Local Reasoning for Robust Observational Equivalence

Dan R. Ghica[1]
Koko Muroya[1,2]
Todd Waugh Ambridge[1]

[1]SoCS, University of Birmingham
[2]RIMS, Kyoto University

PERR 2019 (Prague), 6 April 2019

Outline

1. Motivation
2. The SPARTAN Calculus
3. SPARTAN Semantics – Focussed Graph Rewriting
4. The Characterisation Theorem
5. An Application of The Characterisation Theorem
6. Conclusion & Further Work

Motivation

In a language, terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \iff \forall C.\, C[t_1] \simeq C[t_2]$$ *(for given notions of "terms", "all contexts" and "$\simeq$")*

Motivation

In a language, terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C.\, C[t_1] \simeq C[t_2]$$

(for given notions of "terms", "all contexts" and "$\simeq$")

Equivalence reasoning in a language with effects is *fragile*

Motivation

In a language, terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C.\, C[t_1] \simeq C[t_2]$$

(for given notions of "terms", "all contexts" and "$\simeq$")

Equivalence reasoning in a language with effects is *fragile*

How to make a ***robust*** way of reasoning about languages with effects?

Characterise effects by their consequences on the equational properties of the language

An equational law that is not broken by any semantic feature is *robust*

Motivation

In a language, terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C.\, C[t_1] \simeq C[t_2]$$ *(for given notions of “terms”, “all contexts” and “$\simeq$”)*

Equivalence reasoning in a language with effects is *fragile*

How to make a ***robust*** way of reasoning about languages with effects?

Characterise effects by their consequences on the equational properties of the language

An equational law that is not broken by any semantic feature is *robust*

We need a **common framework** for equational reasoning

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$
\begin{aligned}
t ::= &\ x \mid bind\ x \to t' \ in\ t'' \\
&\mid a \mid new\ a \multimap t'\ in\ t'' \\
&\mid \vec{y}.t' \\
&\mid \phi(\overrightarrow{t'}; \overrightarrow{t''})
\end{aligned}
$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects

They may take eager arguments and deferred arguments (thunks)

They are partitioned into *passive operations* and *active operations*

i.e. Values and redexes

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$
\begin{aligned}
t ::= \\
& x \mid bind\ x \to t'\ in\ t'' \\
\mid\ & a \mid new\ a \multimap t'\ in\ t'' \\
\mid\ & \vec{y}.\, t' \\
\mid\ & \phi(\overrightarrow{t'}; \overrightarrow{t''})
\end{aligned}
$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into *passive operations* and *active operations*
i.e. Values and redexes

$$n \mapsto n(-;-)$$

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$
\begin{aligned}
t ::= {} & \\
& x \mid \mathit{bind}\ x \to t' \ \mathit{in}\ t'' \\
& \mid a \mid \mathit{new}\ a \multimap t' \ \mathit{in}\ t'' \\
& \mid \vec{y}.\, t' \\
& \\
& \mid \phi(\overrightarrow{t'}; \overrightarrow{t''})
\end{aligned}
$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects

They may take eager arguments and deferred arguments (thunks)

They are partitioned into *passive operations* and *active operations*

i.e. Values and redexes

$$t + u \mapsto +(t, u; -)$$

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$t ::= x \mid bind\ x \to t'\ in\ t'' \mid a \mid new\ a \multimap t'\ in\ t'' \mid \vec{y}.t' \mid \phi(\overrightarrow{t'}; \overrightarrow{t''})$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into *passive operations* and *active operations*
i.e. Values and redexes

$$\lambda x.t \mapsto \lambda(-; x.t) \qquad u\ v \mapsto @(u, v; -)$$

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$
\begin{aligned}
t ::= &\ x \mid bind\ x \to t' \ in\ t'' \\
\mid &\ a \mid new\ a \multimap t' \ in\ t'' \\
\mid &\ \vec{y}.t' \\
\\
\mid &\ \phi(\overrightarrow{t'}; \overrightarrow{t''})
\end{aligned}
$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects

They may take eager arguments and deferred arguments (thunks)

They are partitioned into *passive operations* and *active operations*

i.e. Values and redexes

$$t;u \mapsto seq(t;u)$$

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$
\begin{aligned}
t ::=&\\
&\ x \mid bind\ x \to t'\ in\ t''\\
&\mid a \mid new\ a \multimap t'\ in\ t''\\
&\mid \vec{y}.\, t'\\
&\\
&\mid \phi(\overrightarrow{t'}; \overrightarrow{t''})
\end{aligned}
$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects

They may take eager arguments and deferred arguments (thunks)

They are partitioned into *passive operations* and *active operations*

i.e. Values and redexes

$$t := u \mapsto assign(t, u; -)$$

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

$$
\begin{aligned}
t ::= \\
& x \mid bind\ x \to t' \ in\ t'' \\
& \mid a \mid new\ a \multimap t' \ in\ t'' \\
& \mid \vec{y}.\, t' \\
& \mid \phi(\overrightarrow{t'}; \overrightarrow{t''})
\end{aligned}
$$

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects

They may take eager arguments and deferred arguments (thunks)

They are partitioned into *passive operations* and *active operations*

i.e. Values and redexes

$$!\, t \mapsto deref(t; -)$$

The SPARTAN Calculus

Three key intrinsic elements:

Variables	that manage *copying*
Names	that manage *sharing*
Thunks	that manage *evaluation*

Operations are defined extrinsically

$$\begin{aligned} t ::= &\; x \mid bind\ x \to t' \ in\ t'' \\ &\mid a \mid new\ a \multimap t'\ in\ t'' \\ &\mid \vec{y}.t' \\ &\mid \phi(\overrightarrow{t'}; \overrightarrow{t''}) \end{aligned}$$

An Alternative Intuition:

Universal Algebra

+ Sharing, Copying, Thunking

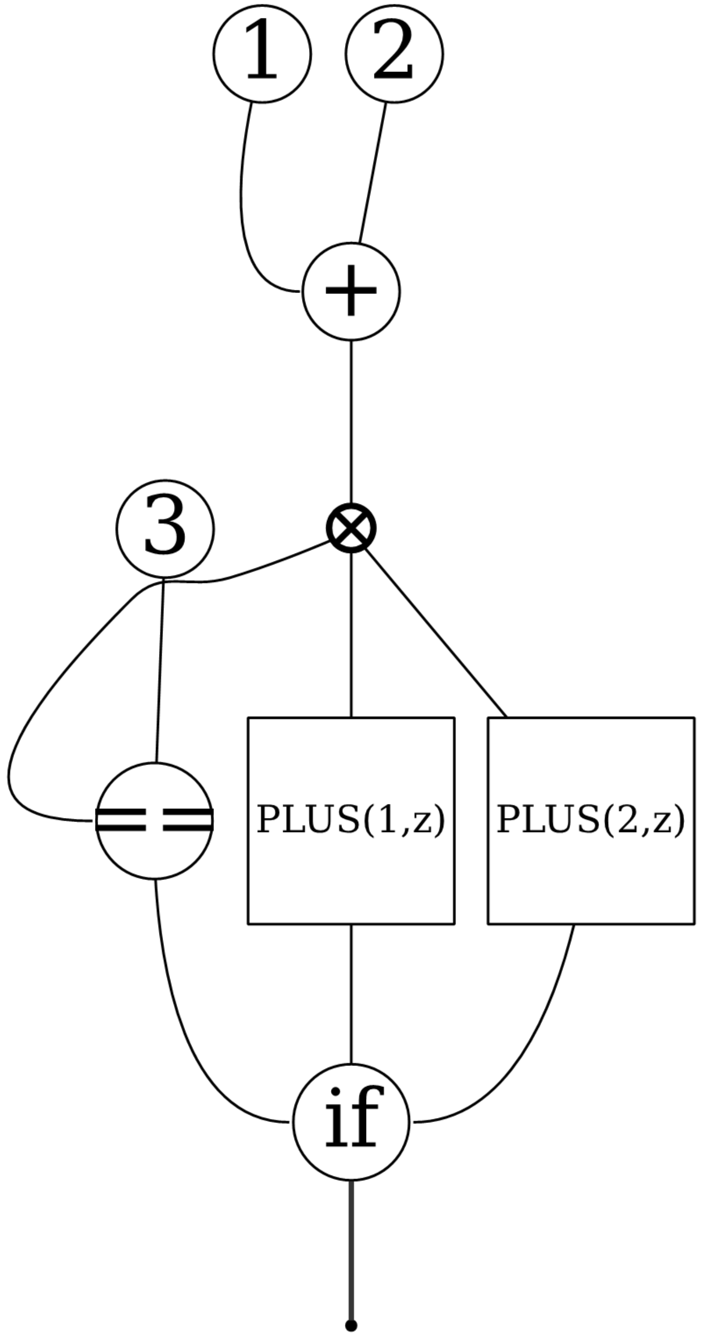
= A Programming Language with Effects

SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$

SPARTAN Semantics

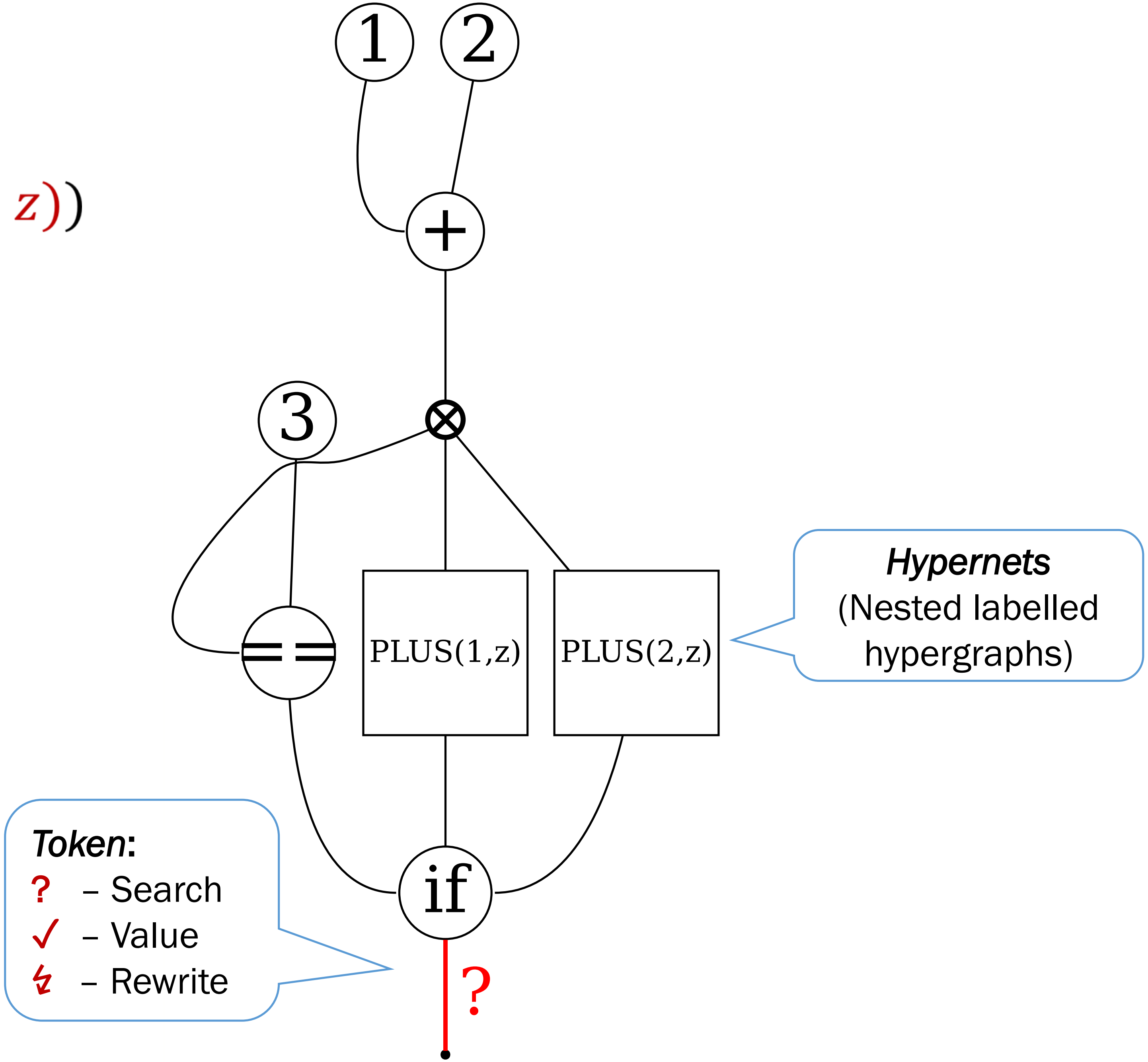
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



Hypernets
(Nested labelled hypergraphs)

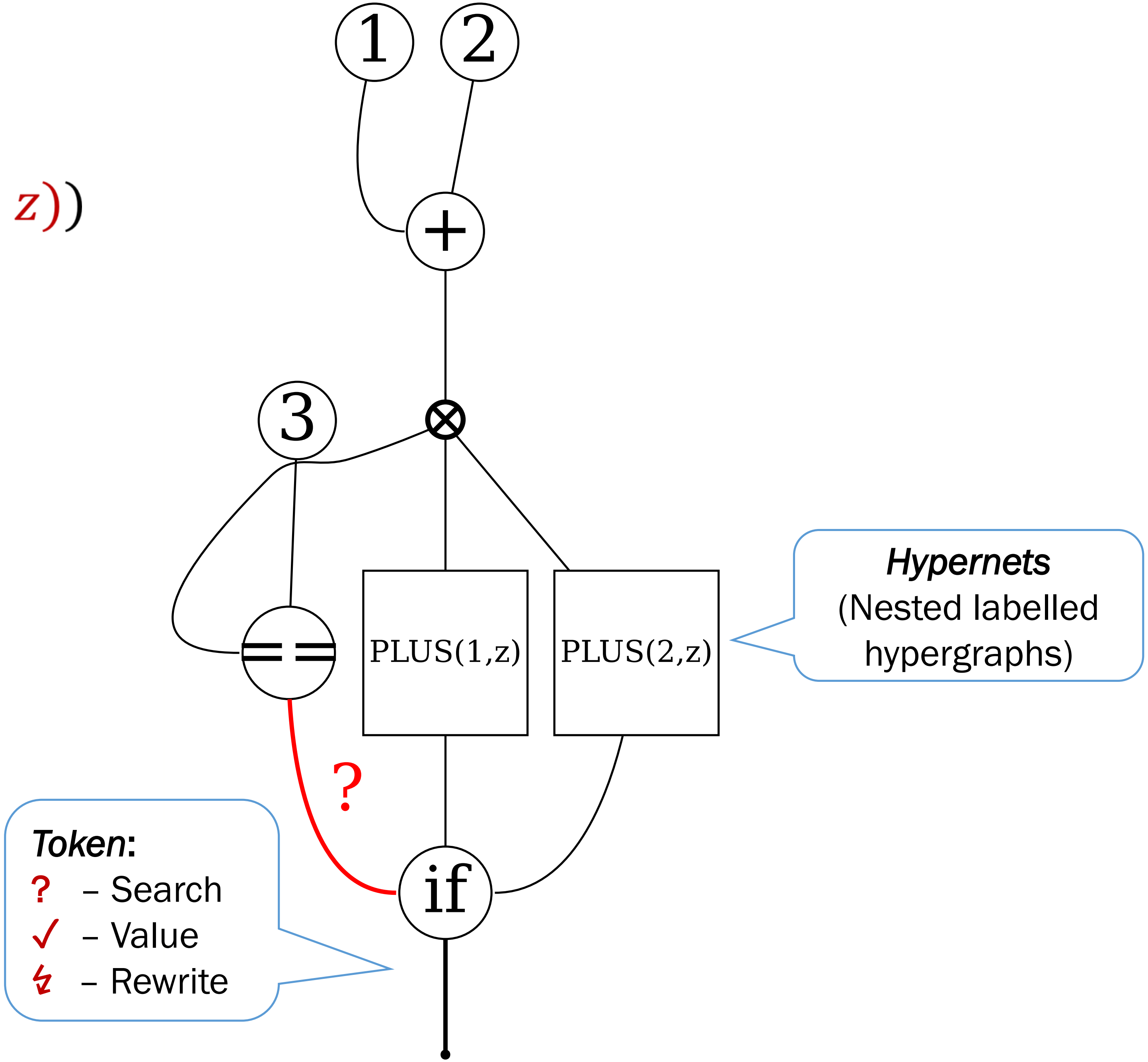
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



