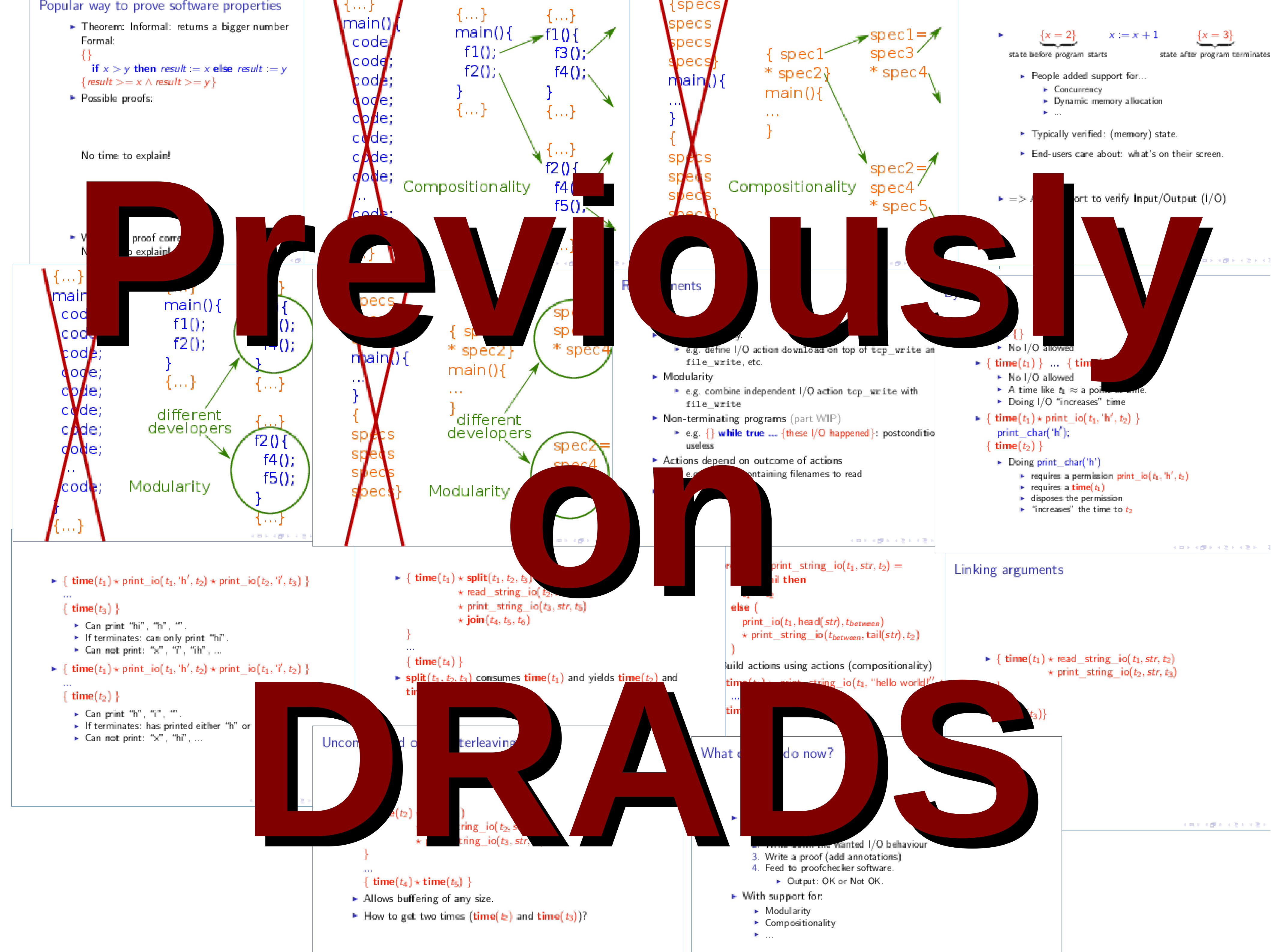
I/O verification: formalization and soundness proof

Willem Penninckx
Bart Jacobs
Frank Piessens

imec-DistriNet, KU Leuven

DRADS 2017

Previously on DRADS



Popular way to prove software properties

- Theorem: Informal: returns a bigger number
  Formal:
  $\{\}$
  if $x > y$ **then** $result := x$ **else** $result := y$
  $\{result >= x \wedge result >= y\}$
- Possible proofs:

No time to explain!

