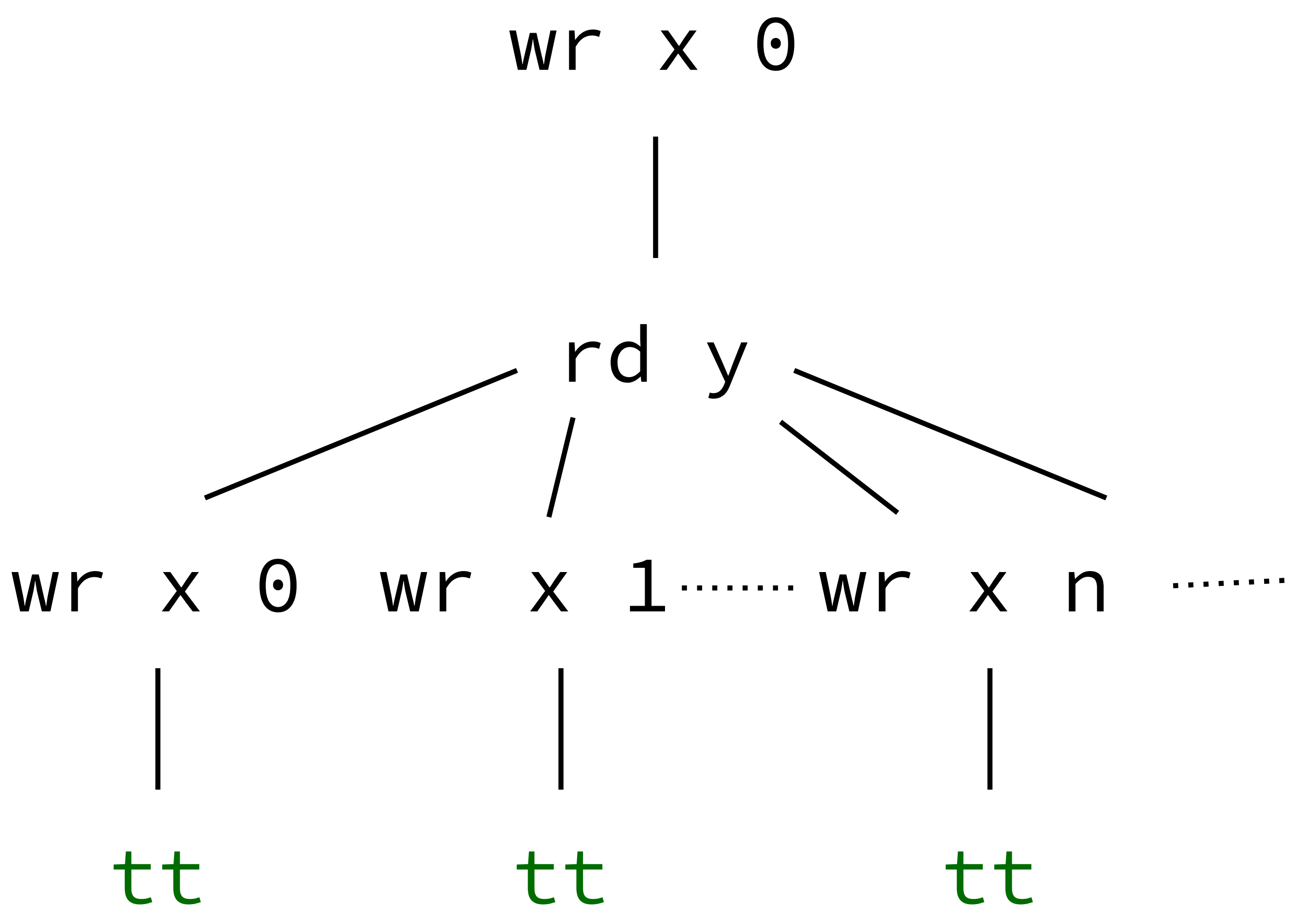




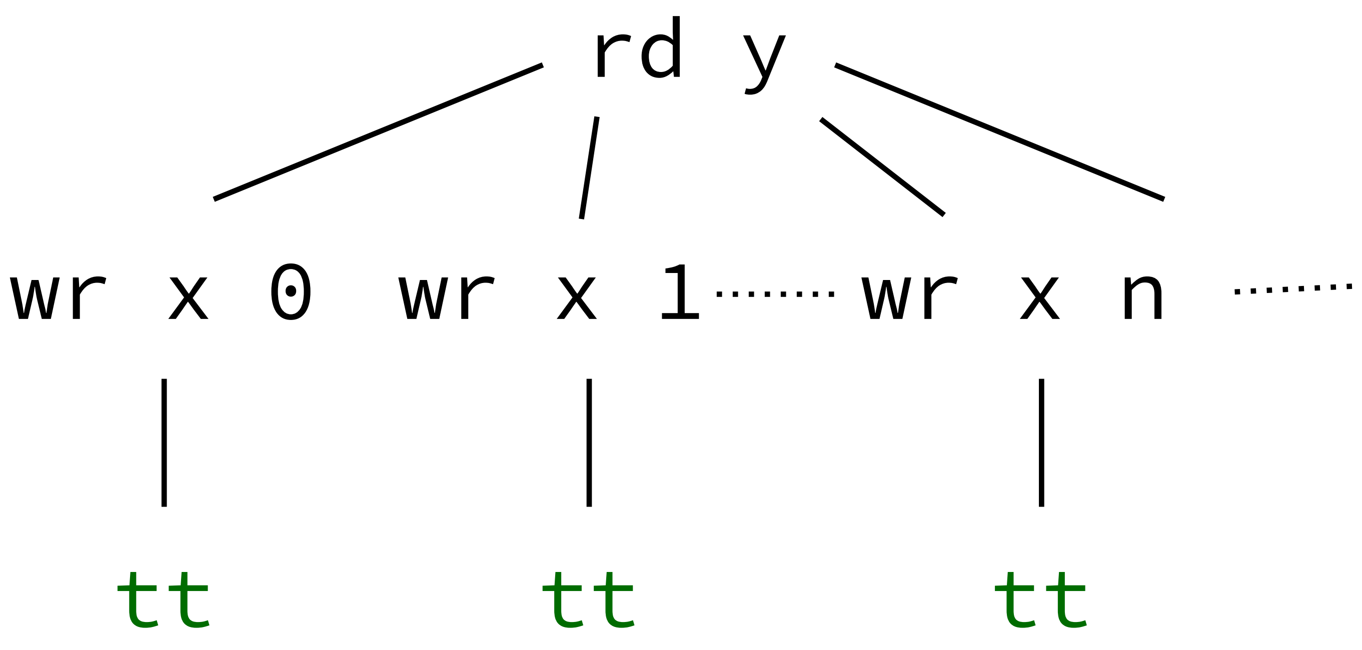
Programs as Stateful Potentially Infinite Trees