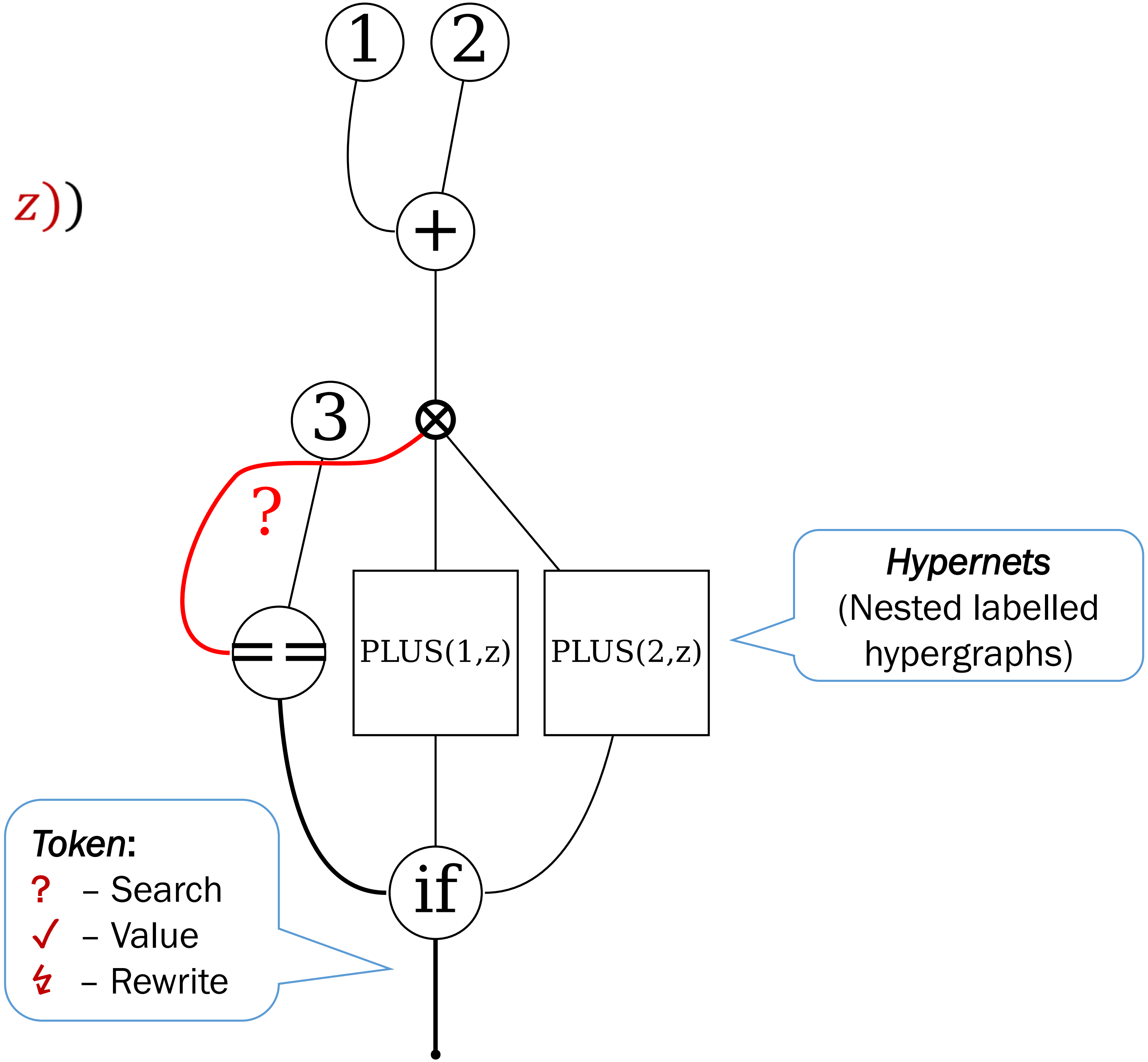
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



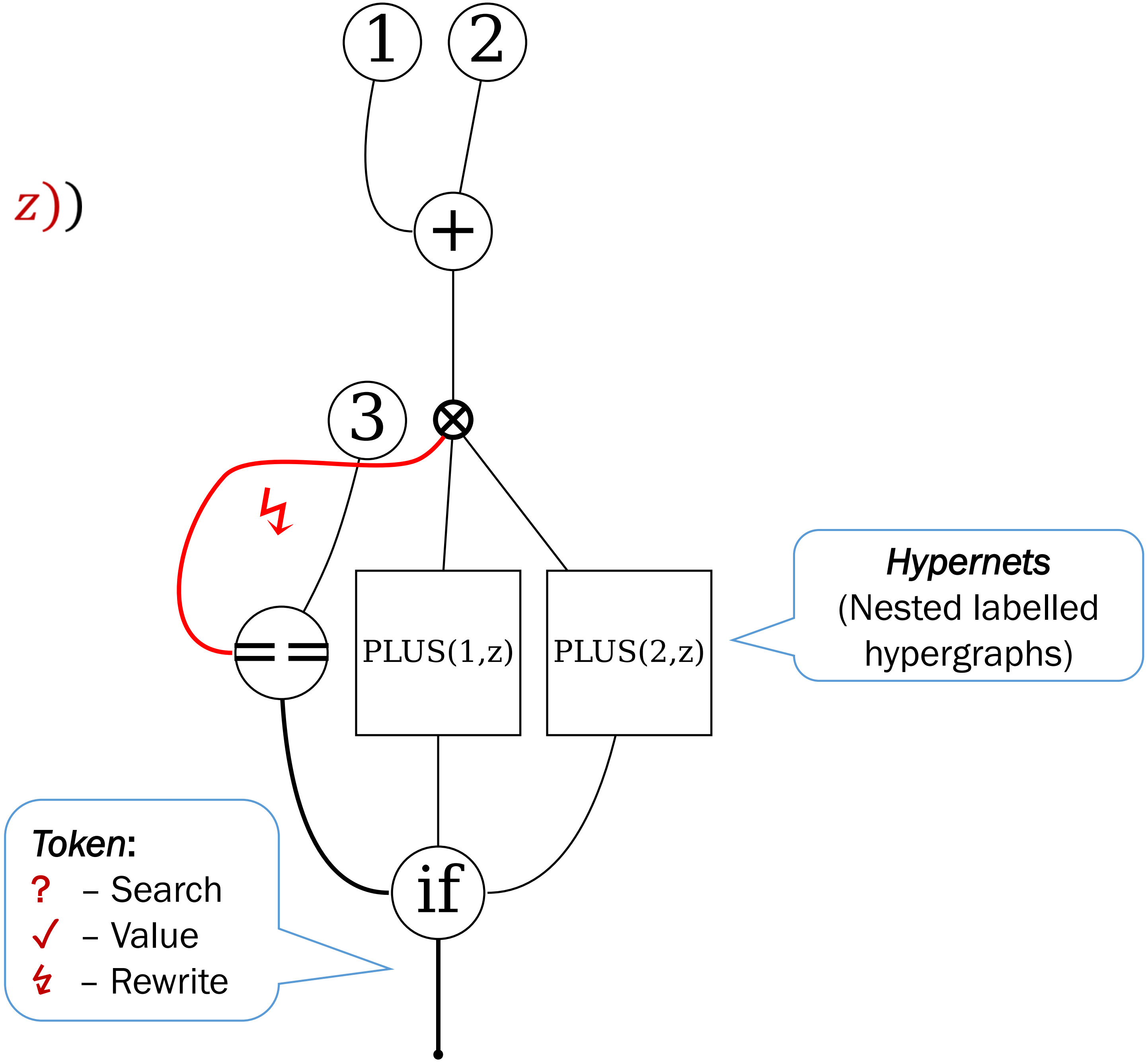
Spartan Semantics

$$bind\ z \to 1 + 2\ in$$
$$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$$



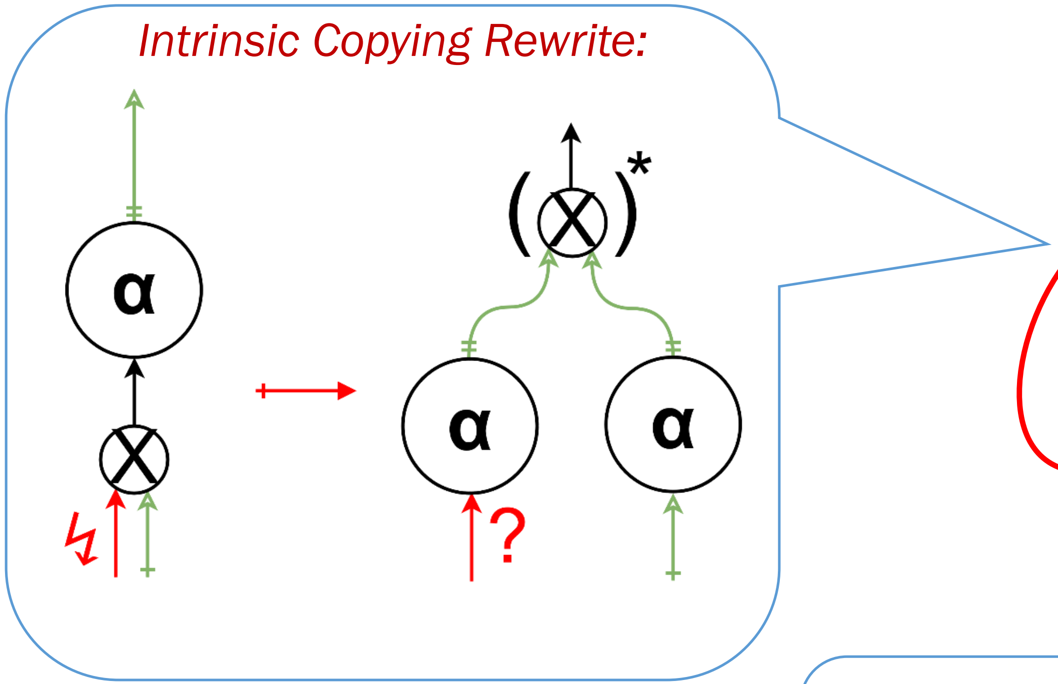
SPARTAN Semantics

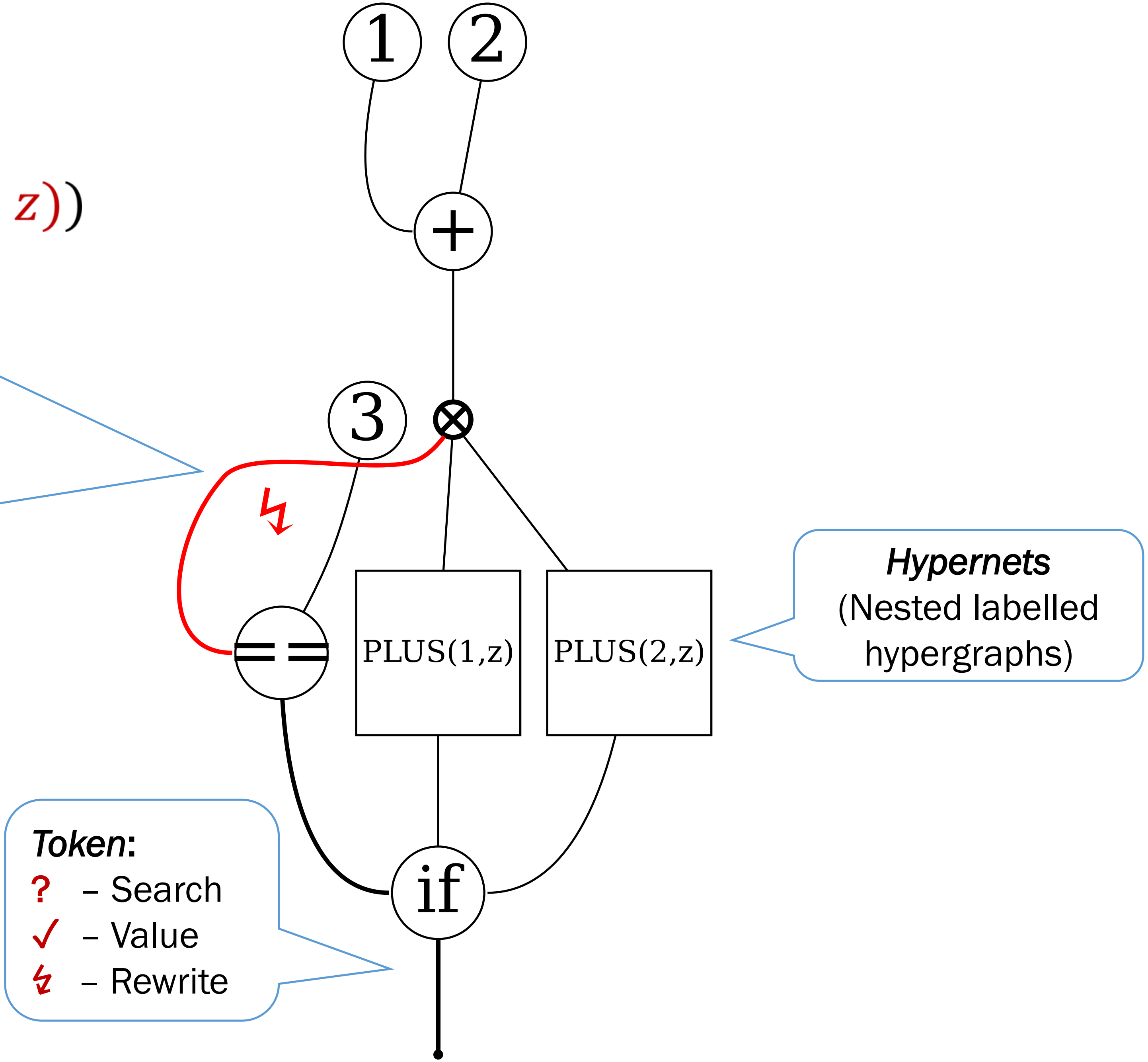
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

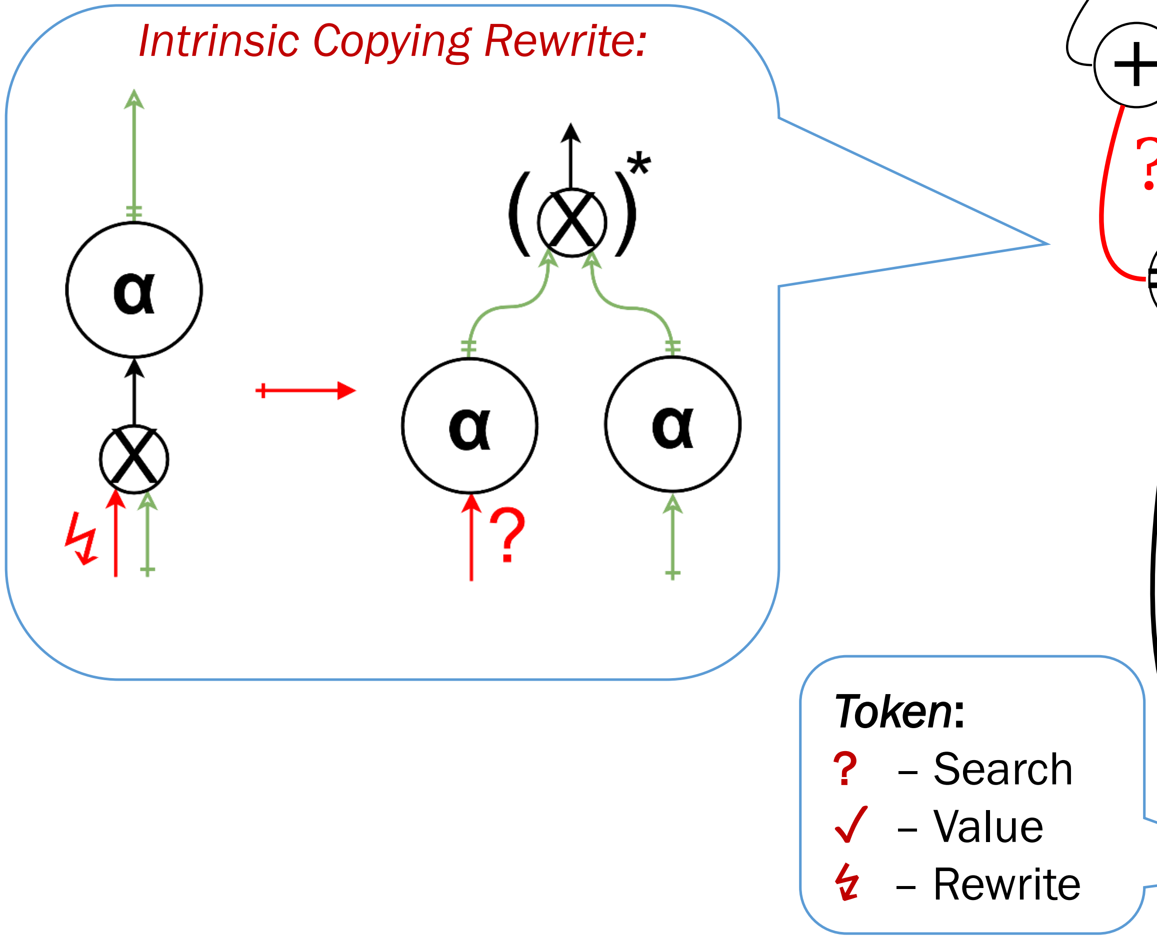
$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$

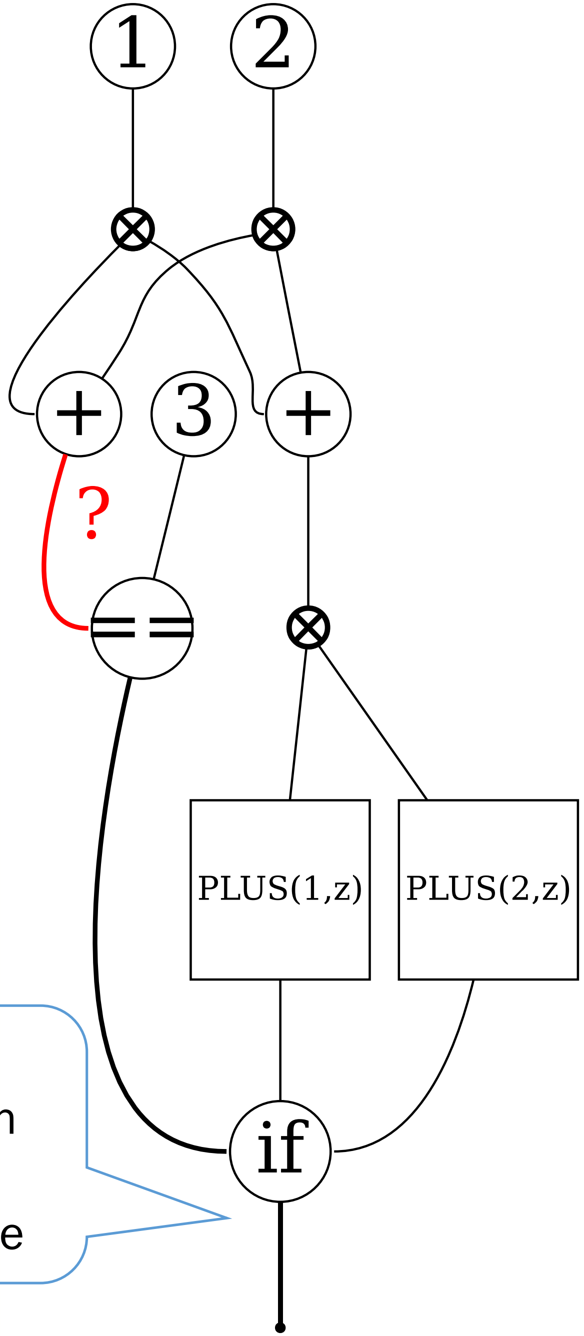




SPARTAN Semantics

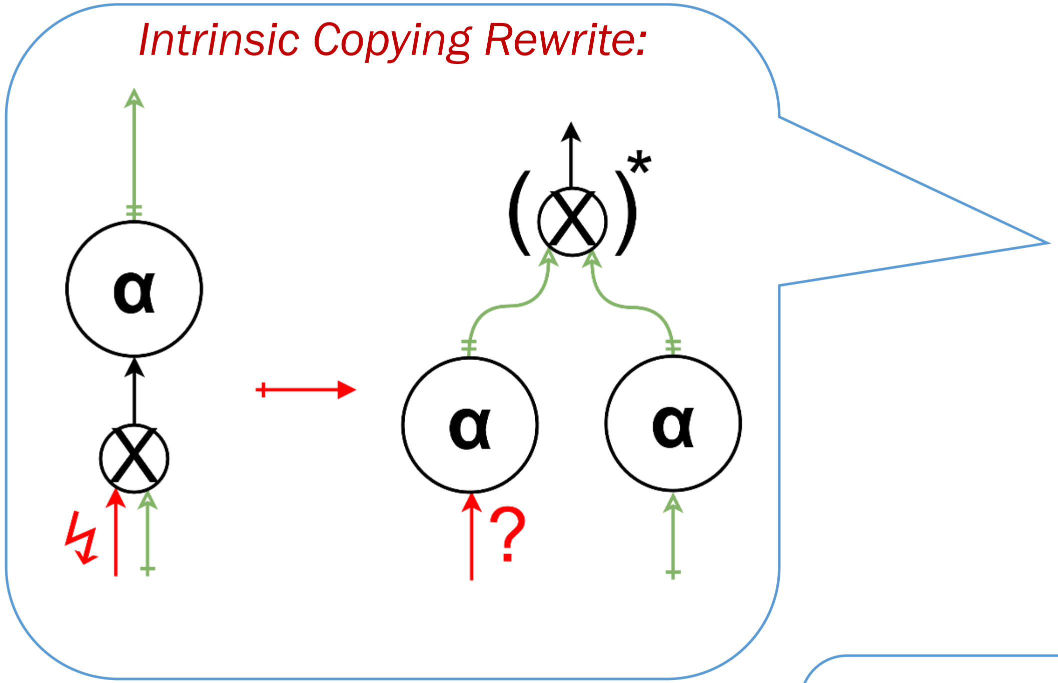
$$bind\ z \to 1 + 2\ in$$
$$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$$