- W... proof corr...
  N... explain!

```
{...}
main(){
  code;
  code;
  code;
  code;
  code;
  code;
  code;
  code;
  ...
  code;
}
{...}
```

Compositionality

```
{...}
main(){
  f1();
  f2();
}
{...}

{...}
f1(){
  f3();
  f4();
}
{...}

{...}
f2(){
  f4();
  f5();
}
{...}
```

```
{specs
specs
specs
specs}
main(){
...
}
{
specs
specs
specs
specs}
```

Compositionality

```
{ spec1
* spec2}
main(){
...
}

spec1=
spec3
* spec4

spec2=
spec4
* spec5
```

- $\{x = 2\}$ state before program starts $x := x + 1$ $\{x = 3\}$ state after program terminates
  - People added support for...
    - Concurrency
    - Dynamic memory allocation
    - ...
  - Typically verified: (memory) state.
  - End-users care about: what's on their screen.
- => ...ort to verify Input/Output (I/O)

different developers

Modularity

different developers

Modularity

R...ments

- e.g. define I/O action download on top of tcp_write and file_write, etc.
- Modularity
  - e.g. combine independent I/O action tcp_write with file_write
- Non-terminating programs (part WIP)
  - e.g. {} **while true** ... {these I/O happened}: postconditio... useless
- Actions depend on outcome of actions
  - e.g. ...ntaining filenames to read

- {}
  - No I/O allowed
- { time(t_1) } ... { tim...
  - No I/O allowed
  - A time like t_1 ≈ a po... time.
  - Doing I/O "increases" time
- { time(t_1) ⋆ print_io(t_1, 'h', t_2) }
  print_char('h');
  { time(t_2) }
  - Doing print_char('h')
    - requires a permission print_io(t_1, 'h', t_2)
    - requires a time(t_1)
    - disposes the permission
    - "increases" the time to t_2

- { time(t_1) ⋆ print_io(t_1, 'h', t_2) ⋆ print_io(t_2, 'i', t_3) }
  ...
  { time(t_3) }
  - Can print "hi", "h", "".
  - If terminates: can only print "hi".
  - Can not print: "x", "i", "ih", ...
- { time(t_1) ⋆ print_io(t_1, 'h', t_2) ⋆ print_io(t_1, 'i', t_2) }
  ...
  { time(t_2) }
  - Can print "h", "i", "".
  - If terminates: has printed either "h" or
  - Can not print: "x", "hi", ...

- { time(t_1) ⋆ split(t_1, t_2, t_3)
  ⋆ read_string_io(t_2, ...
  ⋆ print_string_io(t_3, str, t_5)
  ⋆ join(t_4, t_5, t_6)
  }
  ...
  { time(t_4) }
- split(t_1, t_2, t_3) consumes time(t_1) and yields time(t_2) and ti...

...print_string_io(t_1, str, t_2) =
...il **then**
...
else (
print_io(t_1, head(str), $t_{between}$)
⋆ print_string_io($t_{between}$, tail(str), t_2)
)

...uild actions using actions (compositionality)

time(...) ... print_string_io(t_1, "hello world"...
...
tim...

Linking arguments

- { time(t_1) ⋆ read_string_io(t_1, str, t_2)
  ⋆ print_string_io(t_2, str, t_3)
  ...
  ...t_3)}

Uncon...d o...terleaving

...(t_2) ...)
...ring_io(t_2, s...
⋆ ...ring_io(t_3, str, ...
}
...
{ time(t_4) ⋆ time(t_5) }

- Allows buffering of any size.
- How to get two times (time(t_2) and time(t_3))?