Imp programs are stateful delayed computations

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



$p_3 \triangleq x := y$

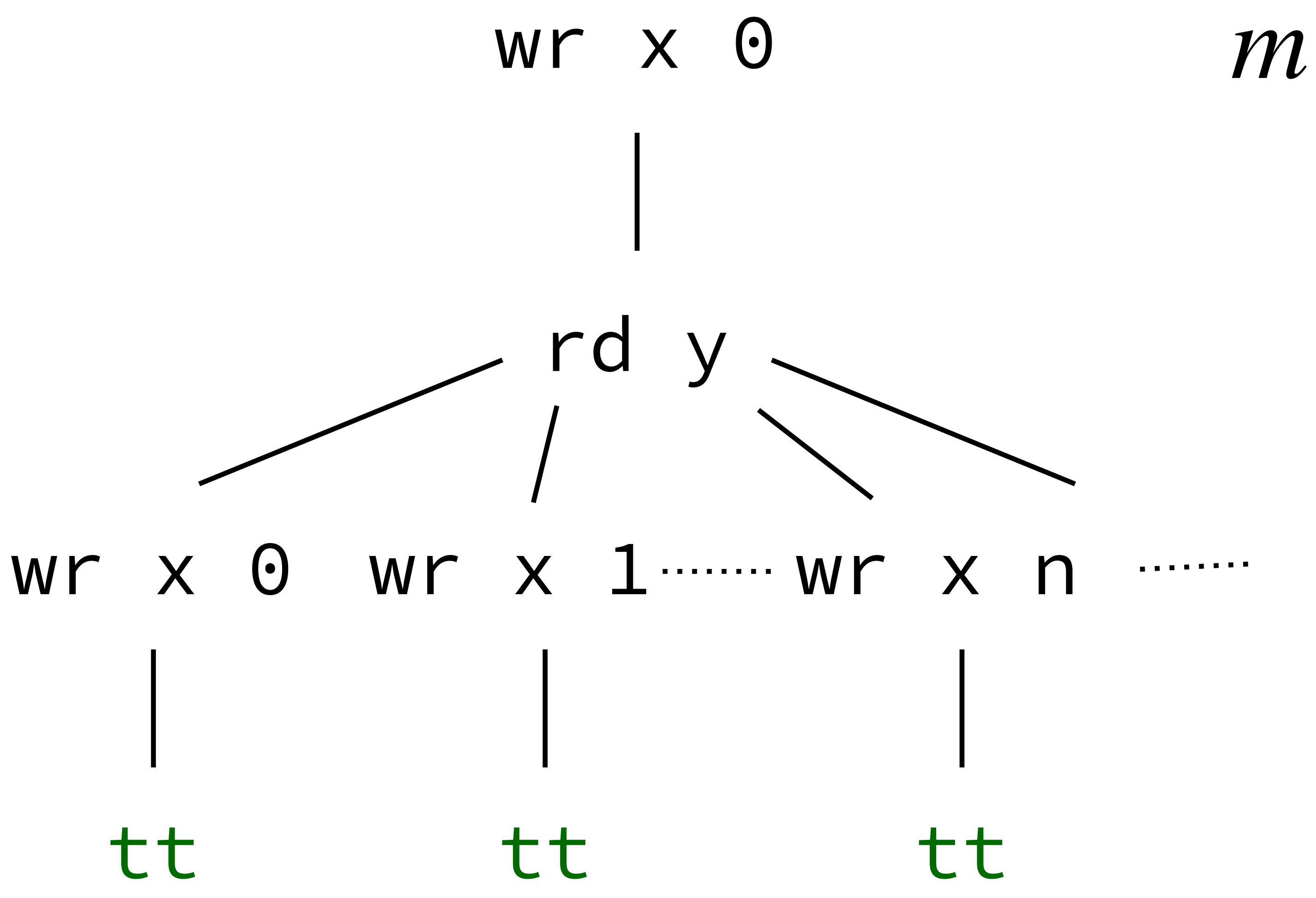


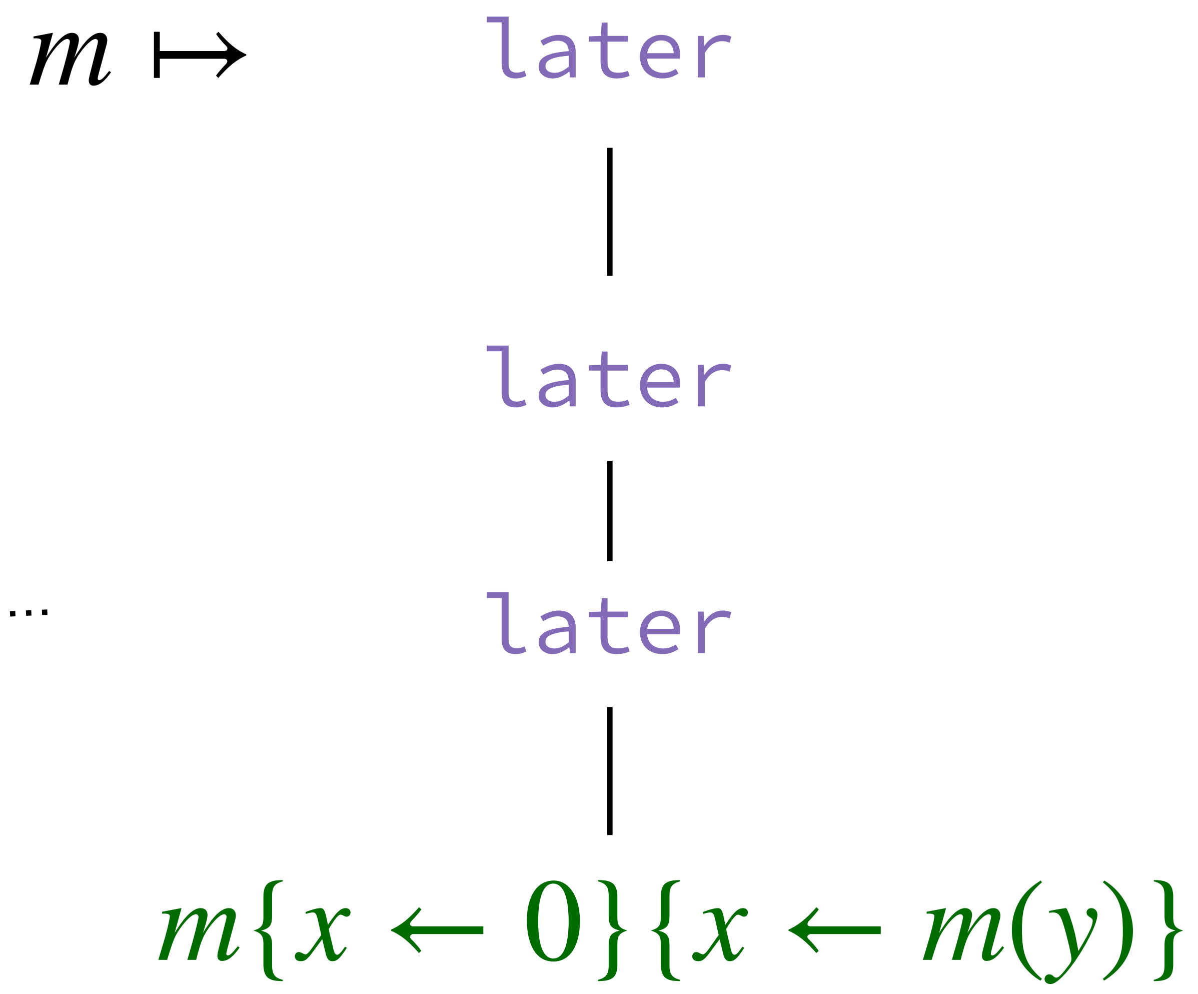
Programs as Stateful Potentially Infinite Trees

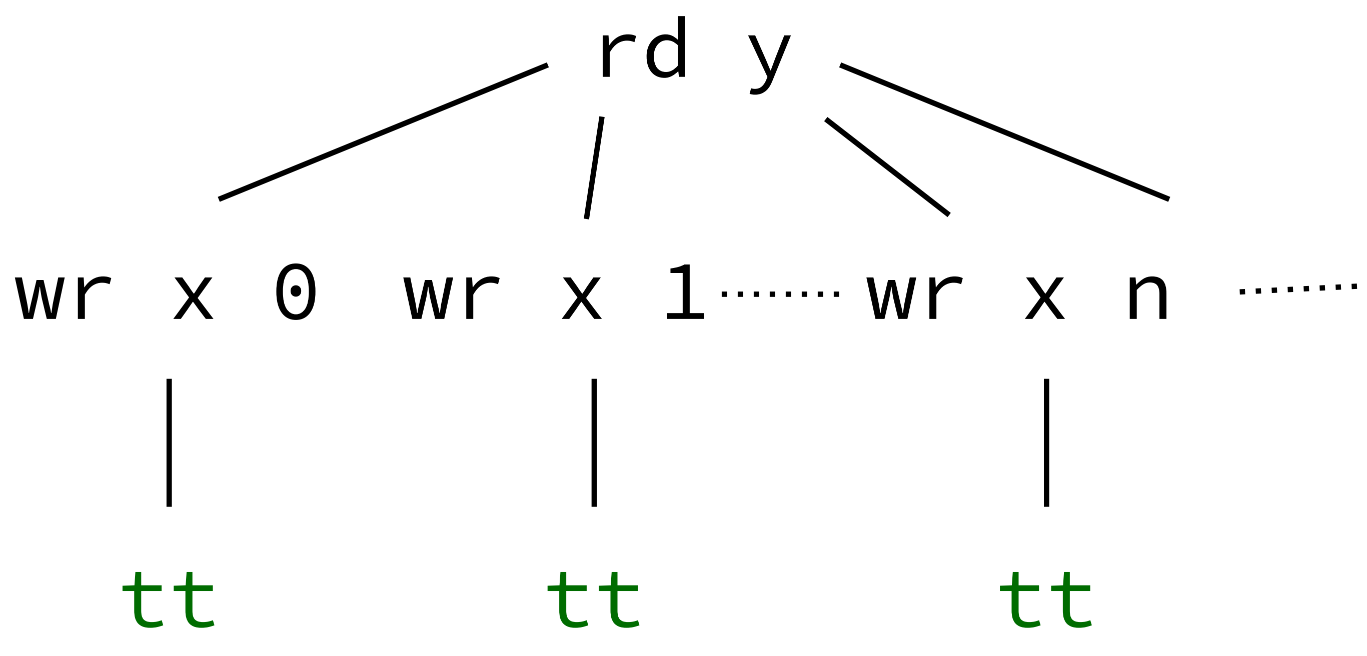
Imp programs are stateful delayed computations