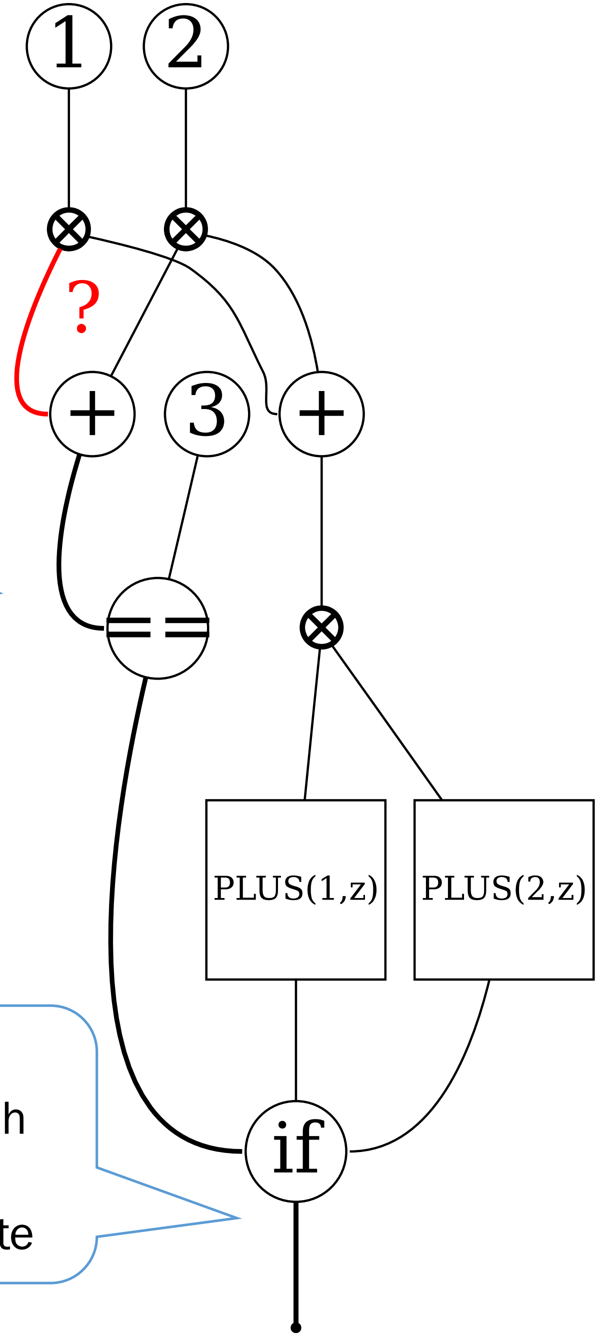




SPARTAN Semantics

$$bind\ z \to 1 + 2\ in$$
$$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$$



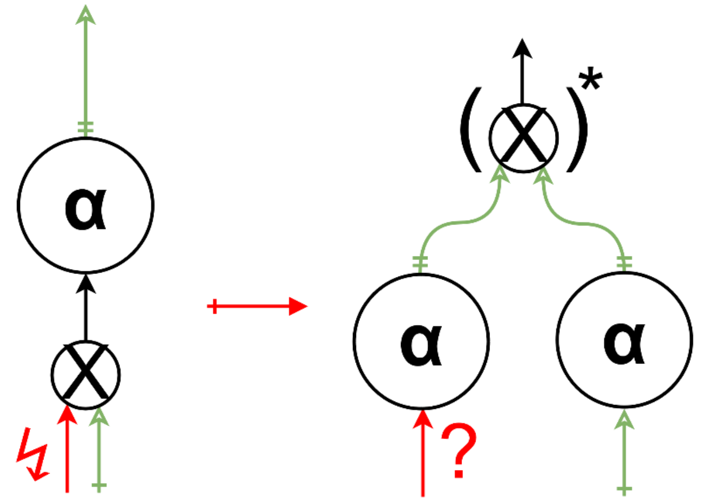


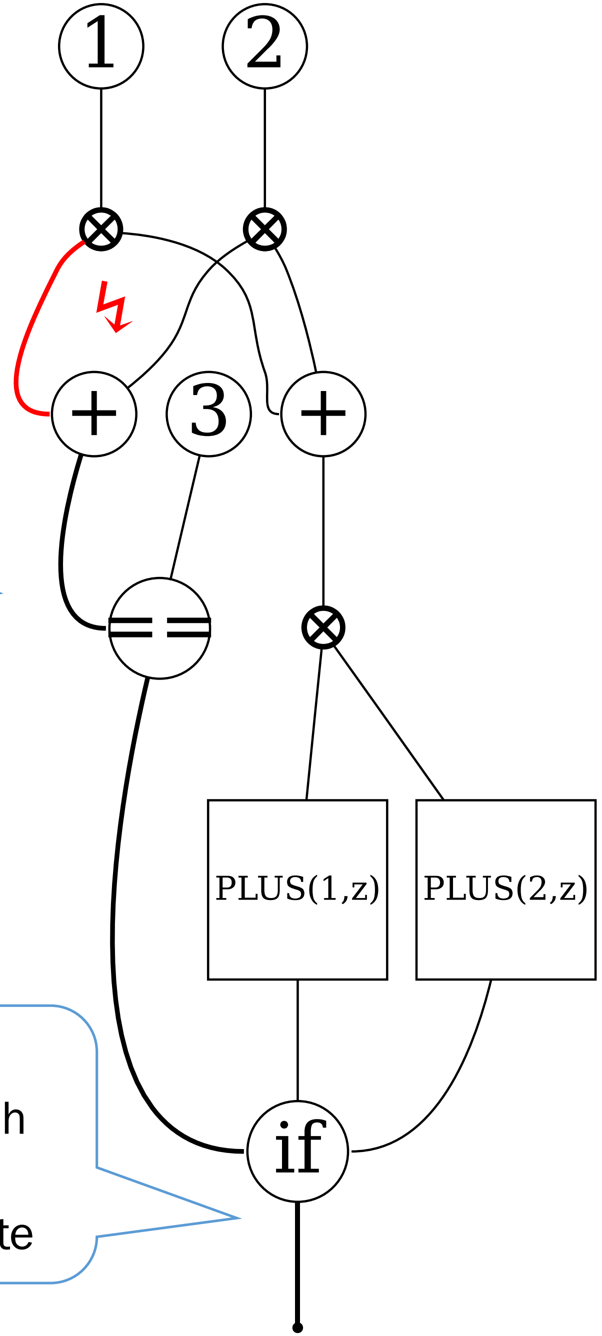
Token:
? – Search
✓ – Value
ϟ – Rewrite

SPARTAN Semantics

$$bind\ z \rightarrow 1 + 2\ in$$
$$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$$

Intrinsic Copying Rewrite:

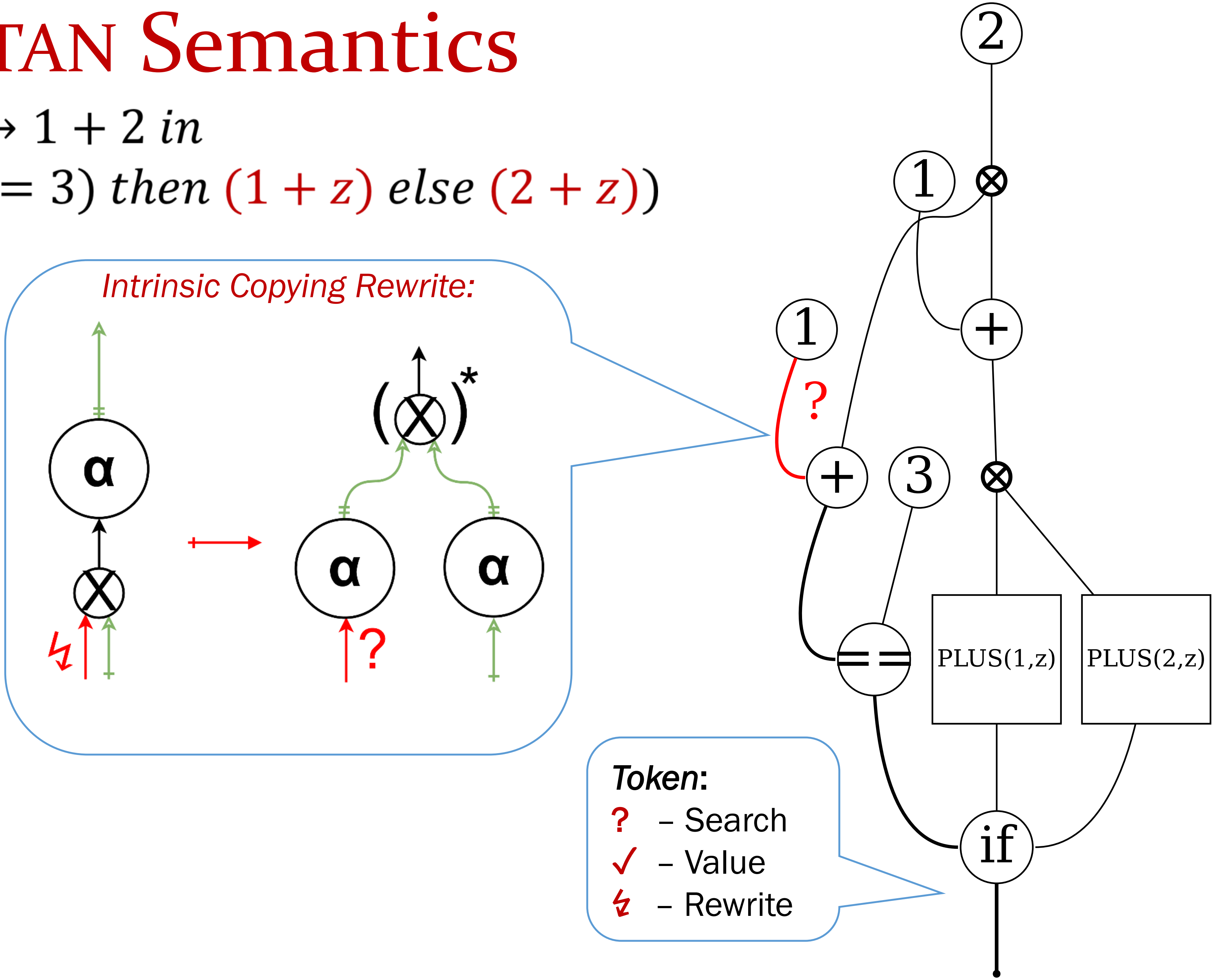




***Token*:**
- ? – Search
- ✓ – Value
- ↯ – Rewrite

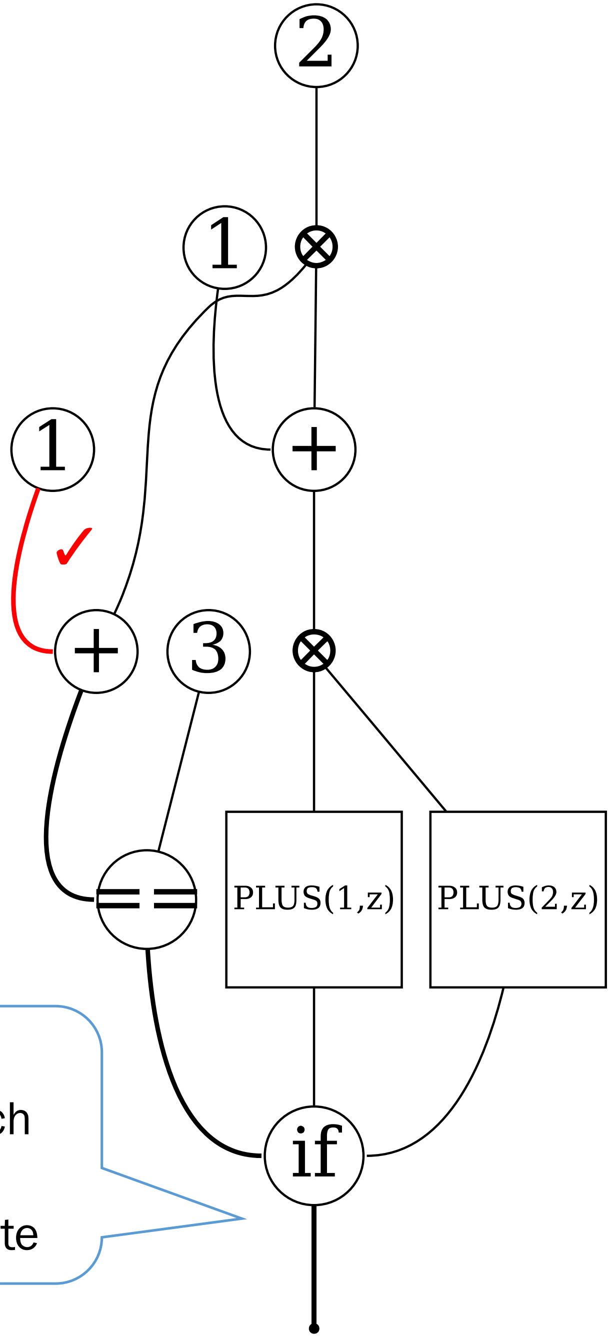
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

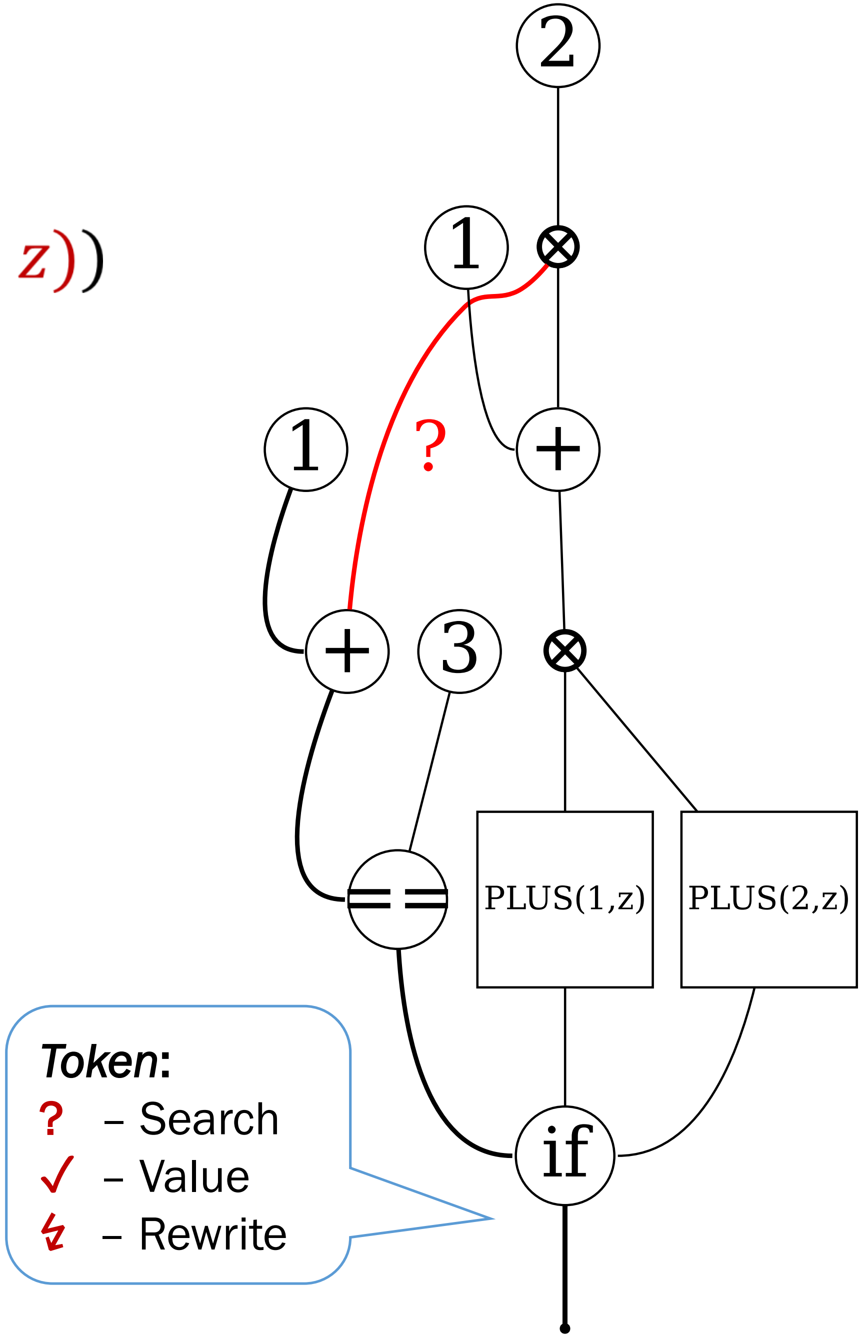
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