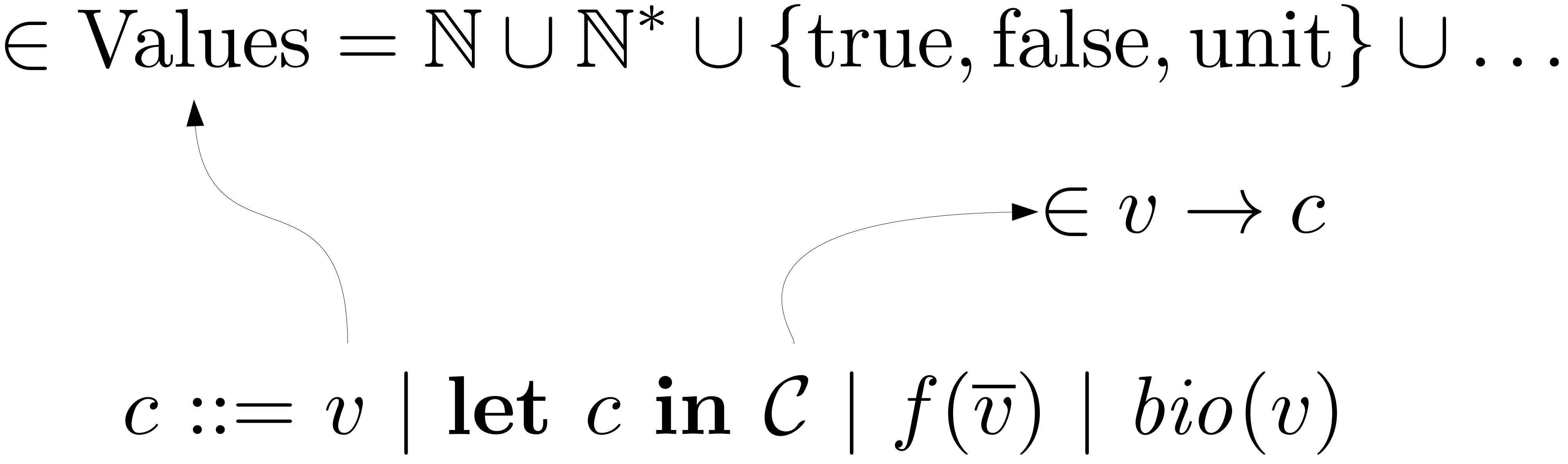
What ... do now?

- ...
  2. ... wanted I/O behaviour
  3. Write a proof (add annotations)
  4. Feed to proofchecker software.
     - Output: OK or Not OK.
- With support for:
  - Modularity
  - Compositionality
  - ...

TOC

- $c ::= \ldots$
- $c \Downarrow \tau, v$
- $C ::= \ldots$
- $h \Downarrow \mathbb{T}$
-

$$\begin{tikzcd}
& P \subseteq \lfloor \mathrm{wp}(c, \lfloor Q \rfloor) \rfloor \arrow[d] & \vdash \{P\}\, c\, \{Q\} \arrow[l] \\
\mathbb{T} \sim \tau & \mathrm{safe}(h, \tau, Q(v)) \arrow[l] &
\end{tikzcd}$$



$$\in \text{Values} = \mathbb{N} \cup \mathbb{N}^* \cup \{\text{true}, \text{false}, \text{unit}\} \cup \ldots$$

$$\in v \to c$$

$$c ::= v \mid \textbf{let } c \textbf{ in } \mathcal{C} \mid f(\overline{v}) \mid bio(v)$$

$$c \Downarrow \tau, v$$

Similar to: "*Trace-Based Coinductive Operational Semantics for While*", Keiko Nakata and Tarmo Uustalu, TPHOLs 2009