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

$p_3 \triangleq x := y$





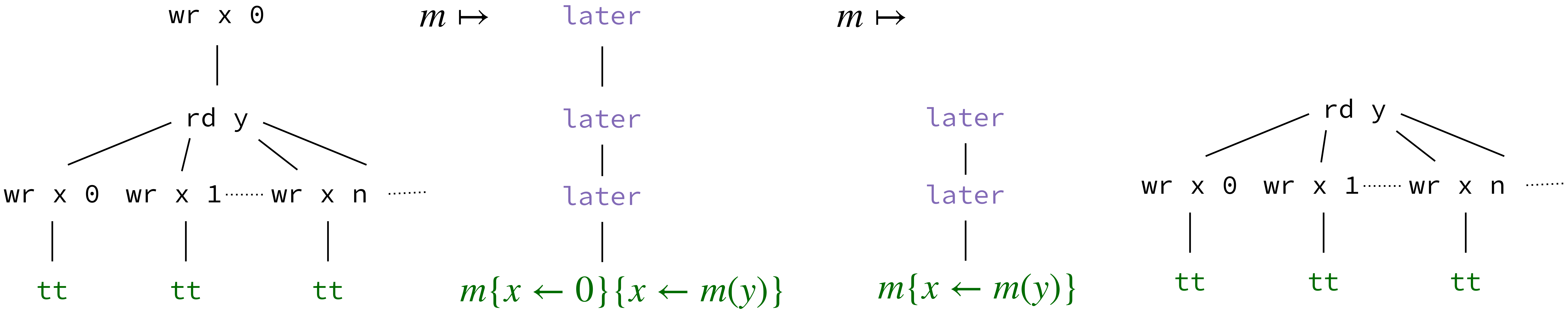


Programs as Stateful Potentially Infinite Trees

Imp programs are stateful delayed computations

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

$p_3 \triangleq x := y$

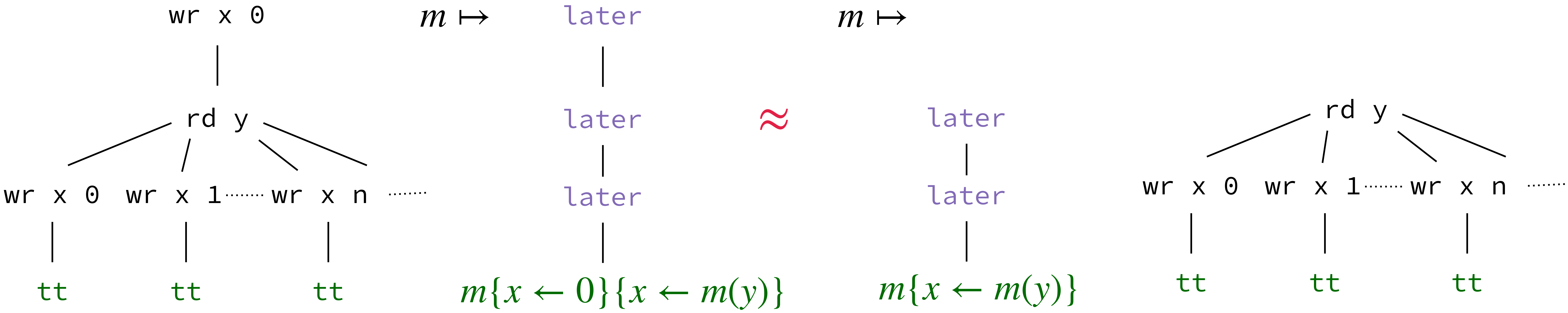


Programs as Stateful Potentially Infinite Trees

Imp programs are stateful delayed computations

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

$p_3 \triangleq x := y$



wr x 0

rd y

wr x 0 wr x 1 ……… wr x n ………

tt tt tt

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

$\approx$

$m \mapsto$ later later $m\{x \leftarrow m(y)\}$

rd y

wr x 0 wr x 1 ……… wr x n ………

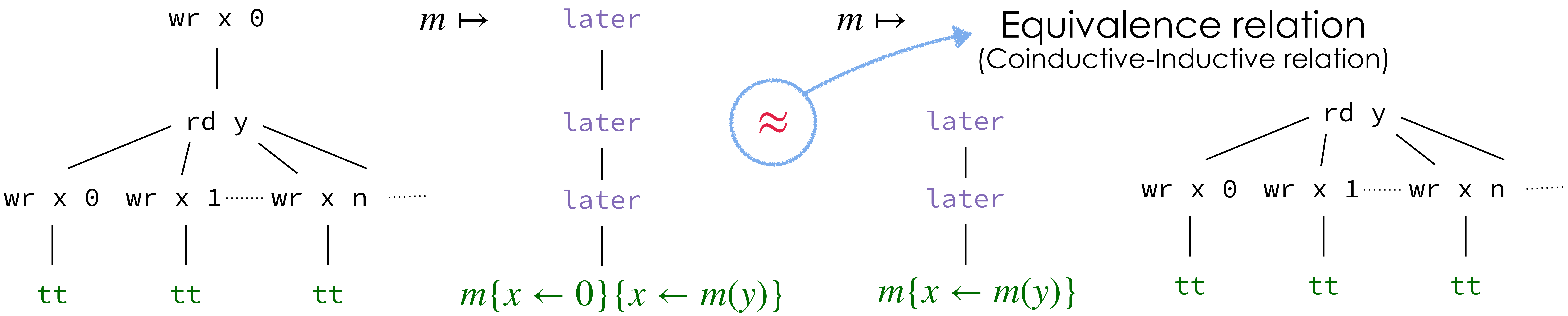
tt tt tt

Programs as Stateful Potentially Infinite Trees

Imp programs are stateful delayed computations

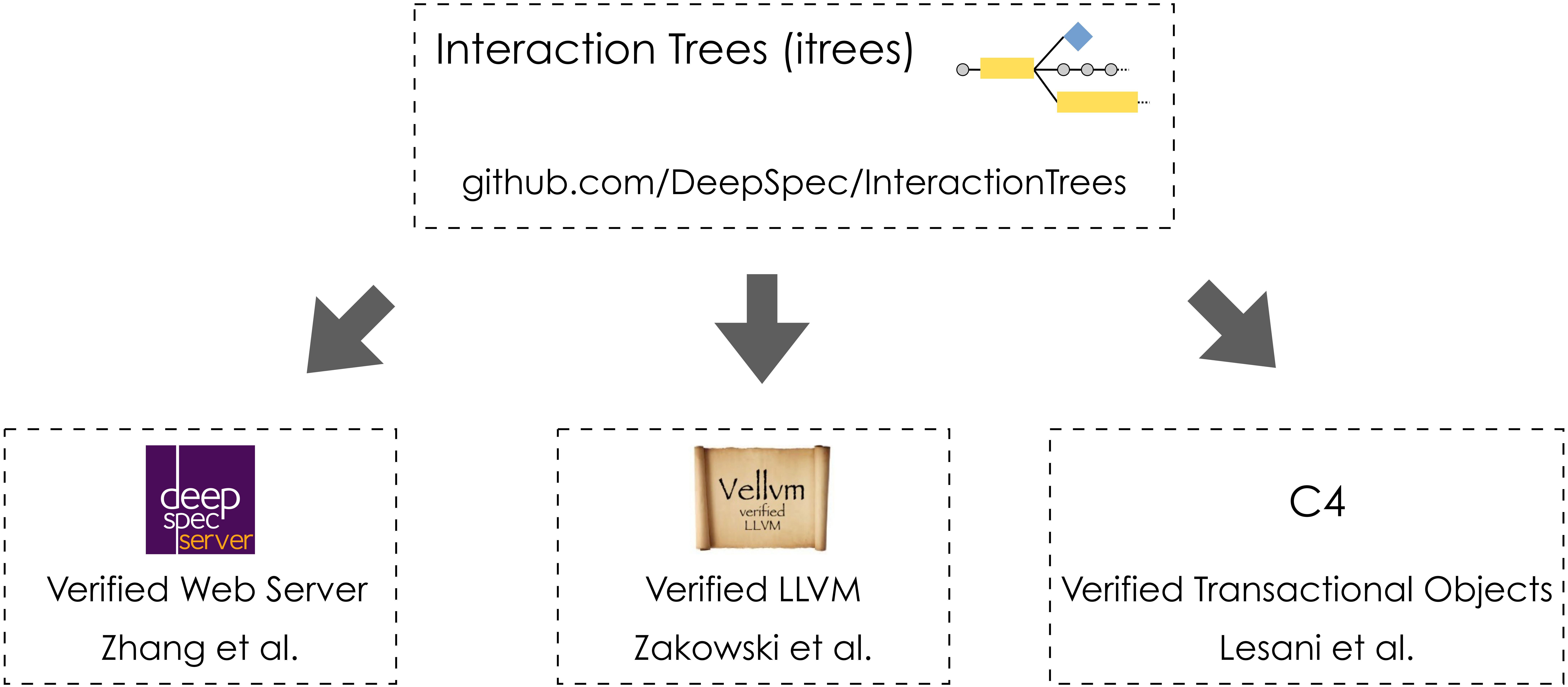
$p_2 \triangleq x := 0; x := y$

$p_3 \triangleq x := y$



Equivalence relation
(Coinductive-Inductive relation)

A Reusable Library, at Scale



Interaction Trees (itrees)

github.com/DeepSpec/InteractionTrees

deep spec server

Verified Web Server

Zhang et al.

Vellvm verified LLVM

Verified LLVM

Zakowski et al.

C4

Verified Transactional Objects

Lesani et al.

Choice Trees

or

Representing
Nondeterministic, Recursive, and Impure
Programs in Coq

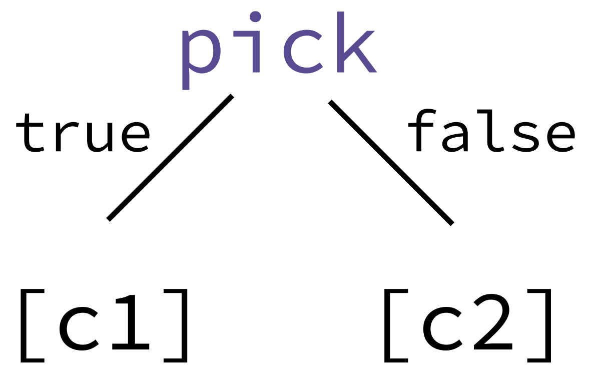
How does the story go with nondeterministic computations?

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

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



pick

true

false

[c1]

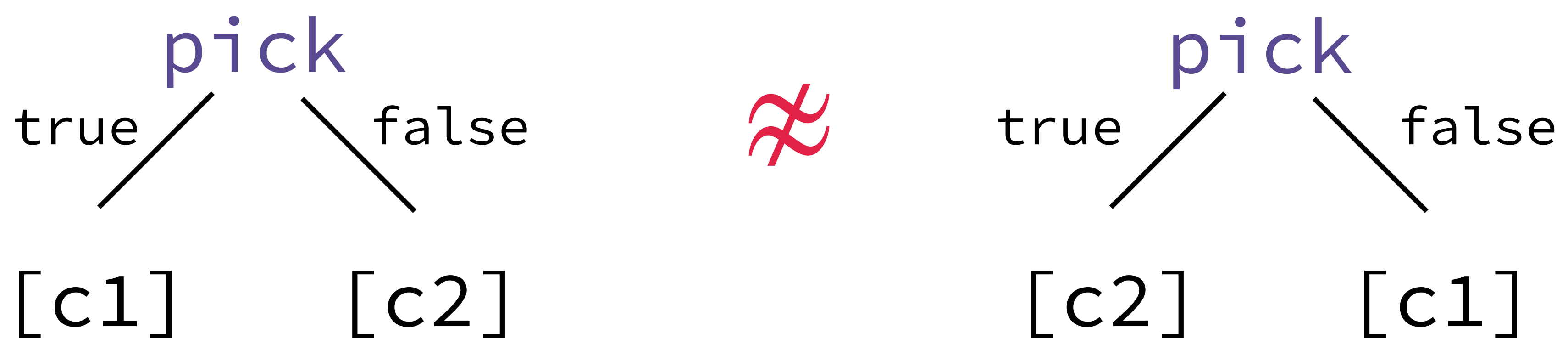
[c2]

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

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



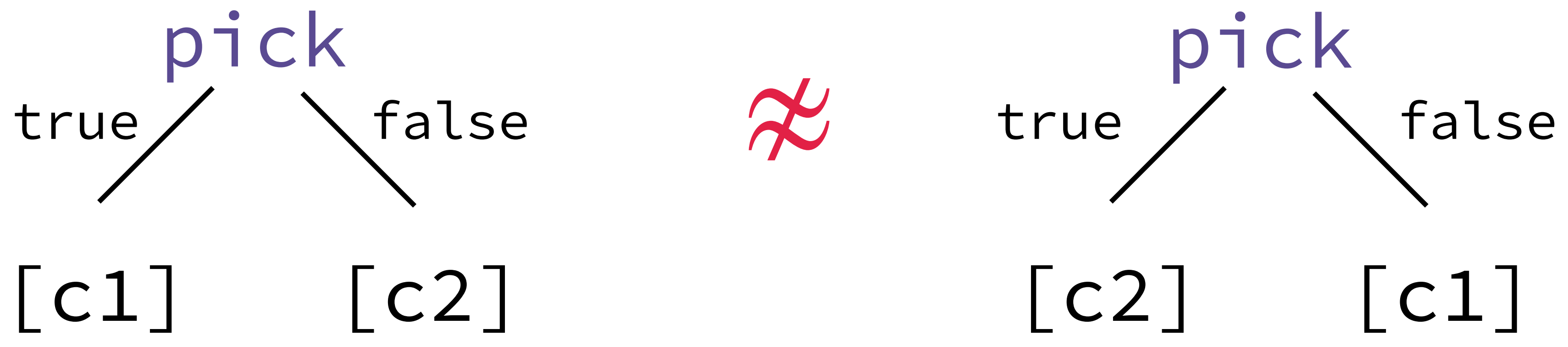
$\triangleq [br\ c_2\ or\ c_1]$

At this stage, pick is not commutative (nor idempotent, nor associative)

Nondeterministic Branching

This paper: what structure should we implement pick into?