Token:
? – Search
✓ – Value
ϟ – Rewrite

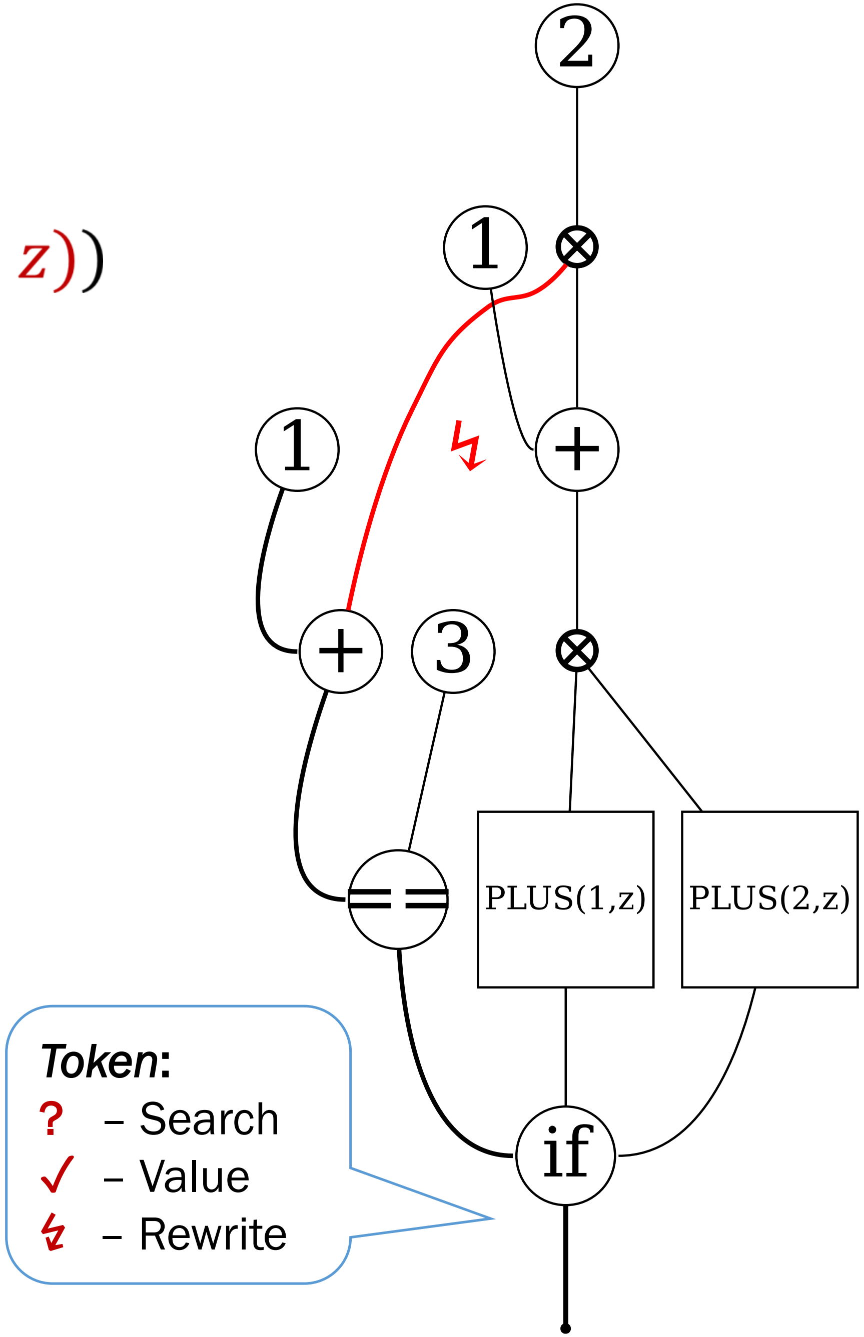
SPARTAN Semantics

$$bind\ z \to 1 + 2\ in$$
$$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$$



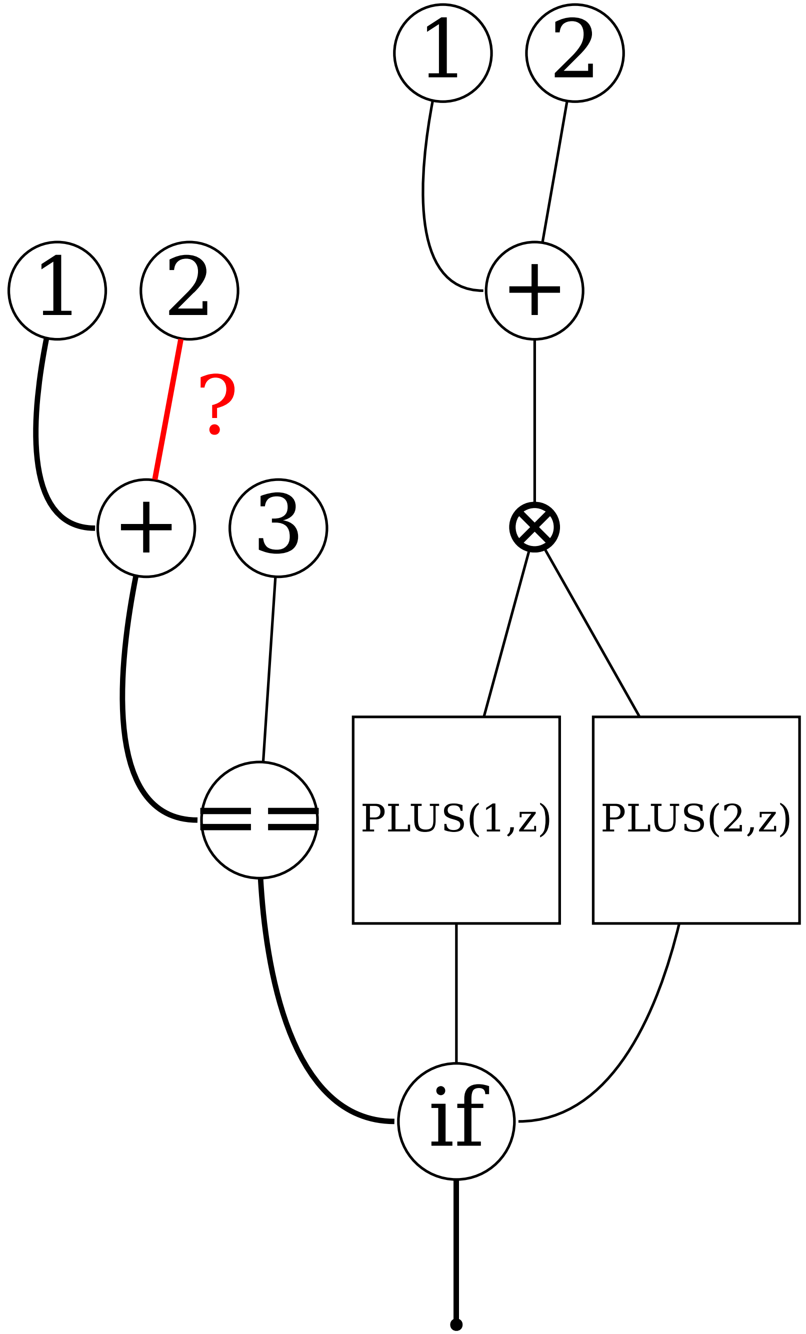
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

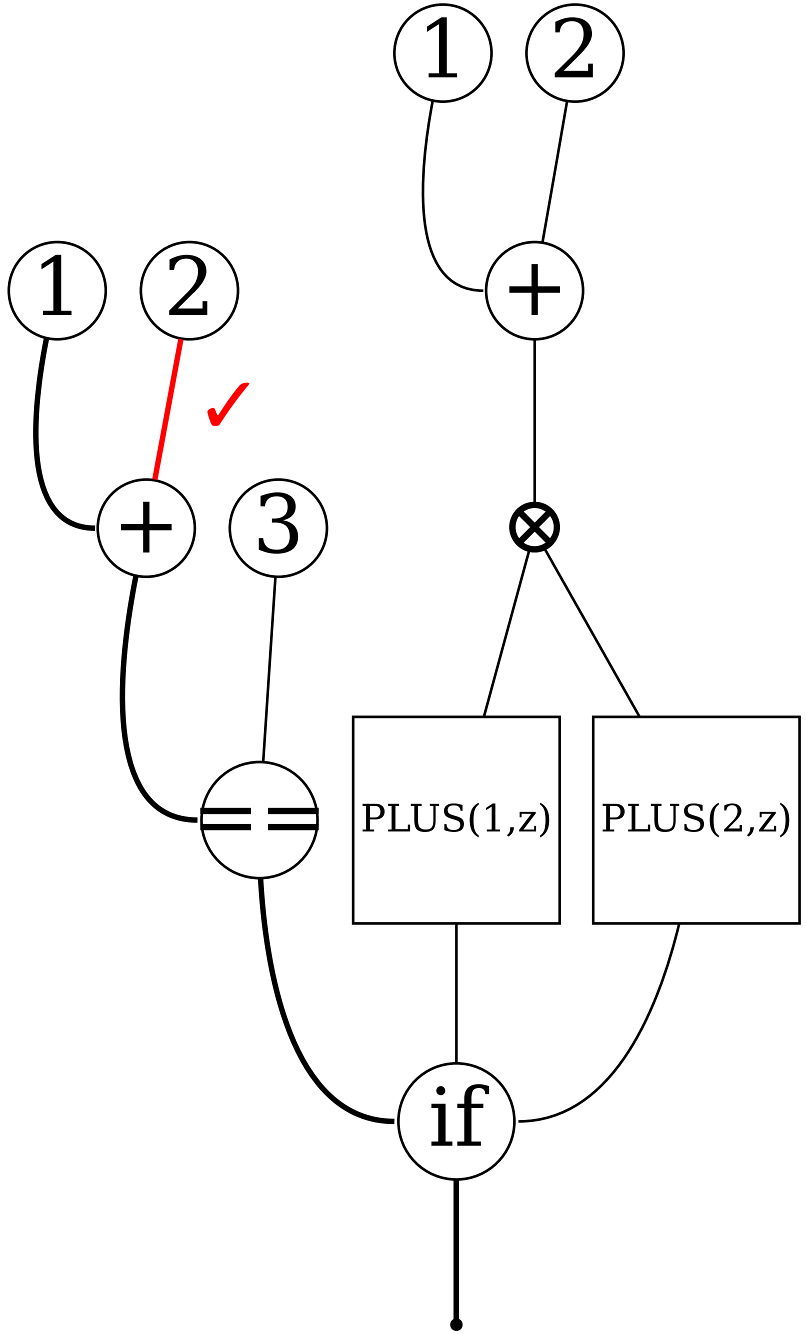
$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

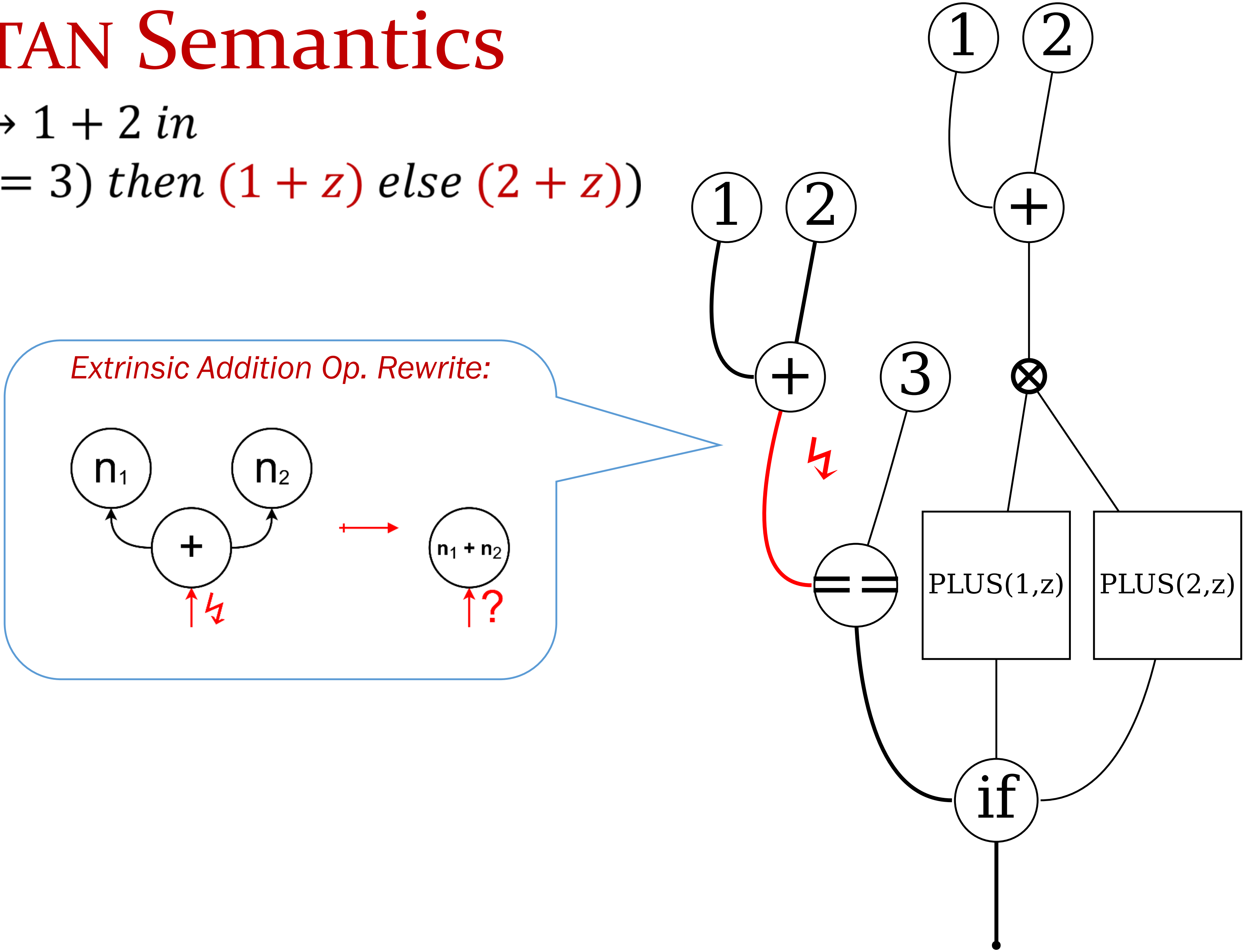
$bind\ z \rightarrow 1 + 2\ in$

$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



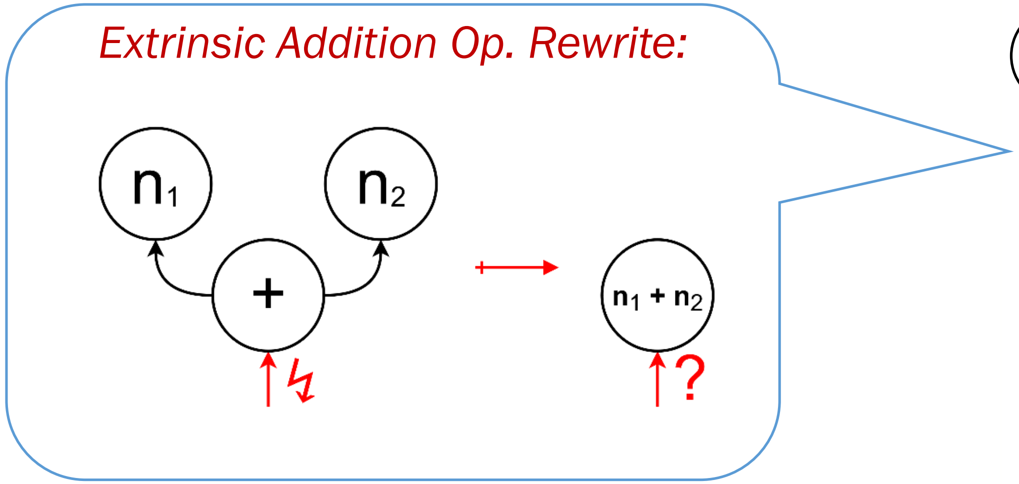
SPARTAN Semantics

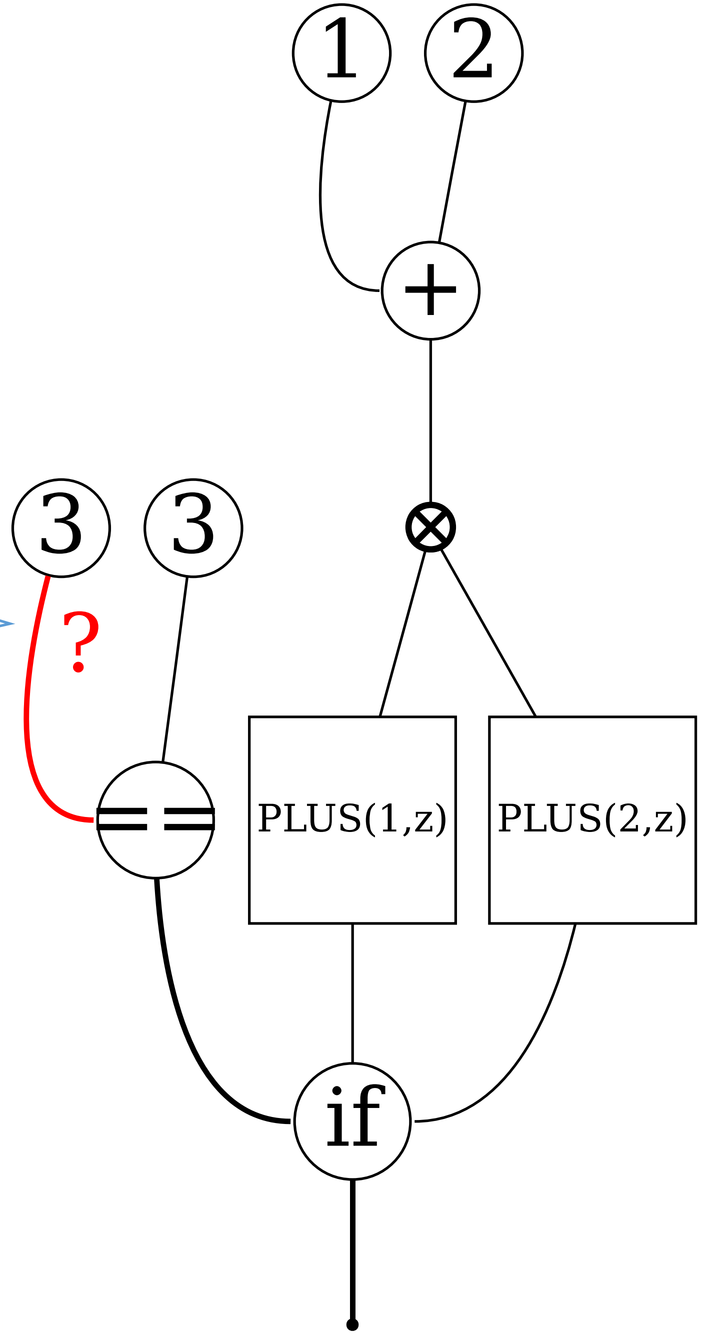
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$