$$\sigma ::= bio(v, v) \mid \text{no\_io}$$

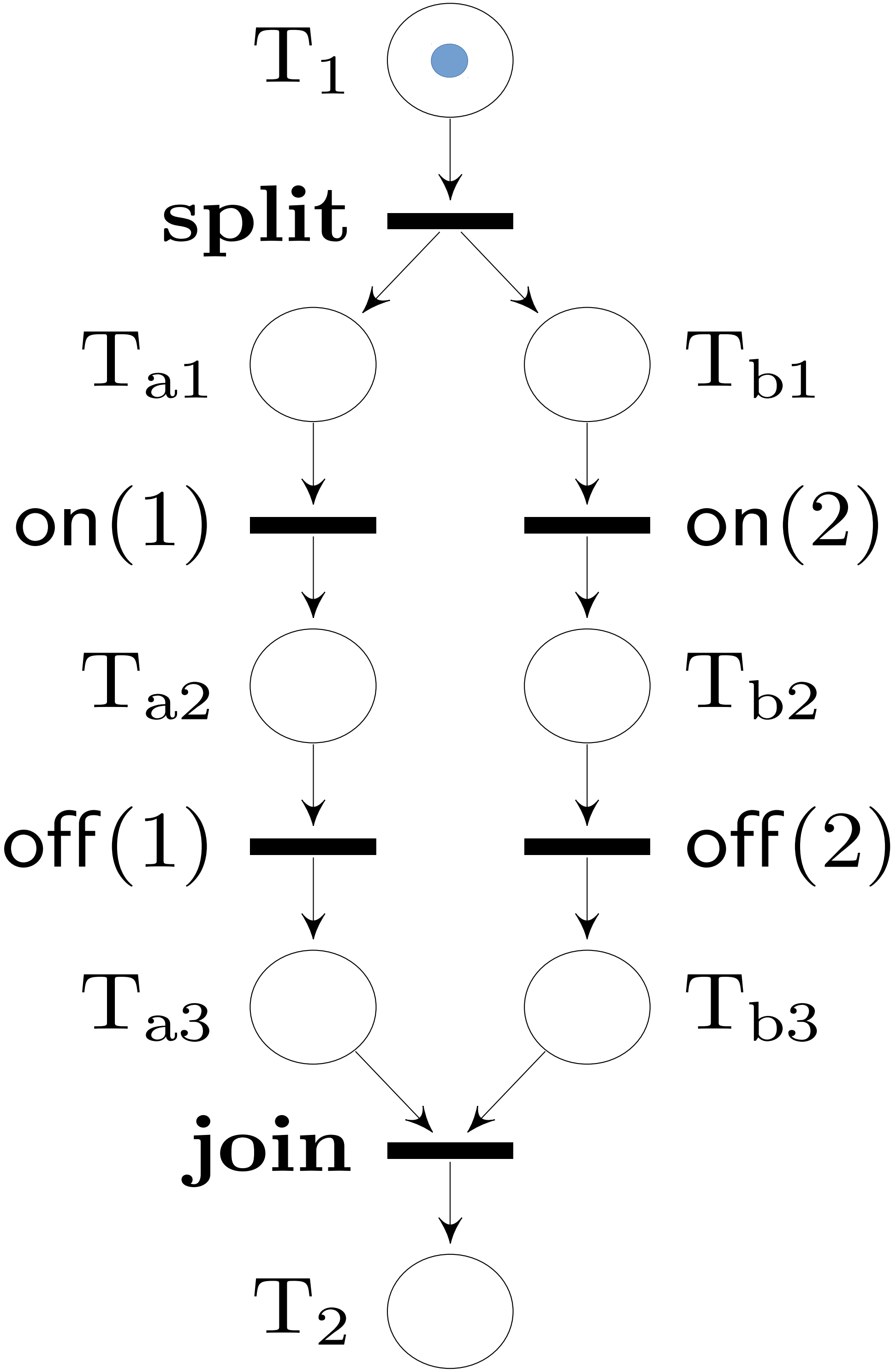
$$\tau ::= \langle\rangle \mid \sigma \cdot \tau$$

Coinductive

$$\begin{array}{c} \\ \hline\hline bio(v) \Downarrow bio(v, v_r) \cdot \langle\rangle, v_r \end{array}\ \text{Bio}$$