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



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

At this stage, pick is not commutative (nor idempotent, nor associative)

Nondeterministic Branching: Which Meaning?

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

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

More specifically, we may mean one of two operational behaviours:

Nondeterministic Branching: Which Meaning?

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

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

More specifically, we may mean one of two operational behaviours:

- The system may **become** either branch

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

Nondeterministic Branching: Which Meaning?

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

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

More specifically, we may mean one of two operational behaviours:

- The system may **become** either branch $$\frac{}{br\ c_1\ or\ c_2 \to c_1}$$
- The system may **take a transition** offered by either branch $$\frac{c_1 \to c_1'}{br\ c_1\ or\ c_2 \to c_1'}$$

Nondeterministic Branching: Which Meaning?

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

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

Depending on our choice of semantics, the program p may be stuck, or not

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

Let’s Take the Perspective of an LTS

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

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

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

Let’s Take the Perspective of an LTS

Case 1:

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

$p \to stuck$ is possible

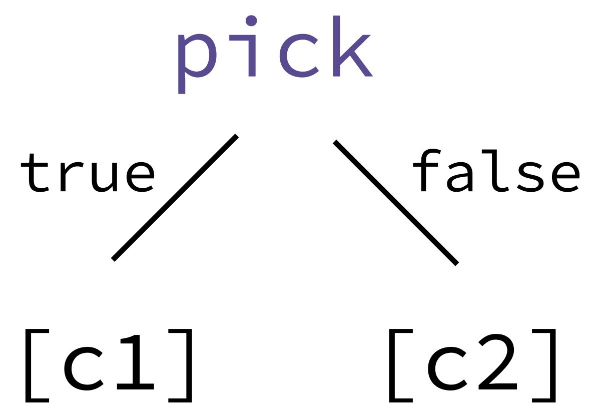
Case 2:

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

$p \to stuck$ is not possible

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

Let’s Take the Perspective of an LTS



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

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

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* is possible

Case 1:

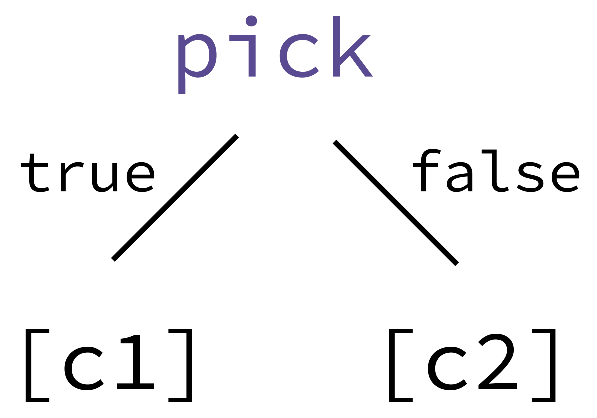
$$\frac{}{br\ c_1\ or\ c_2 \rightarrow c_1}$$

$p \rightarrow$ *stuck* is possible

Case 2:

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

$p \rightarrow$ *stuck* is not possible



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

Let’s Take the Perspective of an LTS

Case 0 (itree):

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

$p \xrightarrow{true} stuck$ is possible

Case 1:

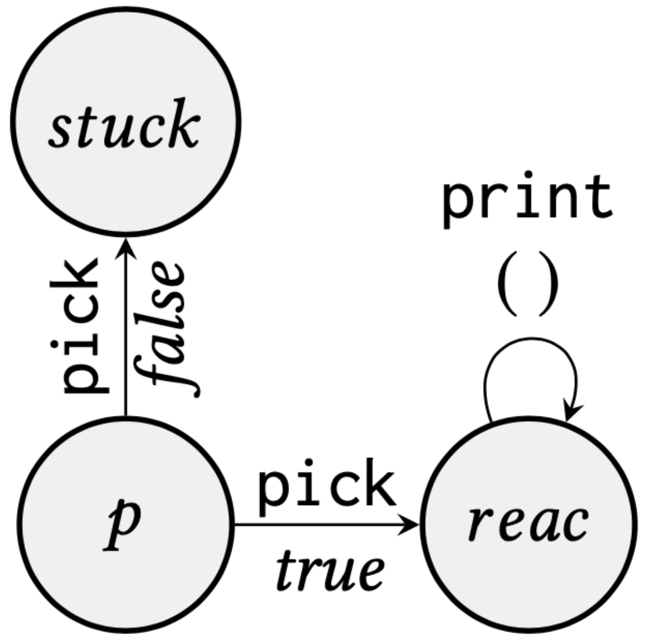
$$\frac{}{br\ c_1\ or\ c_2 \rightarrow c_1}$$

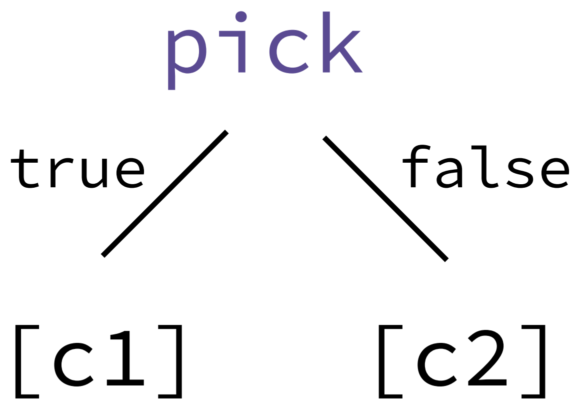
$p \rightarrow stuck$ is possible

Case 2:

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

$p \rightarrow stuck$ is 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):

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

$p \xrightarrow{true} stuck$ is possible

Case 1:

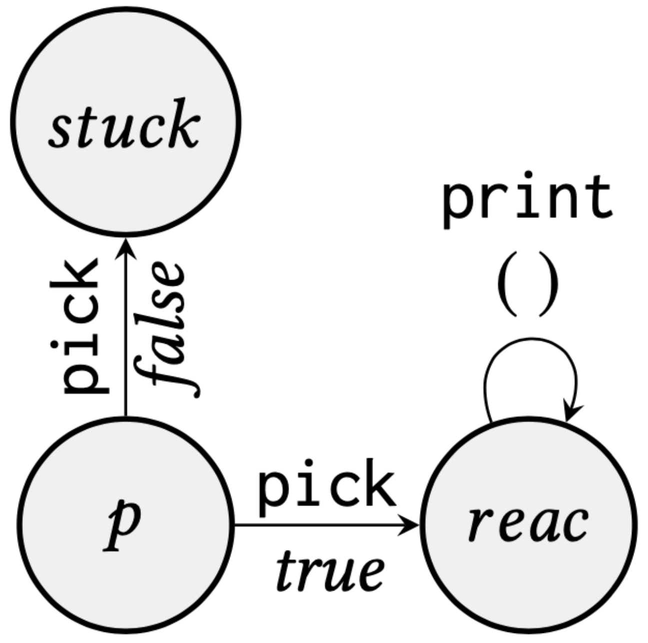
$$\frac{}{br\ c_1\ or\ c_2 \rightarrow c_1}$$

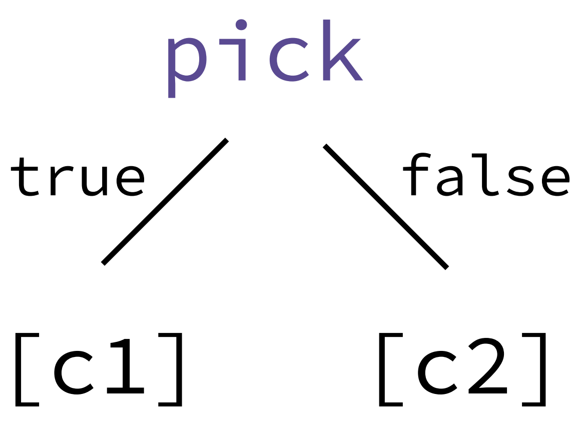
$p \rightarrow stuck$ is possible

Case 2:

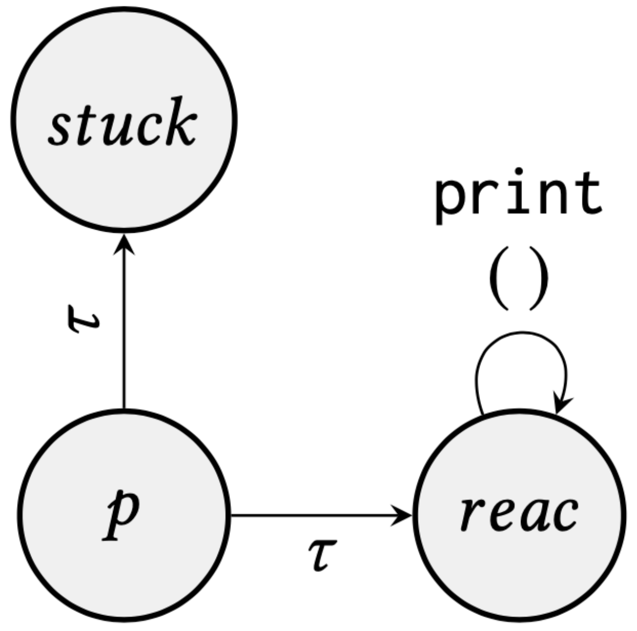
$$\frac{c_1 \rightarrow c_1'}{br\ c_1\ or\ c_2 \rightarrow c_1'}$$