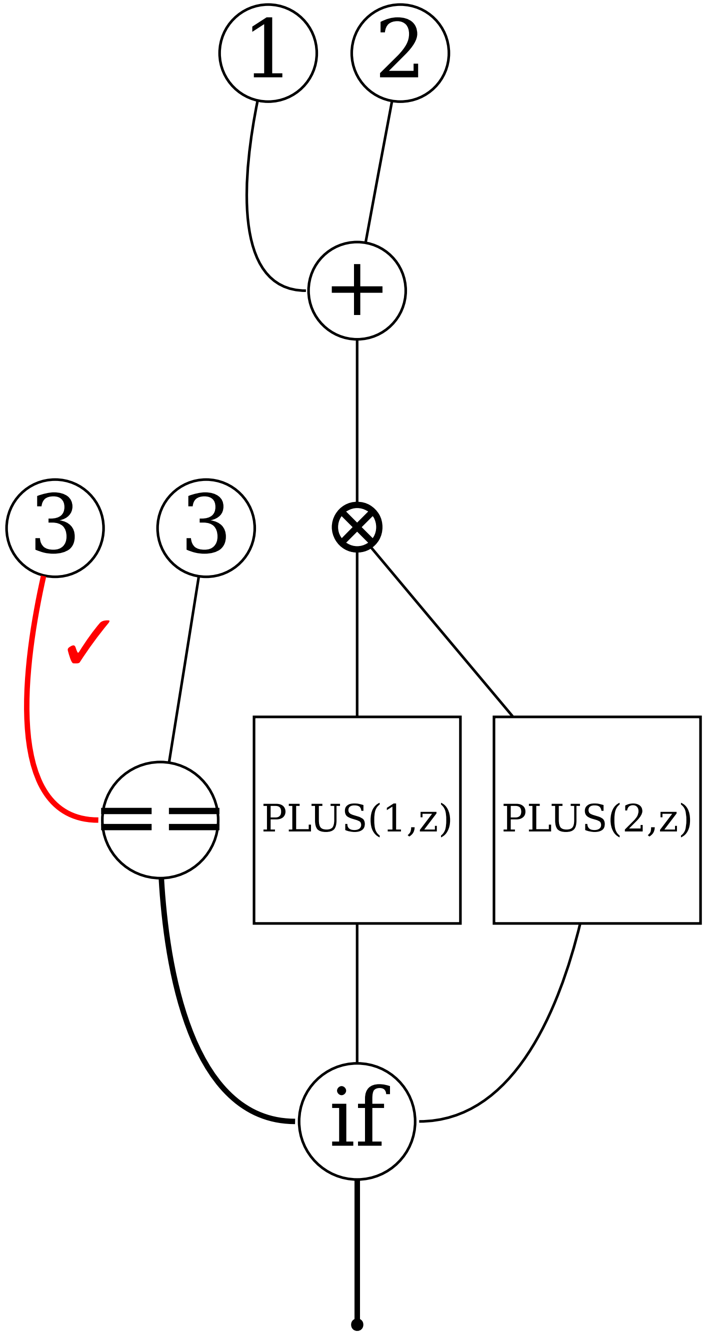




SPARTAN Semantics

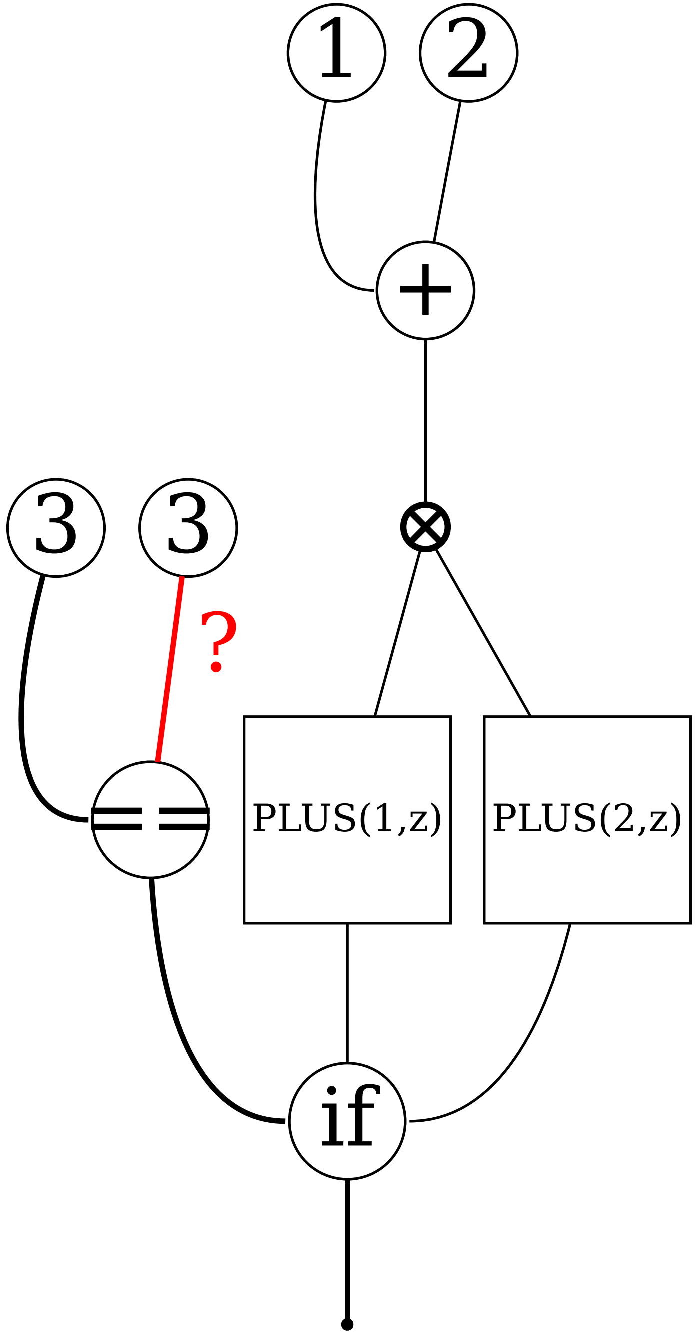
$bind\ z \to 1 + 2\ in$

$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



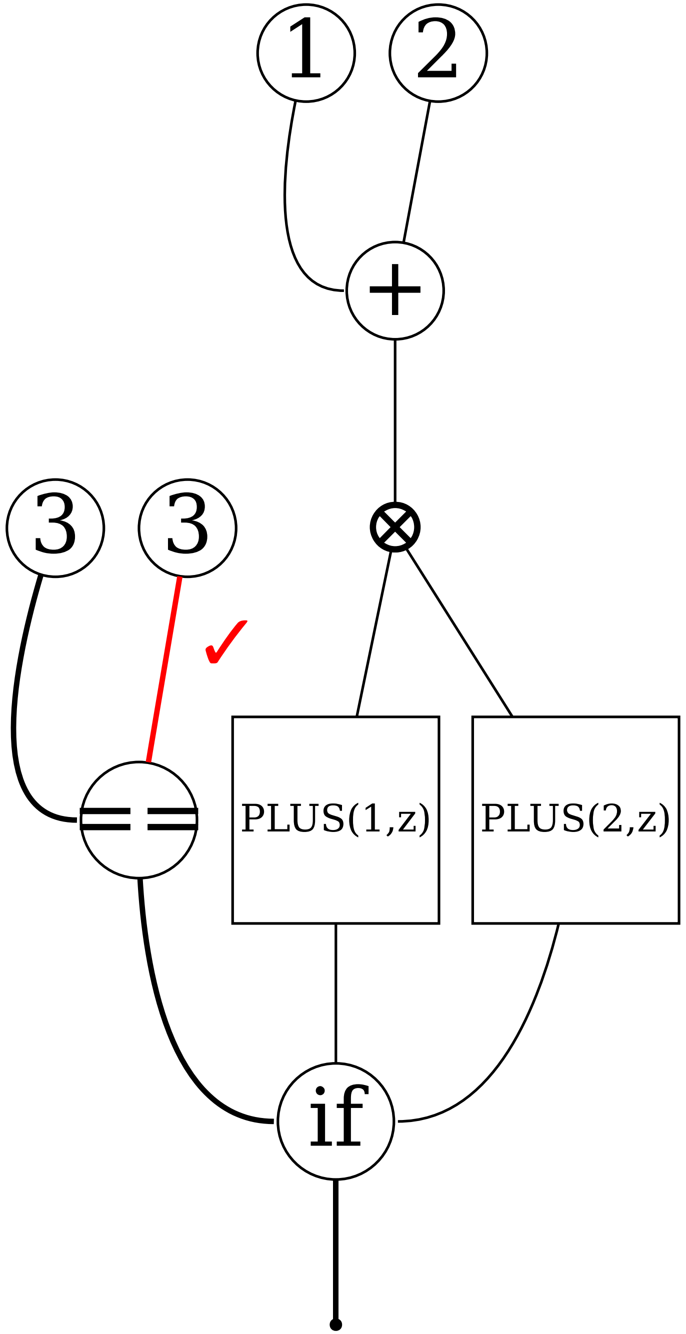
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



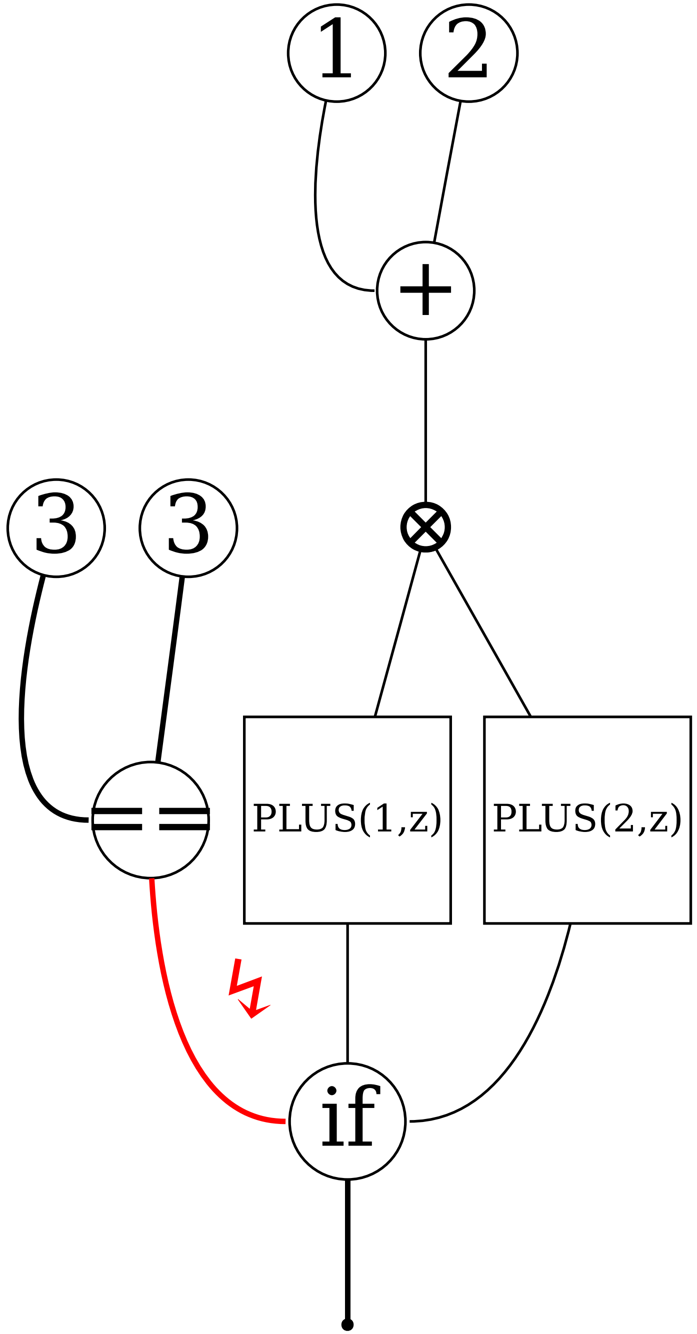
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



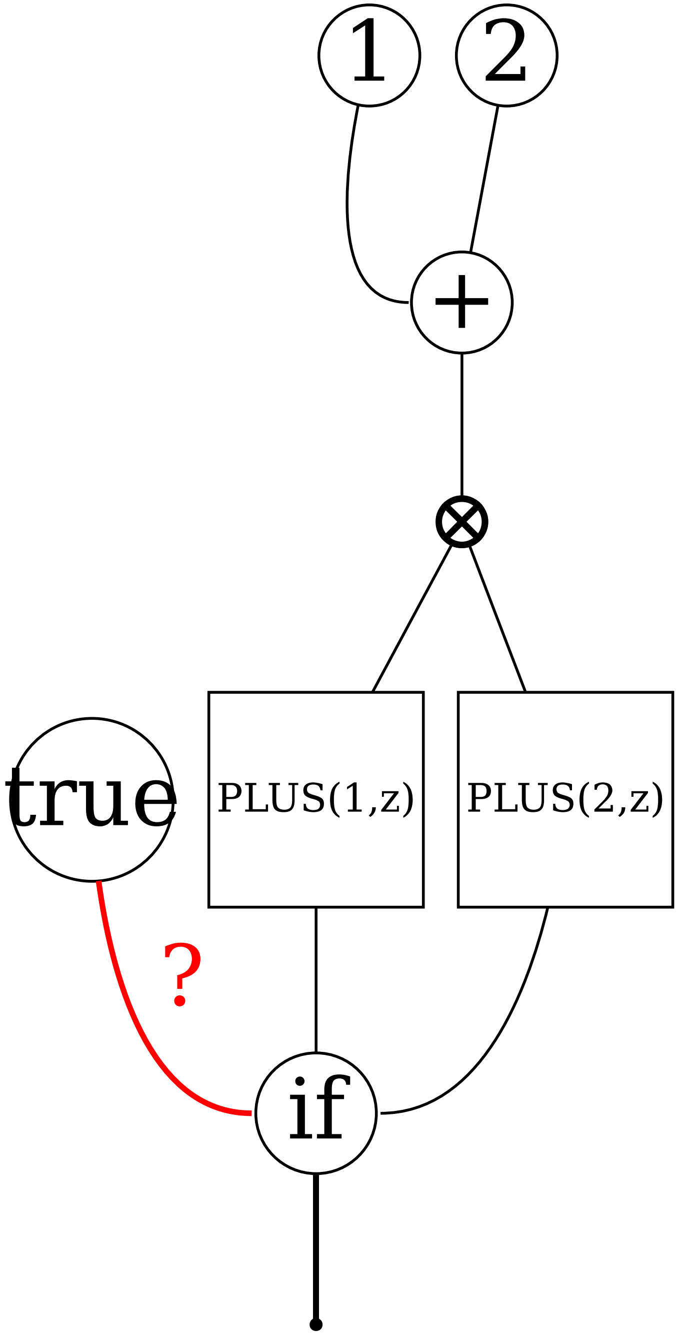
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



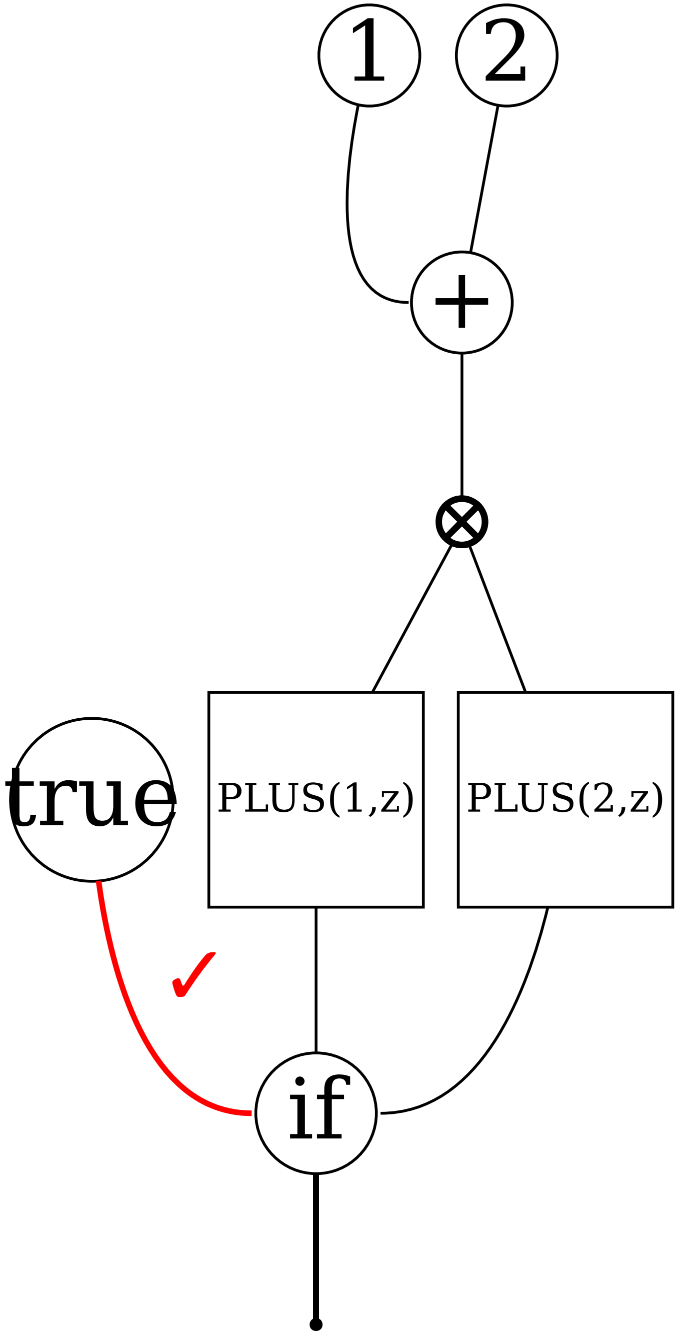
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