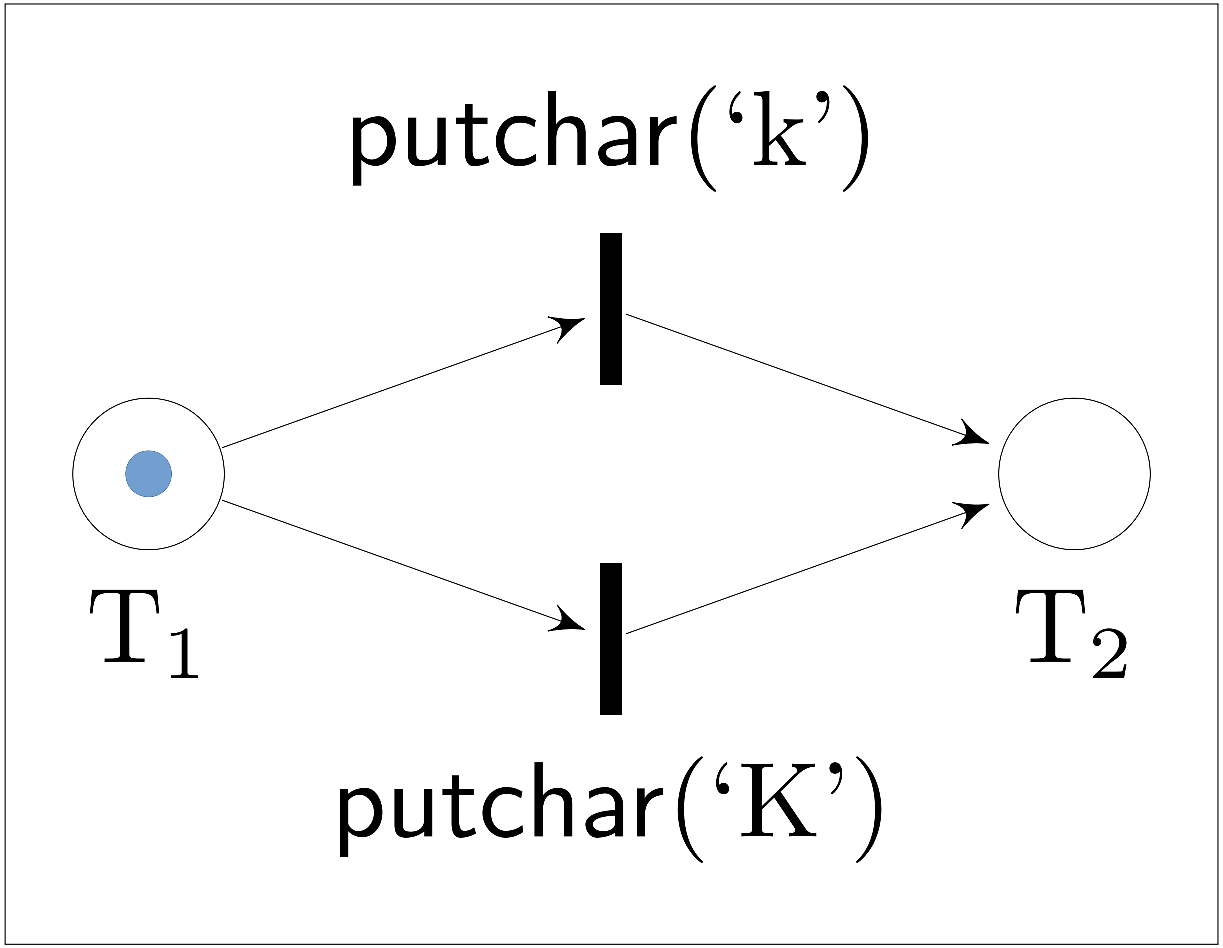
$$\begin{array}{c} (\mathrm{fc}(f))(\overline{v_1}) \Downarrow \tau, v_2 \\ \hline\hline f(\overline{v_1}) \Downarrow \mathrm{no\_io} \cdot \tau, v_2 \end{array}\ \text{App} \qquad \begin{array}{c} \\ \hline\hline v \Downarrow \langle\rangle, v \end{array}\ \text{Val}$$

$$\begin{array}{c} c \Downarrow \tau_1, v_1 \qquad \mathcal{C}(v_1) \Downarrow \tau_2, v_2 \\ \hline\hline \textbf{let}\ c\ \textbf{in}\ \mathcal{C} \Downarrow \tau_1 \cdot \tau_2, v_2 \end{array}\ \text{Let}$$