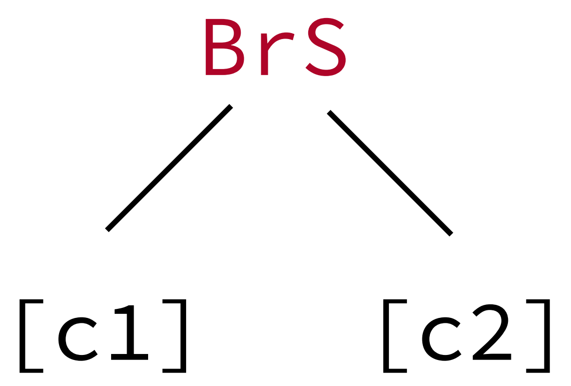
$p \rightarrow stuck$ is not possible





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





Stepping branch,
we observe that a branch
has been taken

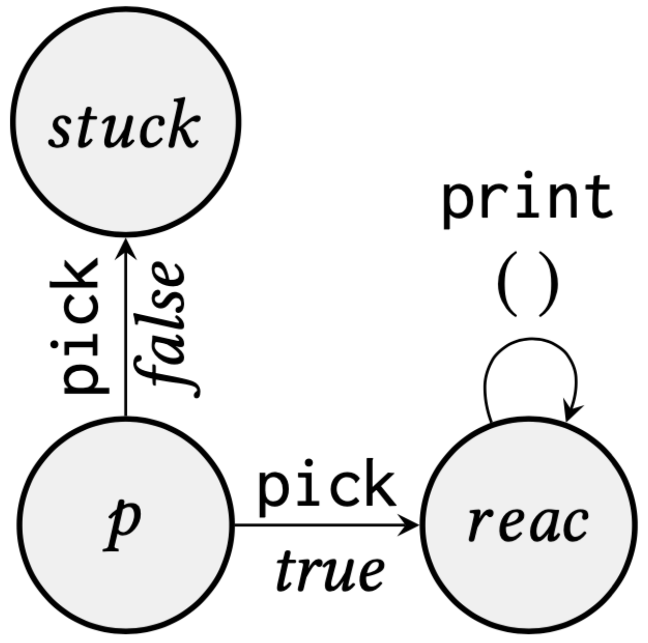
$p \triangleq br\ (while\ true\ do\ print)\ or\ stuck$

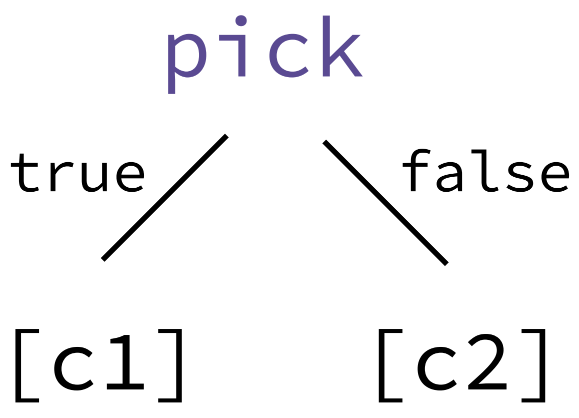
Let's Take the Perspective of an LTS

Case 0 (itree):

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

$p \xrightarrow{true} stuck$ is possible



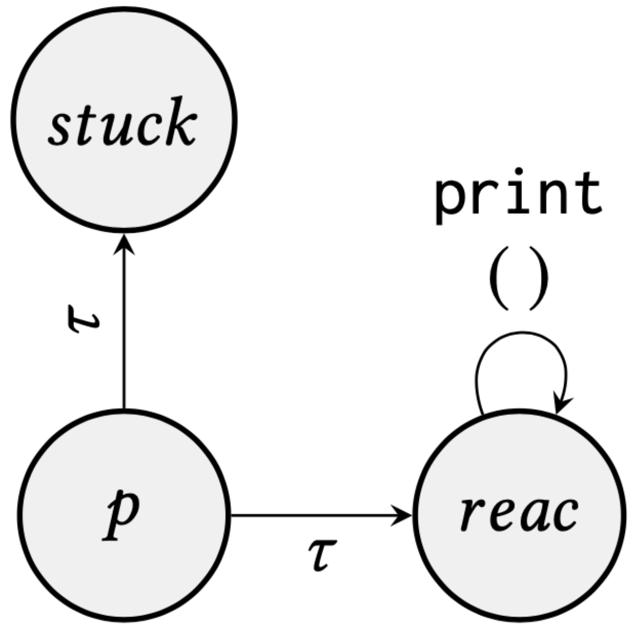


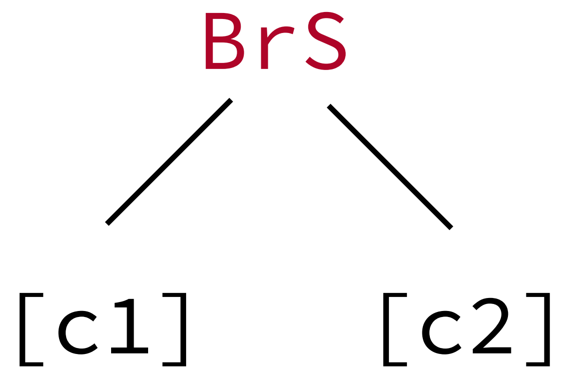
External event,
we observe which event happened,
what branch we took

Case 1:

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

$p \rightarrow stuck$ is possible



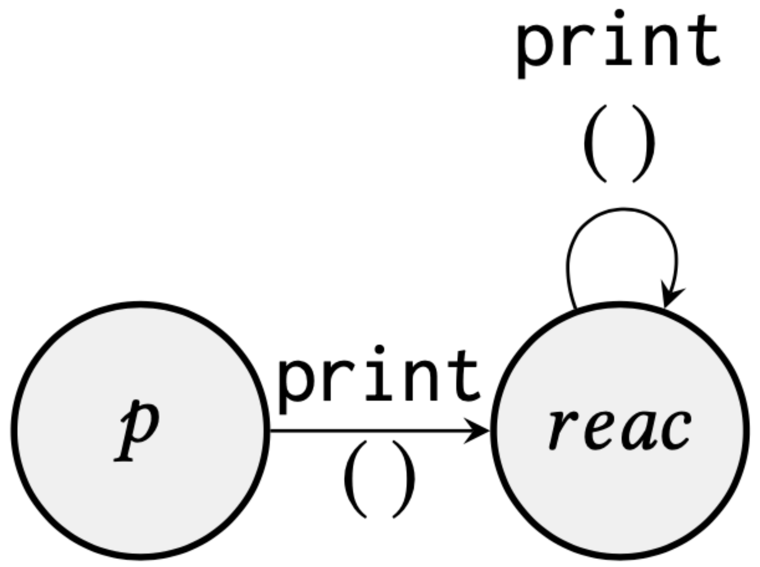


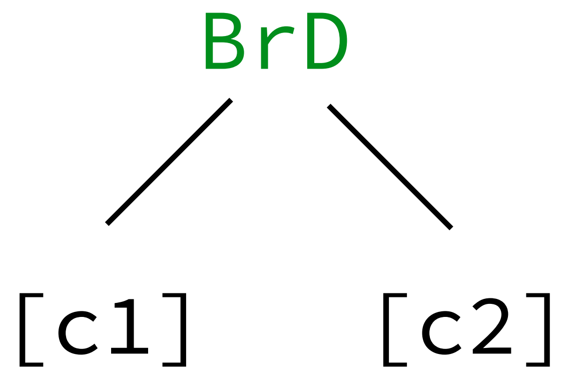
Stepping branch,
we observe that a branch
has been taken

Case 2:

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

$p \rightarrow stuck$ is not possible

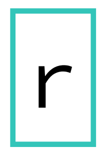




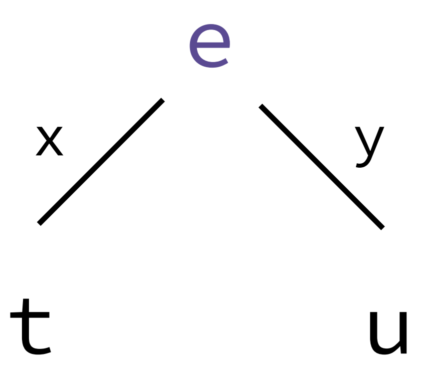
Delayed branch,
there's a branch,
but we don't observe it

Choice Trees

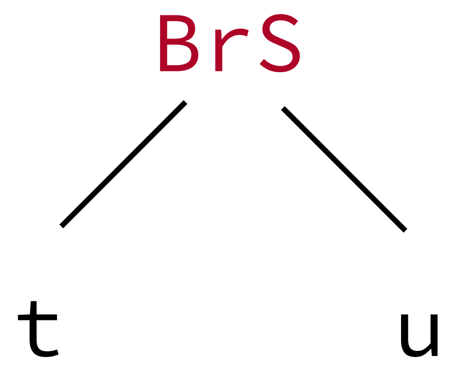
A *ctree E R* models a computation as a potentially infinite tree made of:



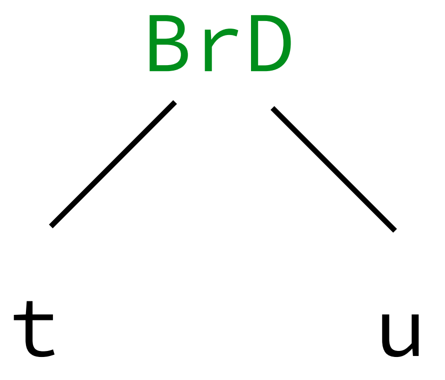
Leaves,
pure computations
(of type R)