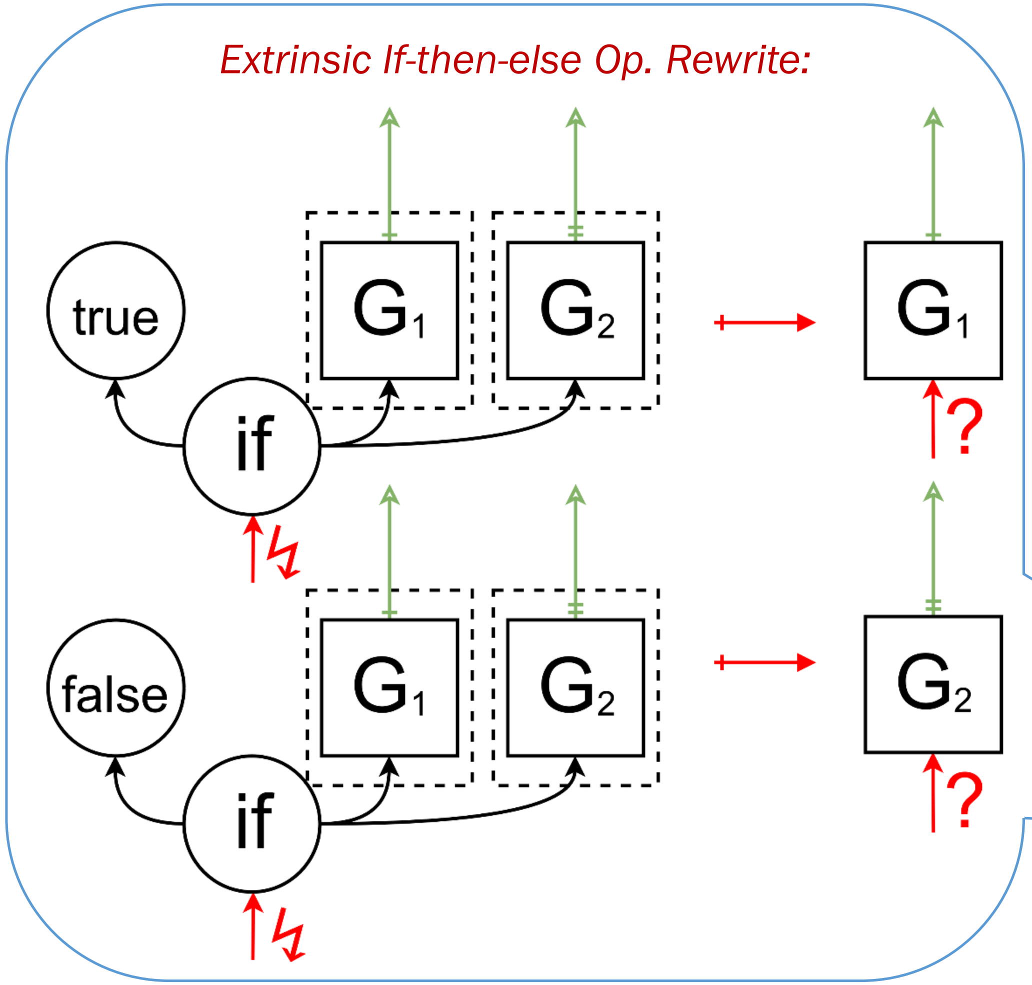
SPARTAN Semantics

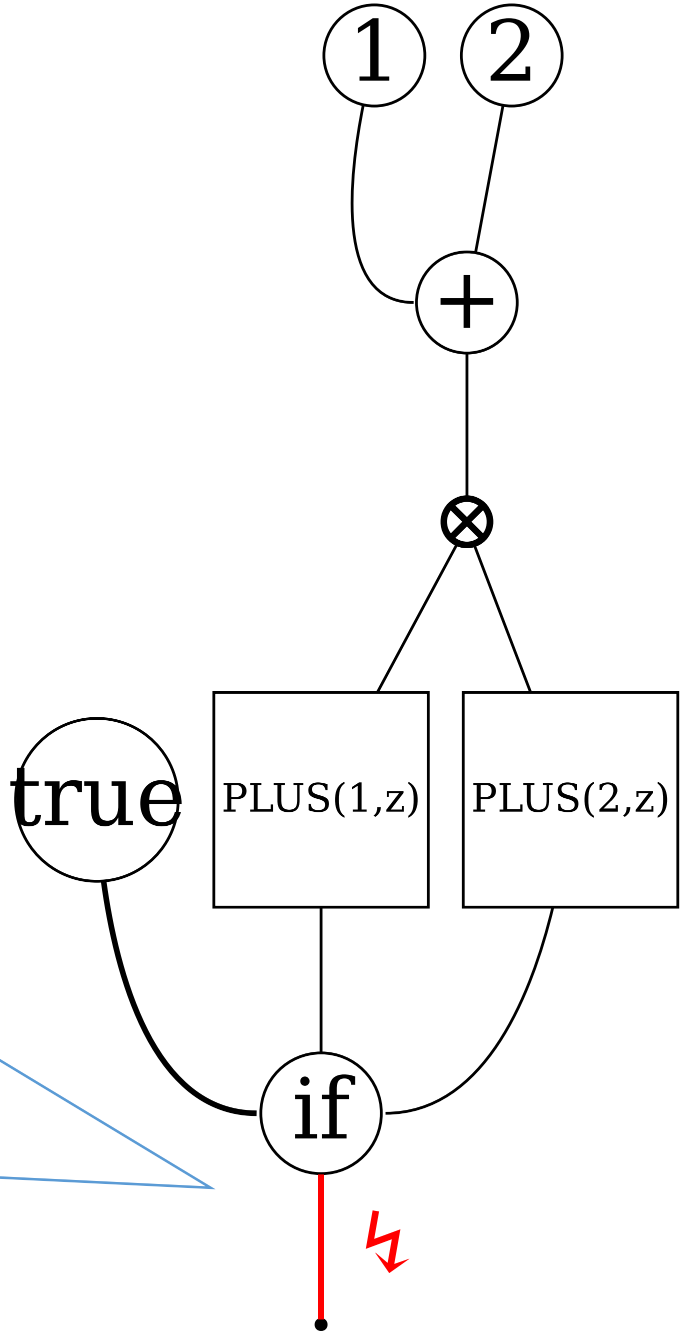
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

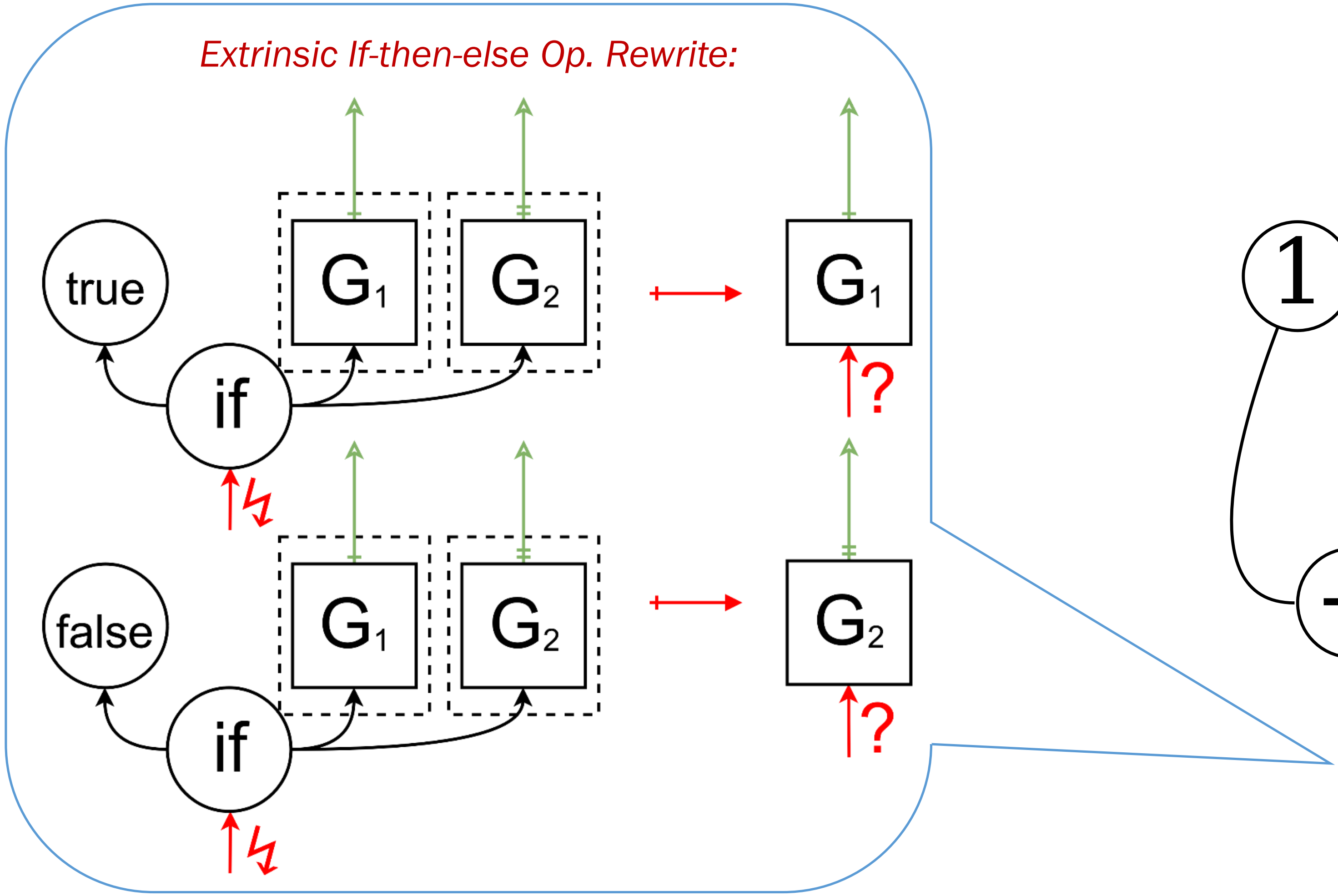
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$

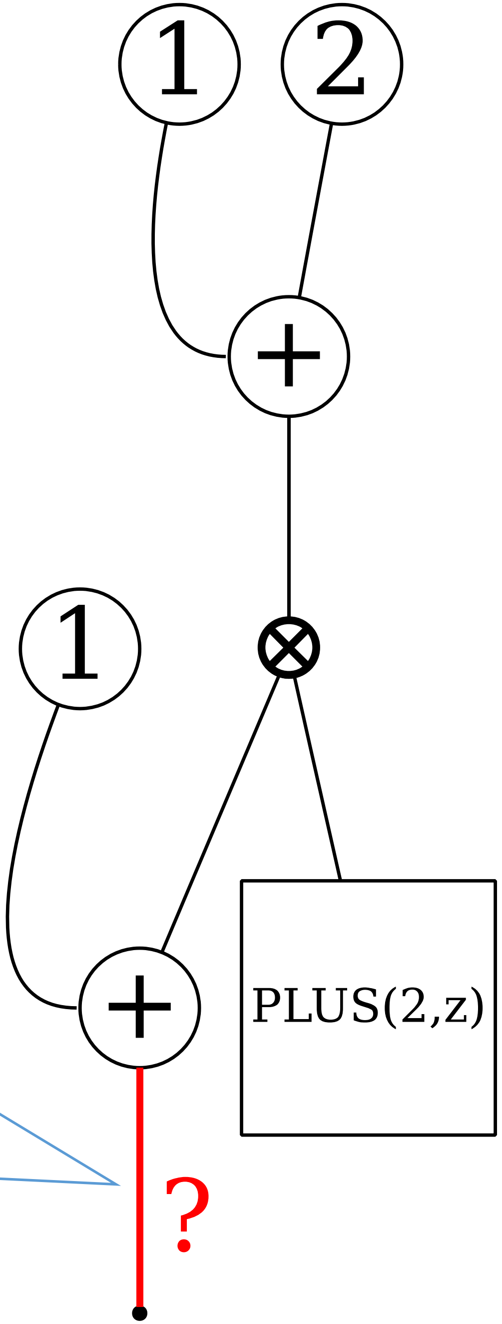




SPARTAN Semantics

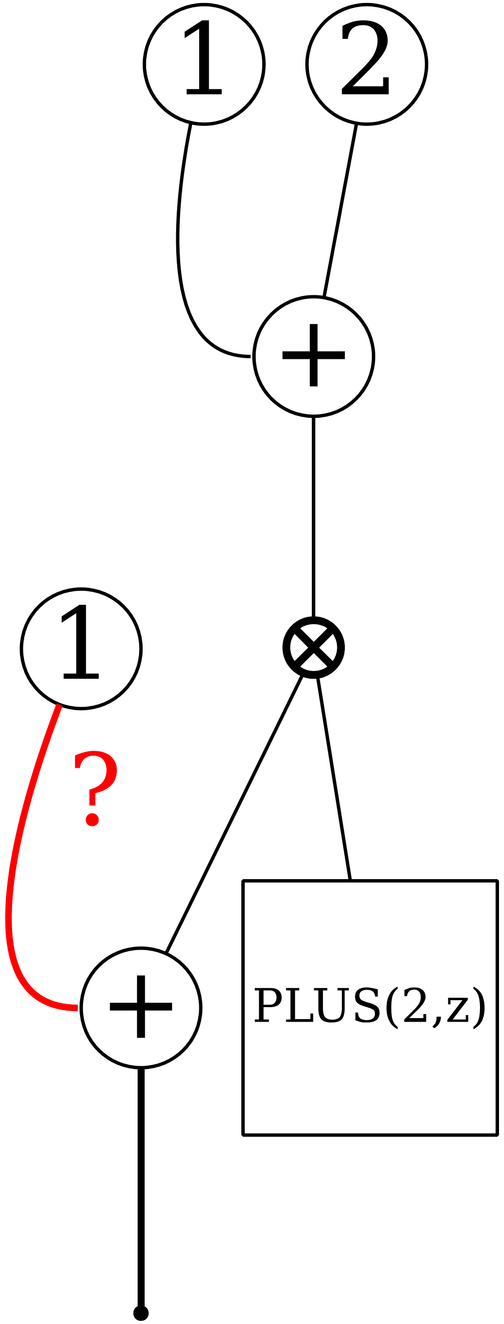
$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$





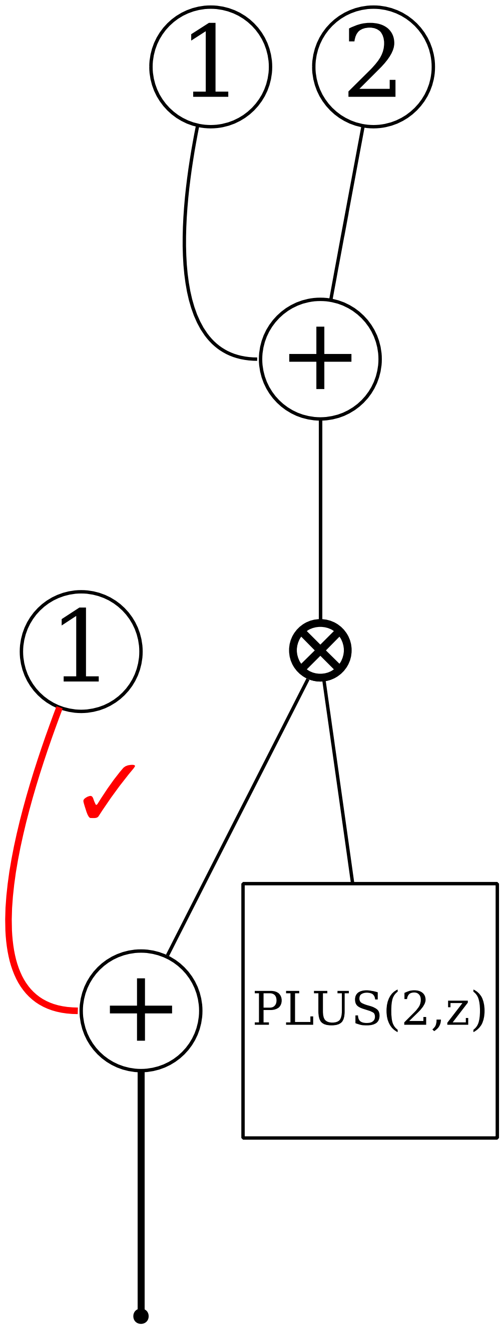
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