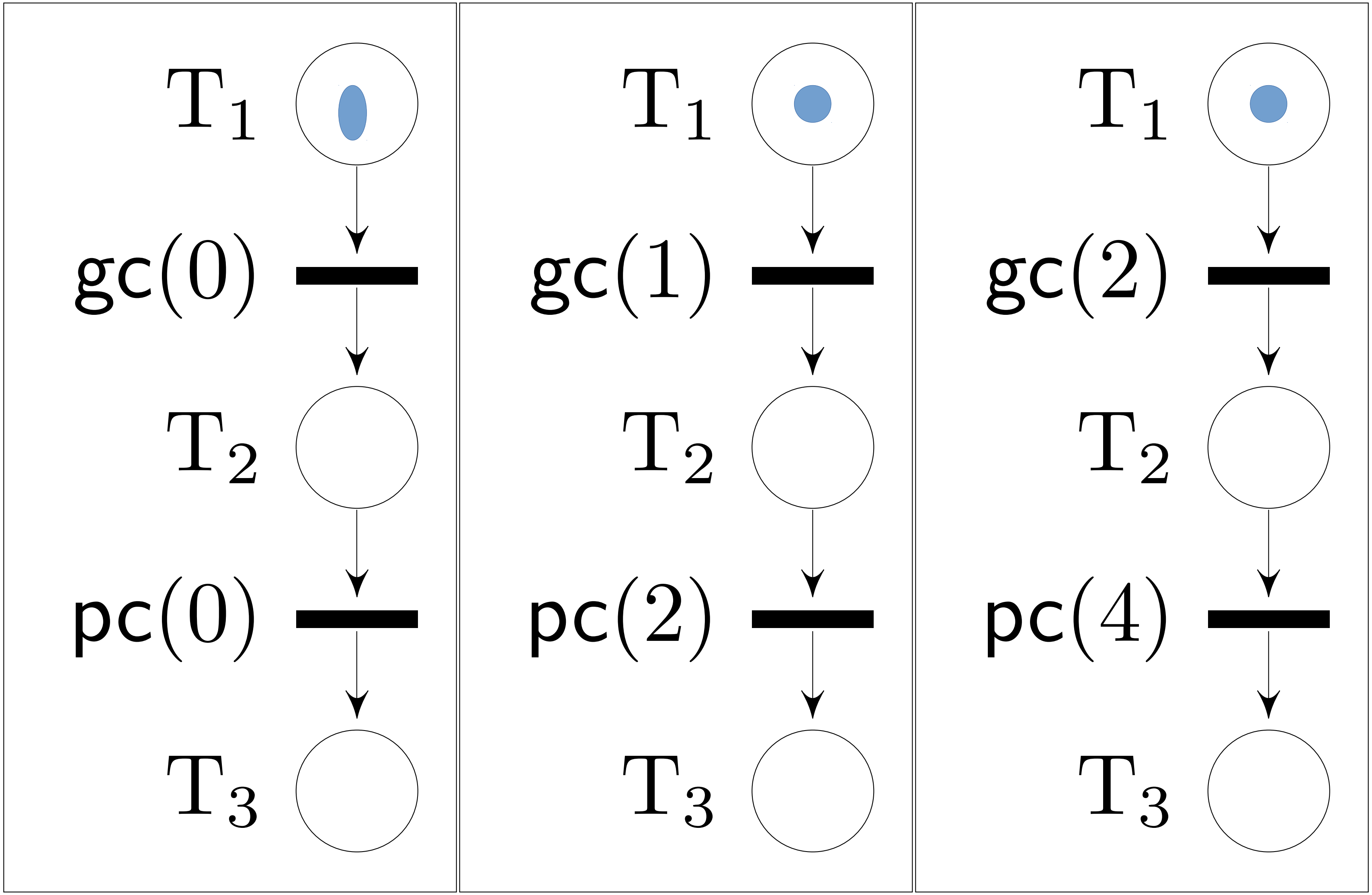


- $\langle \epsilon, \mathsf{on}(1), \mathsf{off}(1), \mathsf{on}(2), \mathsf{off}(2), \epsilon \rangle$
- $\langle \epsilon, \mathsf{on}(1), \mathsf{on}(2), \mathsf{off}(2), \mathsf{off}(1), \epsilon \rangle$
- $\langle \epsilon, \mathsf{on}(2), \mathsf{off}(2), \mathsf{on}(1), \mathsf{off}(1), \epsilon \rangle$

Multiple exec => prog. choice



Multiple nets => env. choice



$$
\begin{aligned}
C ::= \; & bio(t, v, v, t) \mid \text{CENSORED} \\
& \mid \textbf{split}(\mathrm{t},\ \mathrm{t},\ \mathrm{t}) \mid \textbf{join}(\mathrm{t},\ \mathrm{t},\ \mathrm{t}) \mid \textbf{token}(\mathrm{t})
\end{aligned}
$$

$$C \in \text{Chunks}$$

$$h \in \text{Heaps} = \text{Chunks} \to \mathbb{N} \cup \{\infty\}$$

$$P \subseteq \text{Heaps}$$

$$h \Downarrow \mathbb{T}$$

$$\mathbb{T} ::= \langle\rangle \mid \epsilon \cdot \mathbb{T} \mid bio(v_o, v_i) \cdot \mathbb{T} \qquad \text{(coind.)}$$