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)
```

LTSs Underlying CTrees

Question: How to build the LTS underlying a ctree?

$$\mathit{label} \;::=$$

LTSs Underlying CTrees

Question: How to build the LTS underlying a ctree?

$$label \ ::= \ val\ x$$

$$\boxed{\text{r}} \xrightarrow{val\ r} \emptyset$$

Leaves,
pure computations
(of type R)

LTSs Underlying CTrees

Question: How to build the LTS underlying a ctree?

(Propositional) relation

$$label \ ::= \ val \ x$$

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

Leaves,
pure computations
(of type R)

LTSs Underlying CTrees

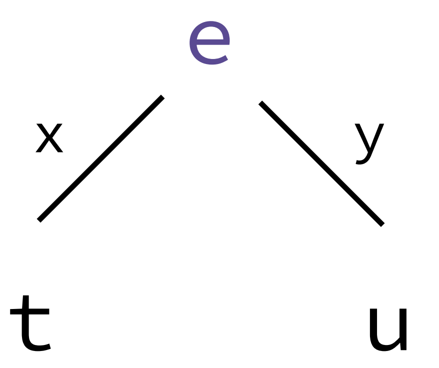
Question: How to build the LTS underlying a ctree?

(Propositional) relation

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

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

Leaves, pure computations (of type R)



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

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

LTSs Underlying CTrees

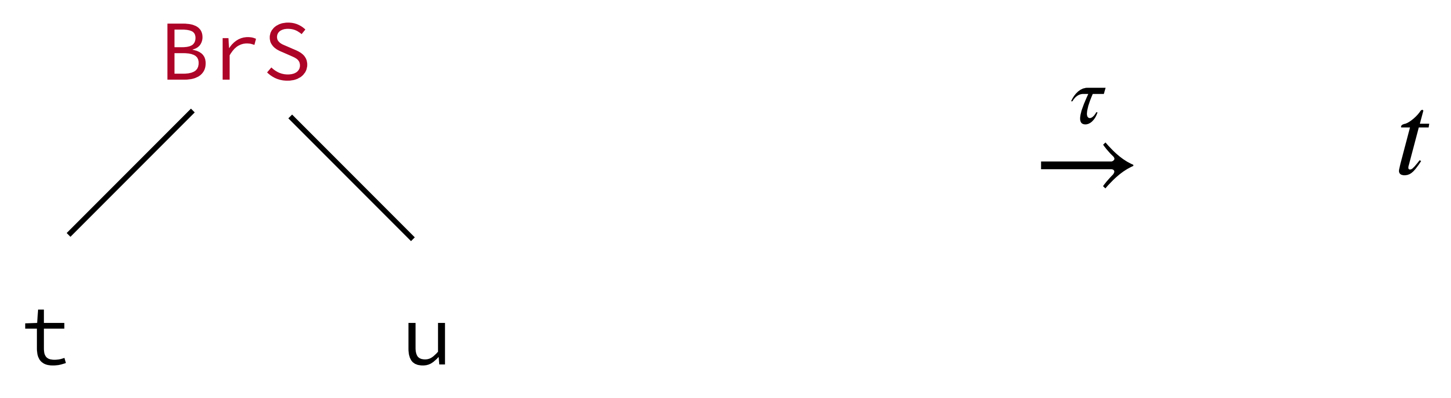
Question: How to build the LTS underlying a ctree?

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

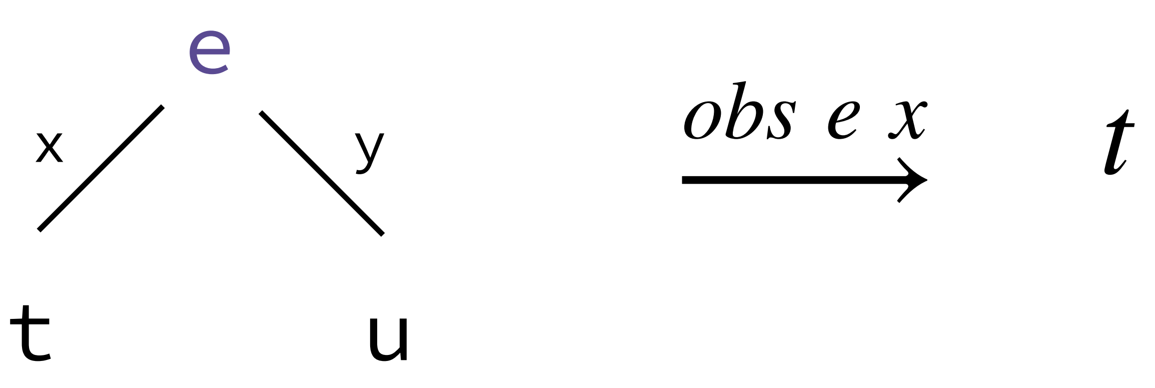
(Propositional) relation

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

Leaves,
pure computations
(of type R)



Stepping branches,
an internal choice which may
be observed



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

LTSs Underlying CTrees

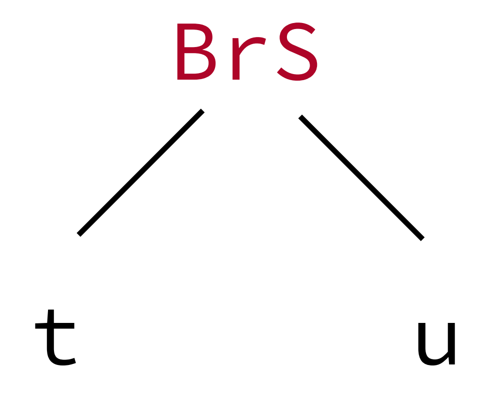
Question: How to build the LTS underlying a ctree?

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

(Propositional) relation