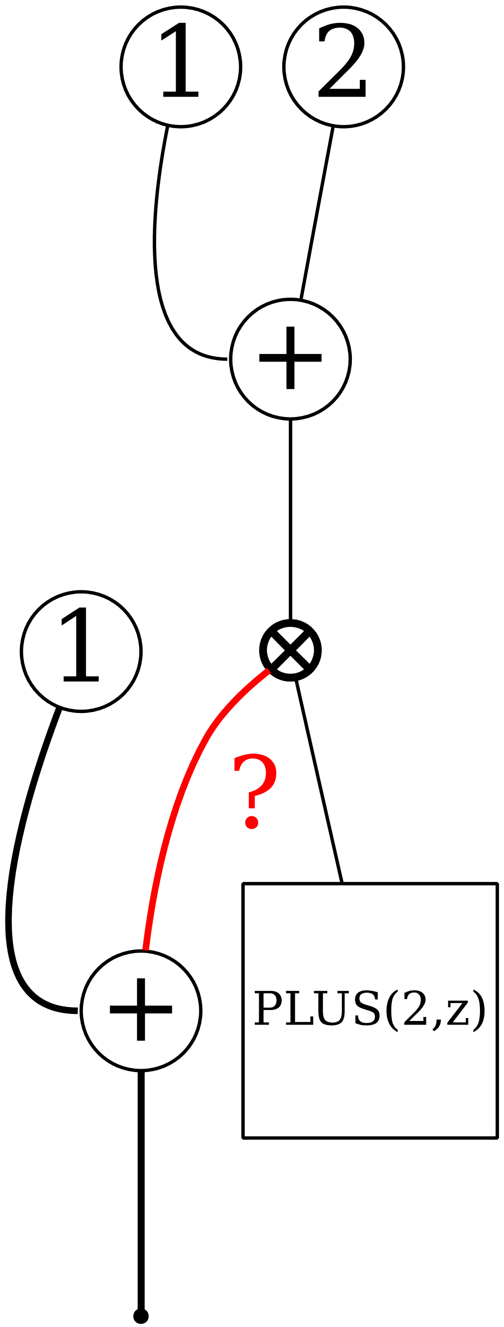
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



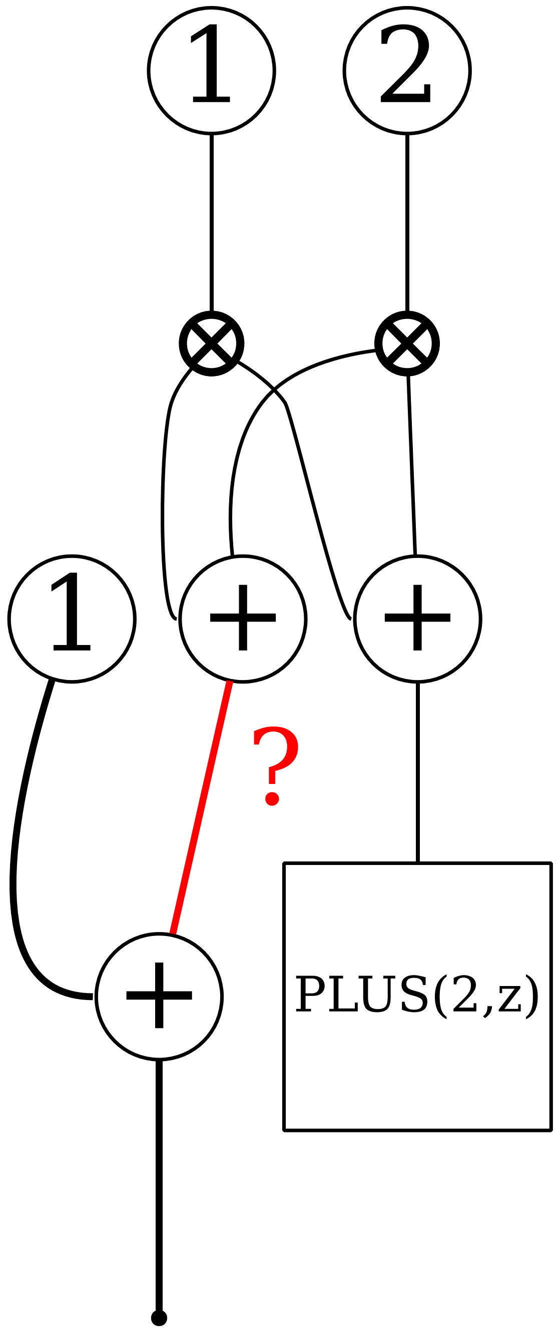
Spartan Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



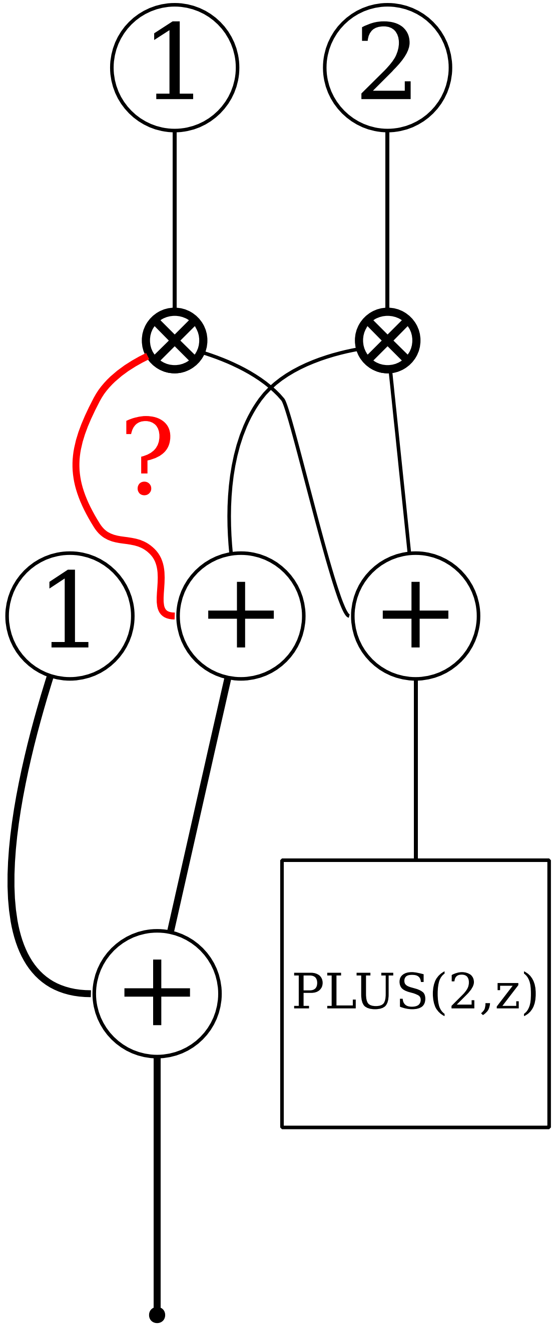
Spartan Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



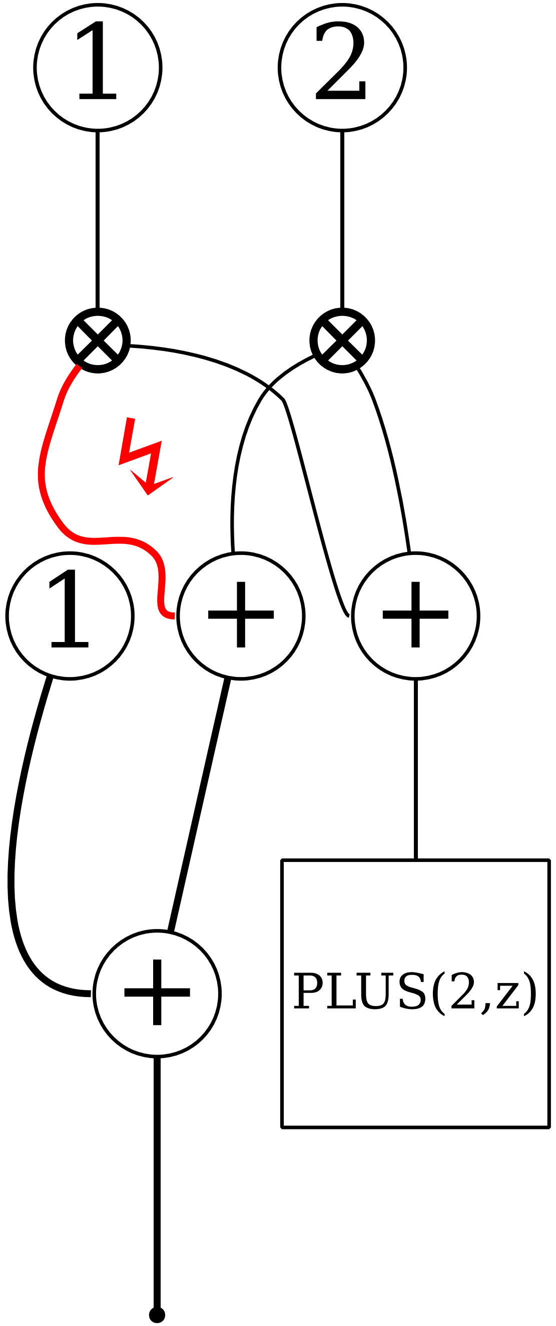
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



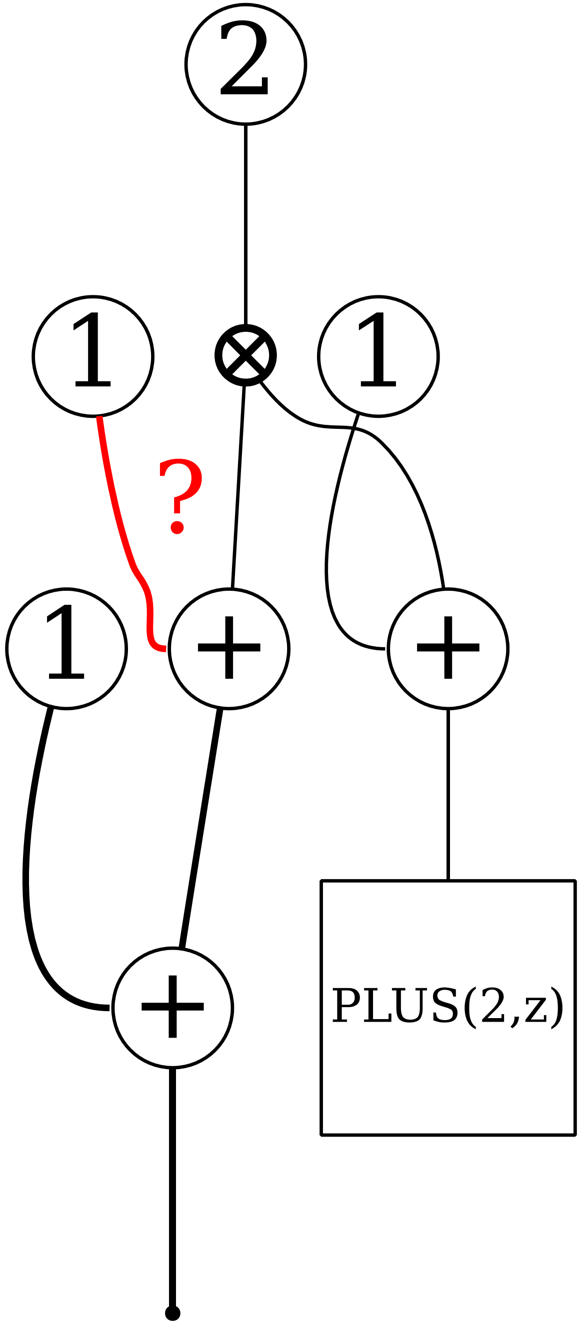
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



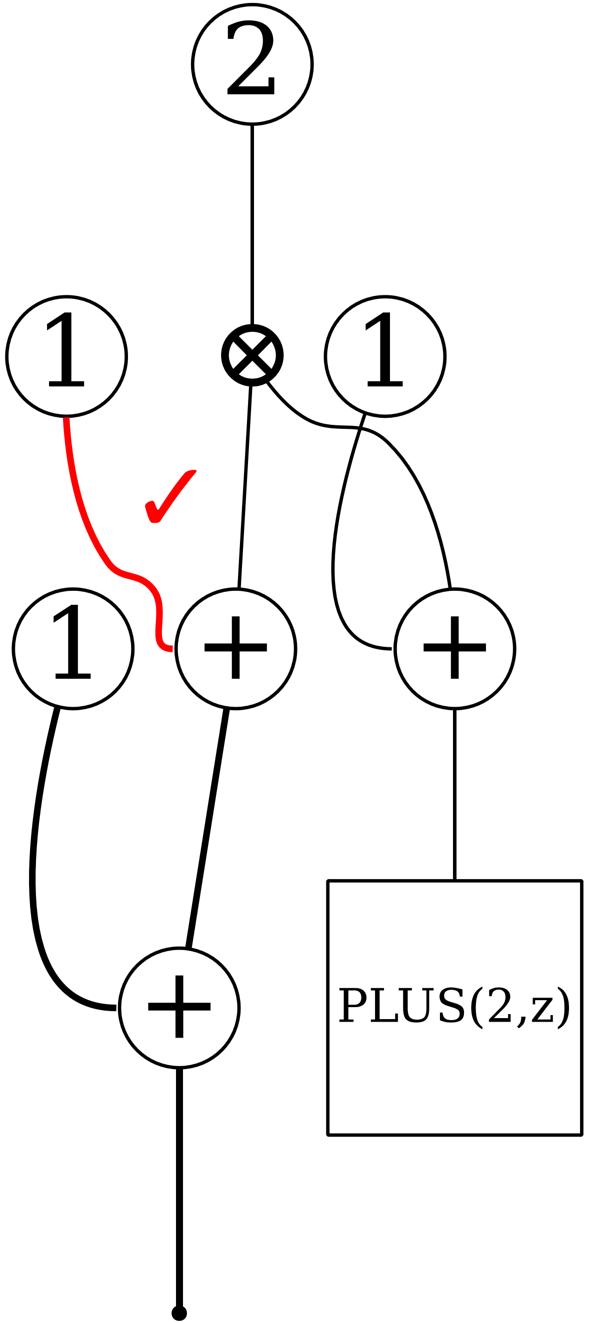
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



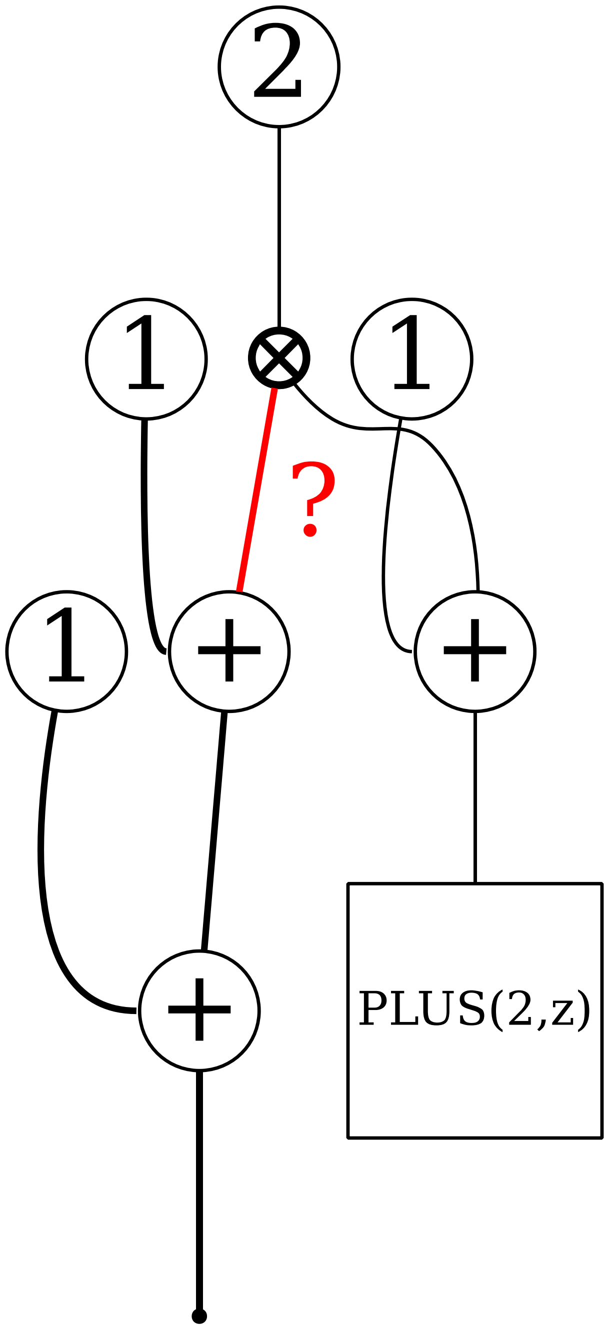
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



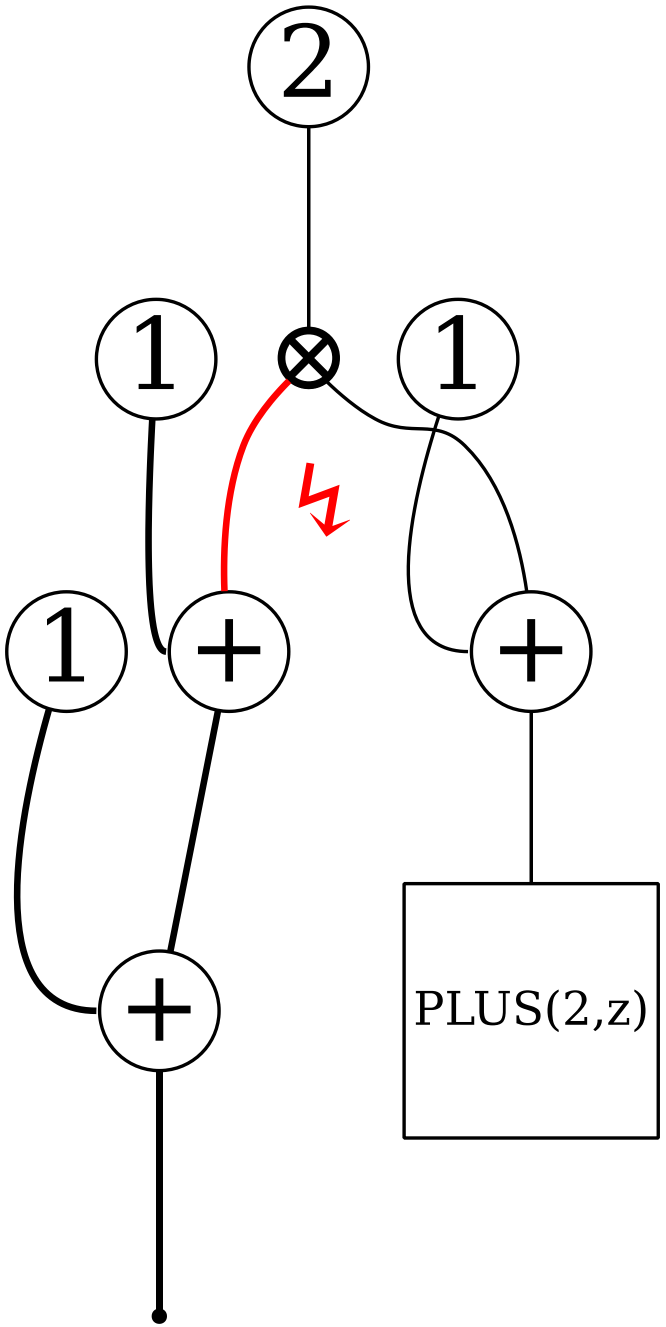
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