- $\{\!\!\{\mathbf{token}(t_1),$
  $bio(t_1, v_o, v_i, t_2)\}\!\!\} \uplus h \xrightarrow{bio(v_o, v_i)} \{\!\!\{\mathbf{token}(t_2)\}\!\!\} \uplus h$

- $\{\!\!\{\mathbf{token}(t_1),$
  $\mathbf{split}(t_1, t_2, t_3)\}\!\!\} \uplus h \xrightarrow{\epsilon} \{\!\!\{\mathbf{token}(t_2), \mathbf{token}(t_3)\}\!\!\} \uplus h$

- $\{\!\!\{\mathbf{token}(t_1), \mathbf{token}(t_2),$
  $\mathbf{join}(t_1, t_2, t_3)\}\!\!\} \uplus h \xrightarrow{\epsilon} \{\!\!\{\mathbf{token}(t_3)\}\!\!\} \uplus h$

$$\frac{}{h \Downarrow \langle\rangle}\ \text{Stop}$$

$$\frac{h \xrightarrow{\epsilon} h' \qquad h' \Downarrow \mathbb{T}}{h \Downarrow \epsilon \cdot \mathbb{T}}\ \text{Epsilon}$$

$$\frac{h \xrightarrow{bio(v_o, v_i)} h' \qquad h' \Downarrow \mathbb{T}}{h \Downarrow bio(v_o, v_i) \cdot \mathbb{T}}\ \text{Bio}$$

$$c \Downarrow \tau, v$$

$$h \Downarrow \mathbb{T}$$

$$\mathbb{T} \sim \tau$$

$$\begin{array}{c} \\ \hline\hline \mathbb{T} \sim \langle\rangle \end{array} \ \text{Empty}$$

$$\begin{array}{c} \mathbb{T} \sim \tau \\ \hline\hline \epsilon^* \cdot bio(v_o, v_i) \cdot \mathbb{T} \sim bio(v_o, v_i) \cdot \tau \end{array} \ \text{Bio}$$

$$\begin{array}{c} v_i \neq v_i' \\ \hline\hline \epsilon^* \cdot bio(v_o, v_i') \cdot \mathbb{T} \sim bio(v_o, v_i) \cdot \tau \end{array} \ \text{Contra}$$

$$\begin{array}{c} \mathbb{T} \sim \tau \\ \hline\hline \mathbb{T} \sim \text{no\_io} \cdot \tau \end{array} \ \text{NoIO}$$

$$
\begin{array}{l}
\mathit{execOK1}(h, \tau, Q(v)) \\
\iff \\
\exists \mathbb{T}.\, h \Downarrow \mathbb{T} \;\wedge\; \mathbb{T} \sim \tau
\end{array}
$$

Easy to use

$$progOK(\,\{P\}\,c\,\{Q\}\,)$$

$$\Downarrow$$ Soundness proof

$$\begin{array}{r}\forall h,\tau,v.\ P(h)\\ \land\ c \Downarrow \tau, v\end{array} \Rightarrow\ execOK(h,\tau,Q(v))$$

Simple & strong def

$$\begin{tikzcd}
\mathit{ProgOK} & & \mathit{ExecOK} \\
\vdash \{P\}\, c\, \{Q\} \arrow[d] & \Rightarrow & \mathrm{safe}(h, \tau, Q(v)) \\
P \subseteq \lfloor \mathrm{wp}(c, \lfloor Q \rfloor) \rfloor \arrow[rr] & \Rightarrow & \mathrm{safe}(h, \tau, Q(v)) \arrow[d] \\
P \subseteq \mathrm{wp}(c, Q) \arrow[u] & \Rightarrow & \exists \mathbb{T}.\, h \Downarrow \mathbb{T} \ \wedge\ \mathbb{T} \sim \tau
\end{tikzcd}$$

I expect time's up by now?