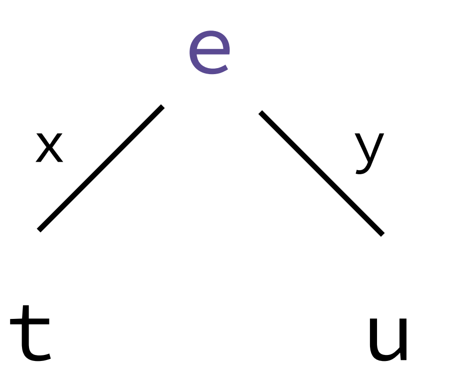
$\boxed{r} \xrightarrow{val\ r} \varnothing$

Leaves,
pure computations
(of type R)



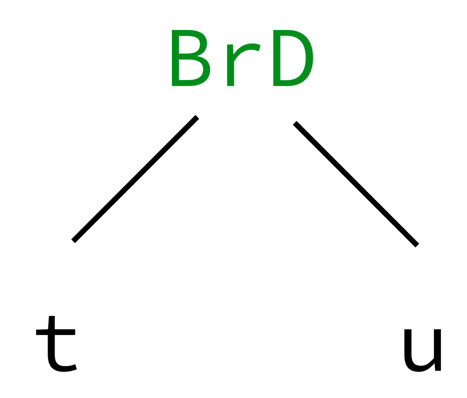
$\xrightarrow{\tau} t$

Stepping branches,
an internal choice which may
be observed



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

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



$\xrightarrow{l} t'$

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

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

Bisimulations Over CTrees

When should two ctrees be deemed equivalent?

Bisimulations Over CTrees

When should two ctrees be deemed equivalent?

When their underlying LTSs are bisimilar

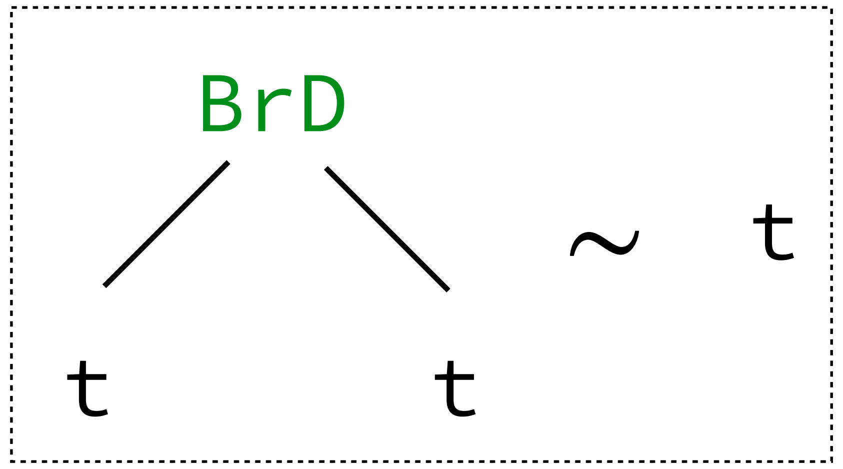
We can rely on standard notions from the process algebra tradition

[Milner 89, Sangiorgi 11, Pous 16, …]

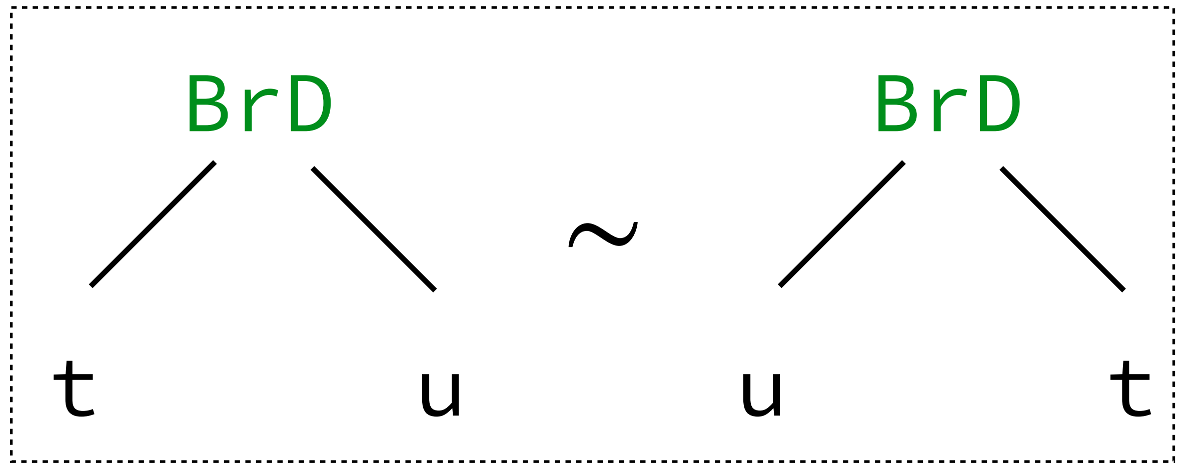
Bisimulations Over CTrees

Algebraic laws for non-determinism through **strong** bisimulation (~)

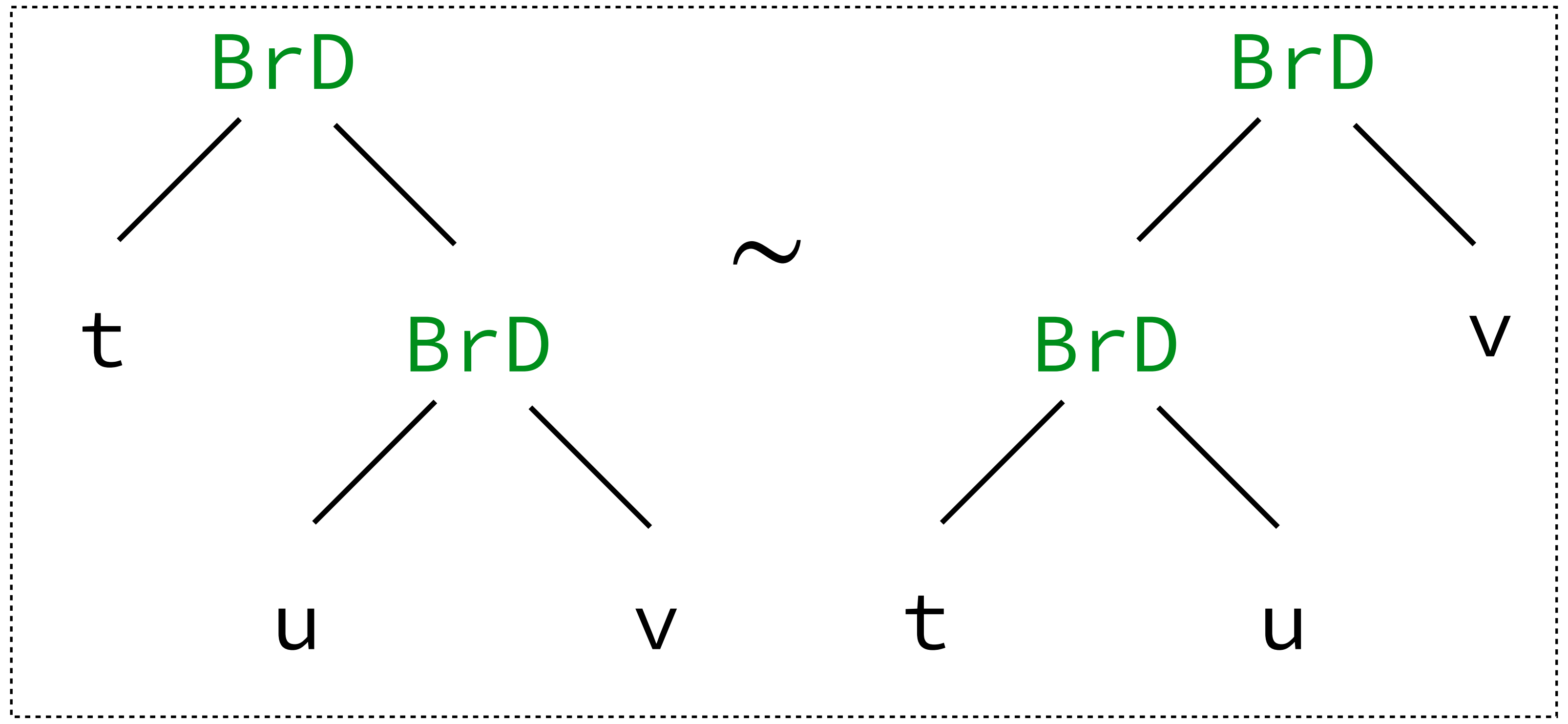
Idempotent



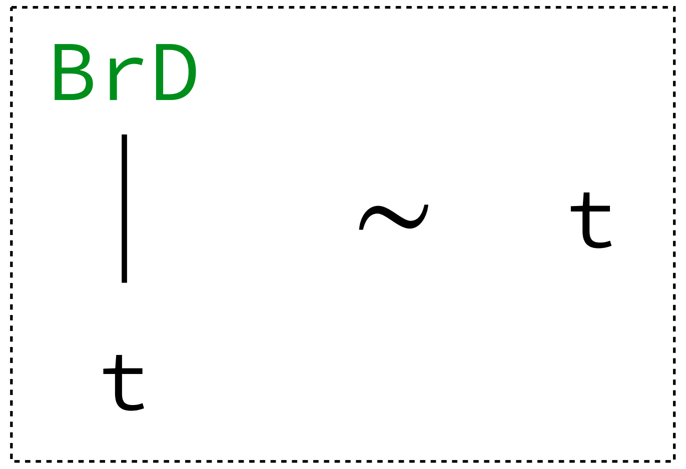
Commutative



Associative



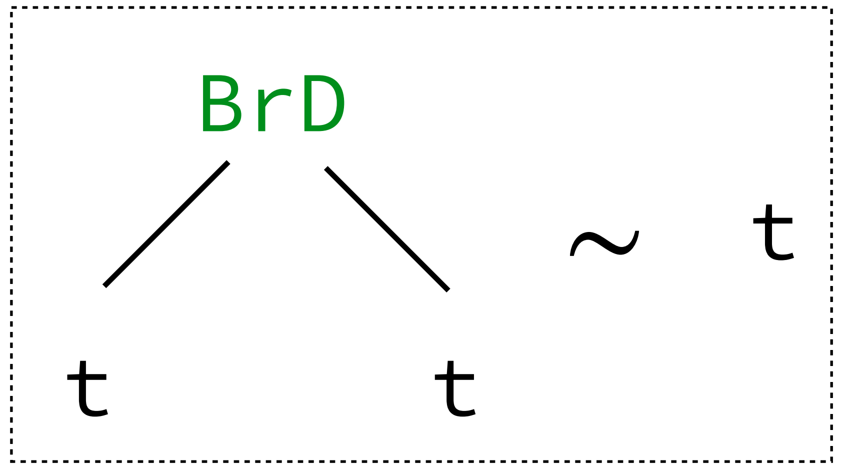
Insensitive to BrD



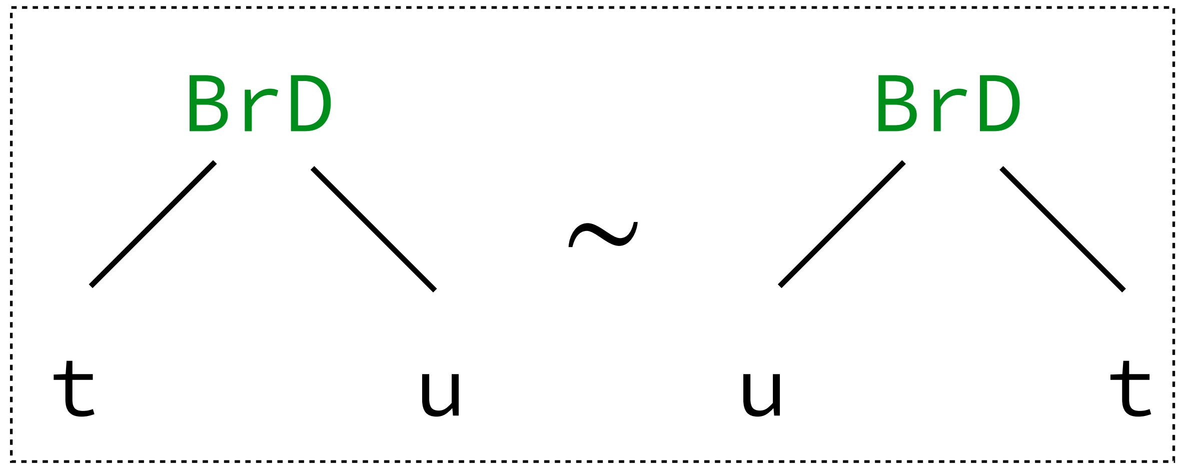
Bisimulations Over CTrees

Algebraic laws for non-determinism through **strong** bisimulation (∼)

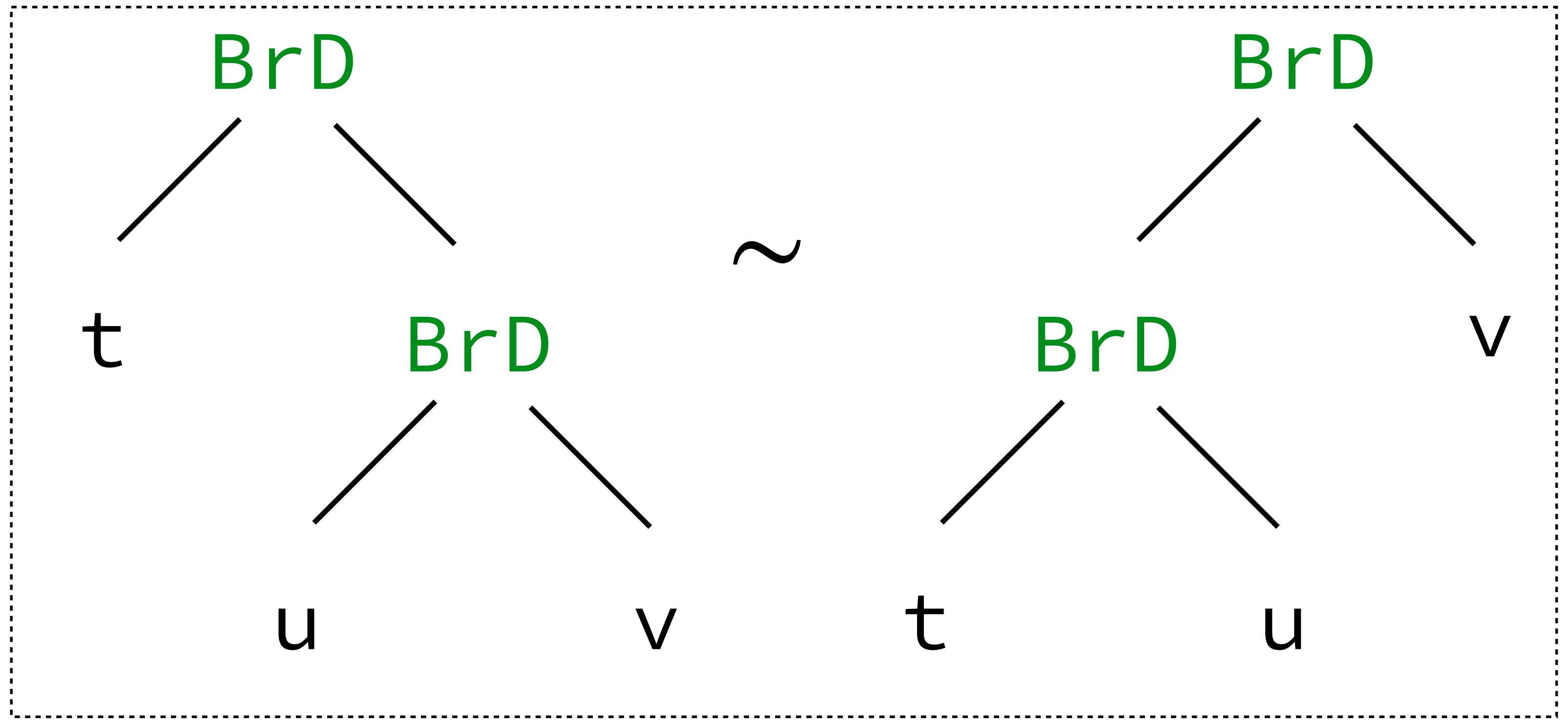
Idempotent



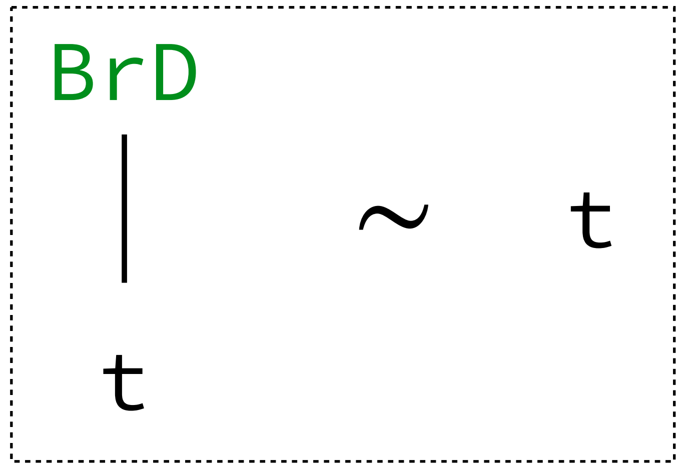
Commutative



Associative

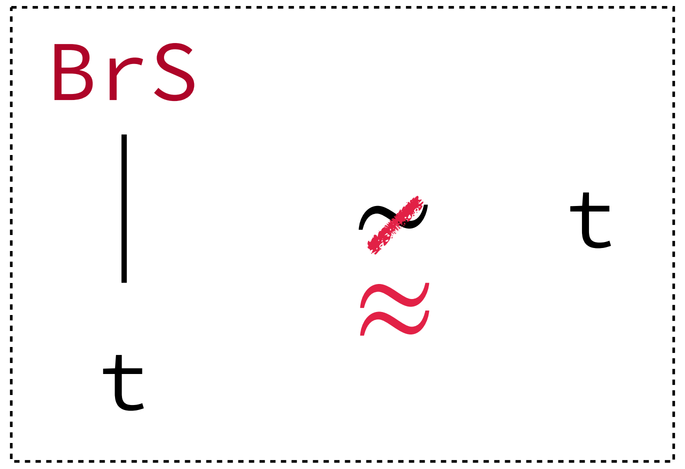


Insensitive to BrD



Insensitivity to BrS through **weak** bisimulation (≈)

Insensitive to BrS



CTrees and Interpretation

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

```
h(pick) ≜ BrD 2

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

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

CTrees and Interpretation

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

```
h(pick) ≜ BrD 2

interp h : itree (Pick + 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)

CTrees and Interpretation

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

```
h(pick) ≜ BrD 2

interp h : itree (Pick + 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)

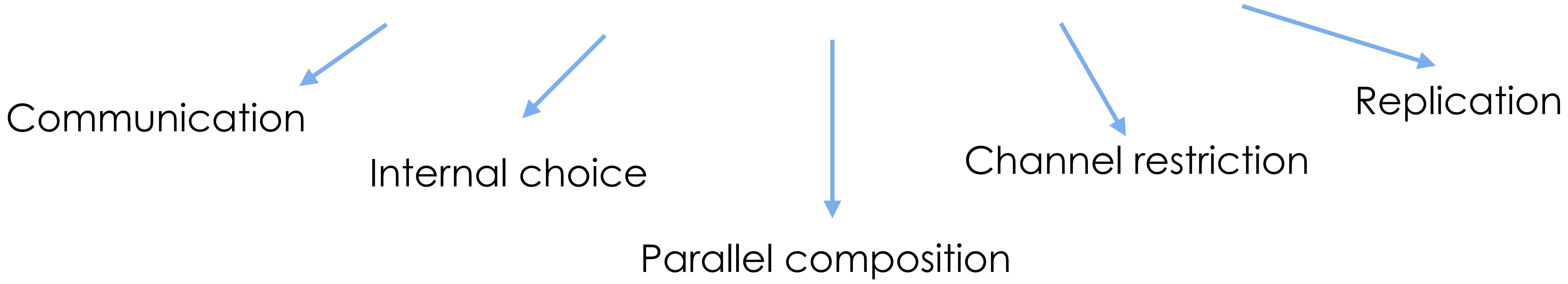
→ Branching nodes can be « interpreted » as well

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

Choice Trees: Case Studies

Calculus of Communicating Systems [Milner, 1980]

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



Communication

Internal choice

Parallel composition

Channel restriction

Replication

Goal: compute a model of ccs using ctrees

→ We establish ccs's traditional equational theory w.r.t. $\sim$ on our model

→ We prove an adequacy result against ccs's operational semantics

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

Cooperative scheduling

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

- 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

$$\mathcal{S}[\![\texttt{fork}\ c1\ (\texttt{fork}\ c2\ \texttt{skip})]\!] \approx \mathcal{S}[\![\texttt{fork}\ c2\ (\texttt{fork}\ c1\ \texttt{skip})]\!]$$

Conclusion

A New Tool in the Interaction Trees Environment

Modelling non-determinism and concurrency as monadic interpreters

→ Two new kind 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

→ Encouraging case studies

Implemented as a Coq library: https://github.com/vellvm/ctrees/tree/popl23

Relies heavily on Pous's coinduction library (coq-coinduction on Opam)

Backup

Nondeterministic branching

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

An idea: *sets* of trees? $\mathcal{I}([br\ c_1\ or\ c_2]) \triangleq [c_1] \cup [c_2]$ (In Coq: `itree E X -> Prop`)

- ❌ `PropT M X ≜ M X -> Prop`
  is not a monad transformer (bind fails to associate to the left)
- ❌ Equivalence is a notion of bijection
  ⇝ existential quantification of a coinductive object
- ❌ Imposes trace equivalence onto us
- ❌ We do not want to go into `Prop`!

Nondeterministic branching

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

An idea: *sets* of trees? $\mathscr{I}([br\ c_1\ or\ c_2]) \triangleq [c_1] \cup [c_2]$ (In Coq: `itree E X -> Prop`)

- ❌ `PropT M X ≜ M X -> Prop`

  is not a monad transformer (bind fails to associate to the left)

- ❌ Equivalence is a notion of bijection

  $\rightsquigarrow$ existential quantification of a coinductive object

- ❌ Imposes trace equivalence onto us

- ❌ We do not want to go into `Prop`!

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

Calculus of Communicating Systems [Milner, 1980]

head p: computes all first reachable actions in a ctree

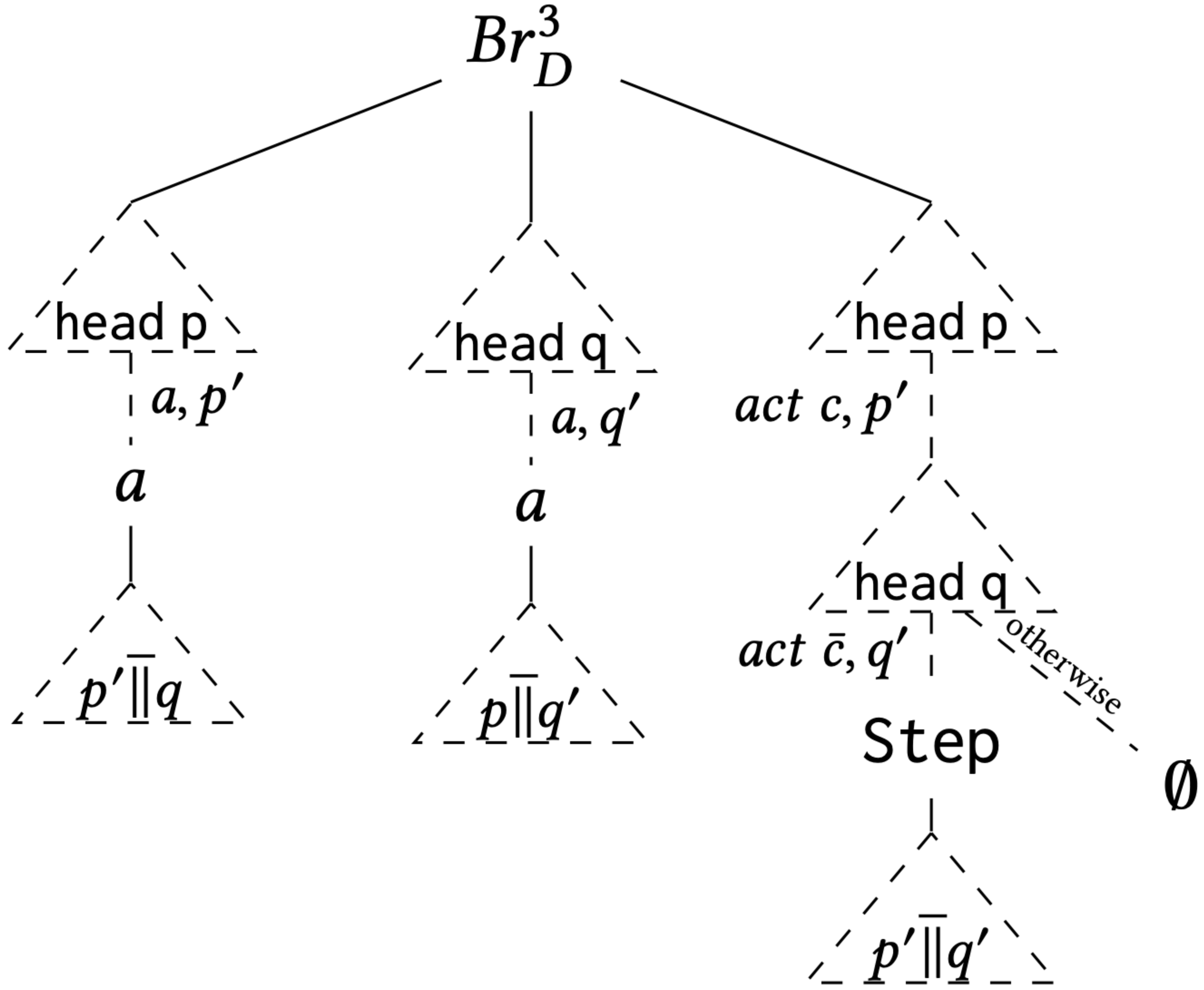


Fig. 19. Depiction of the tree resulting from $p \overline{\|} q$