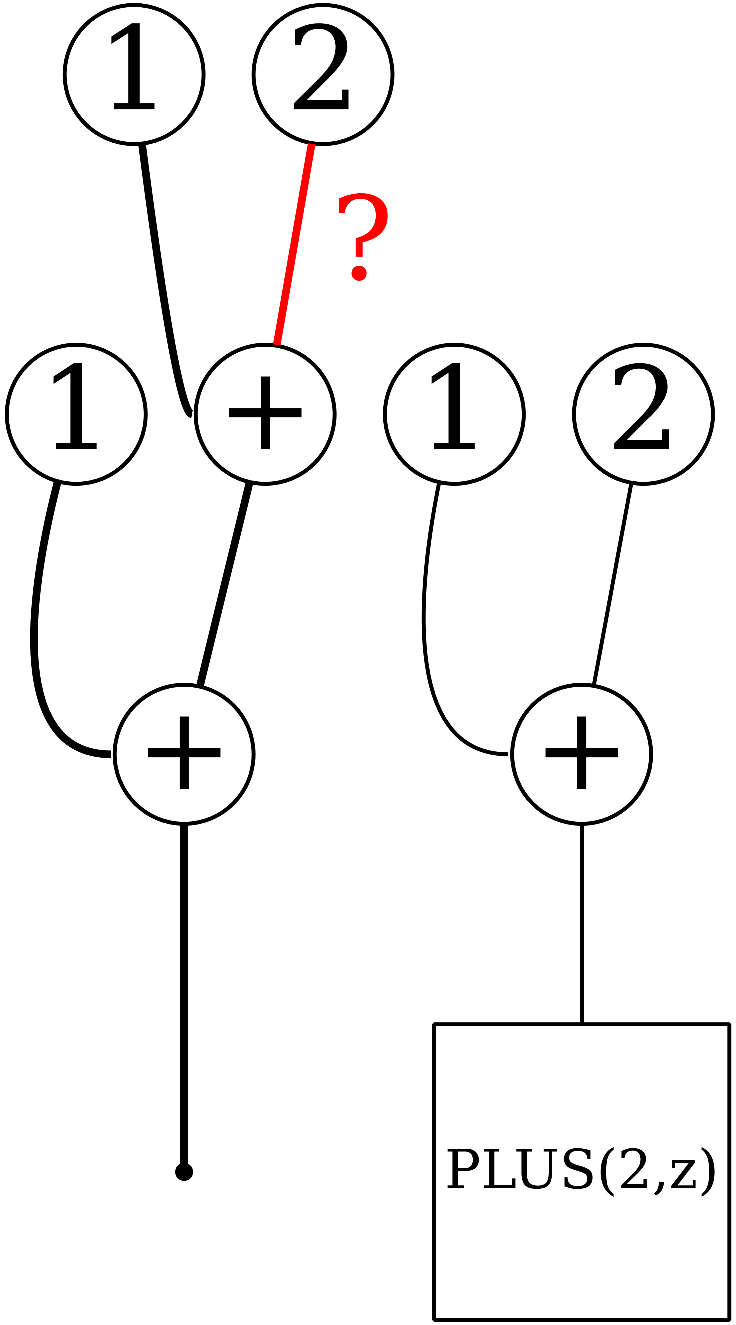
SPARTAN Semantics

$bind\ z \to 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



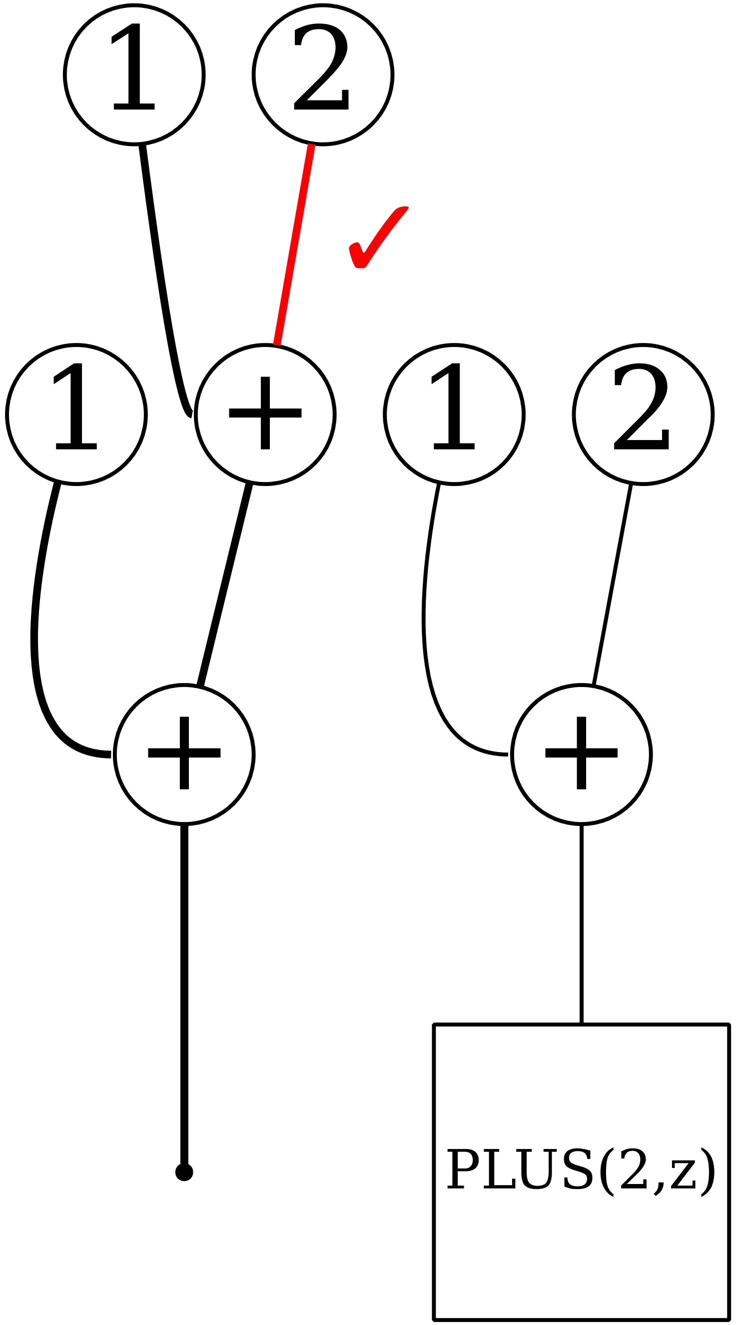
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



Spartan Semantics

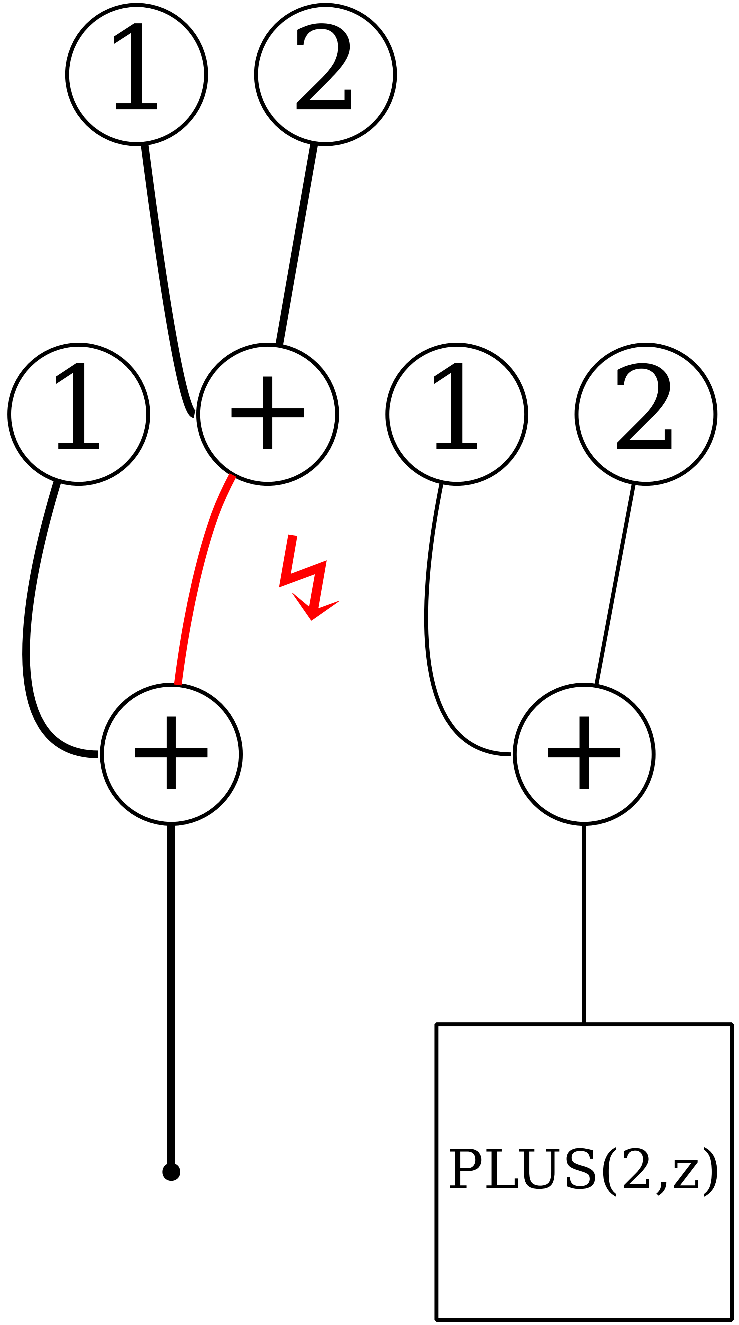
$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



SPARTAN Semantics

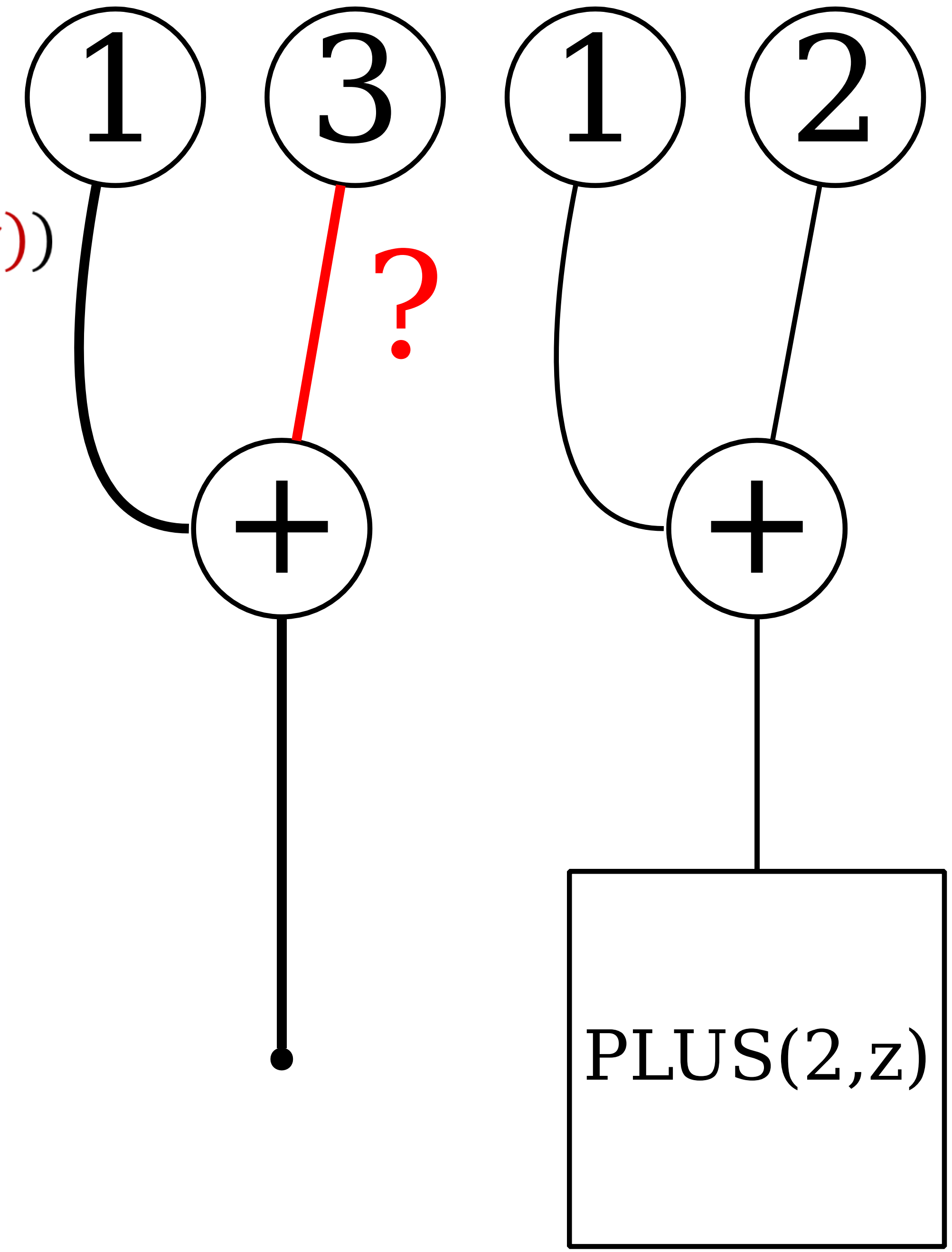
$bind\ z \to 1 + 2\ in$

$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



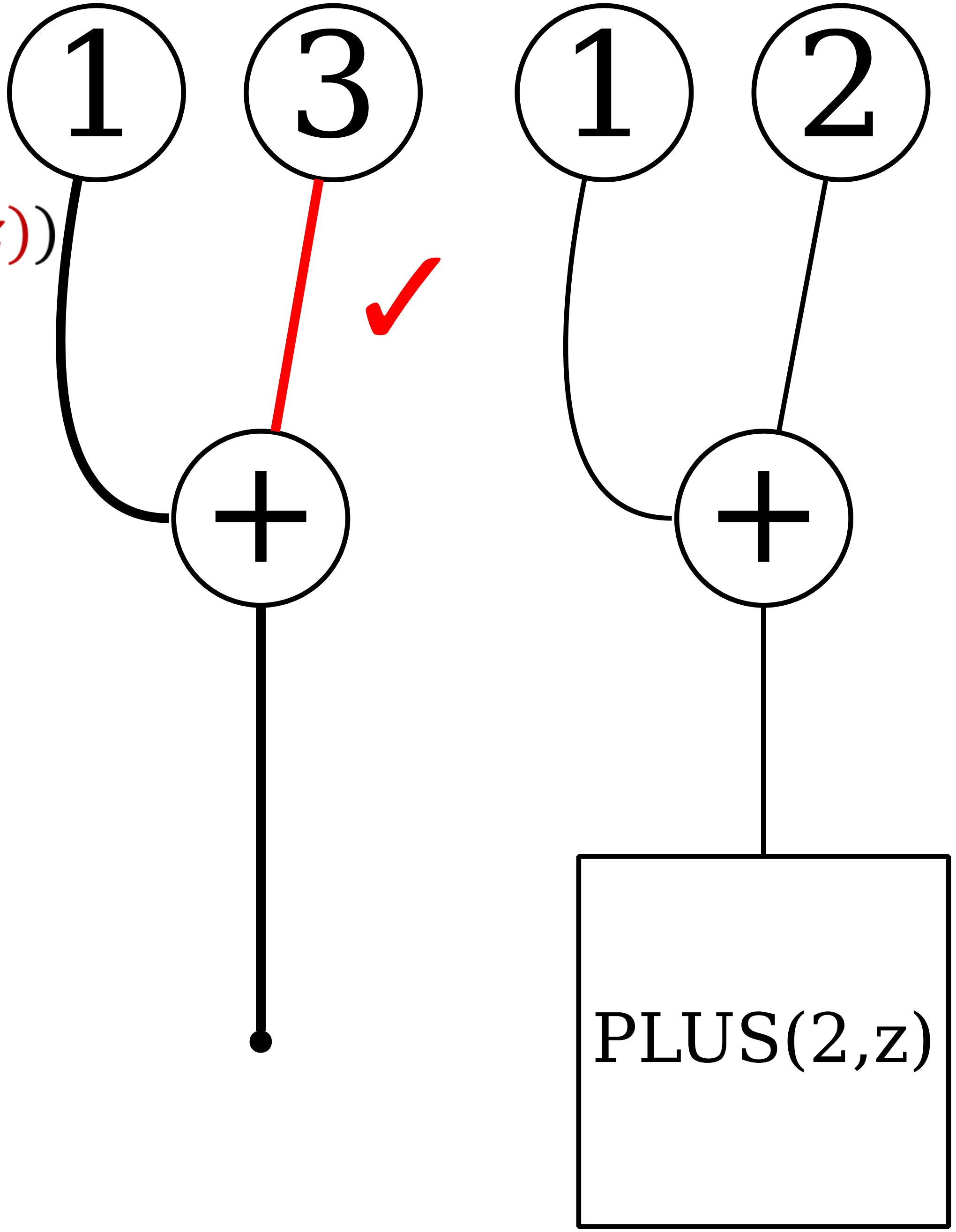
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



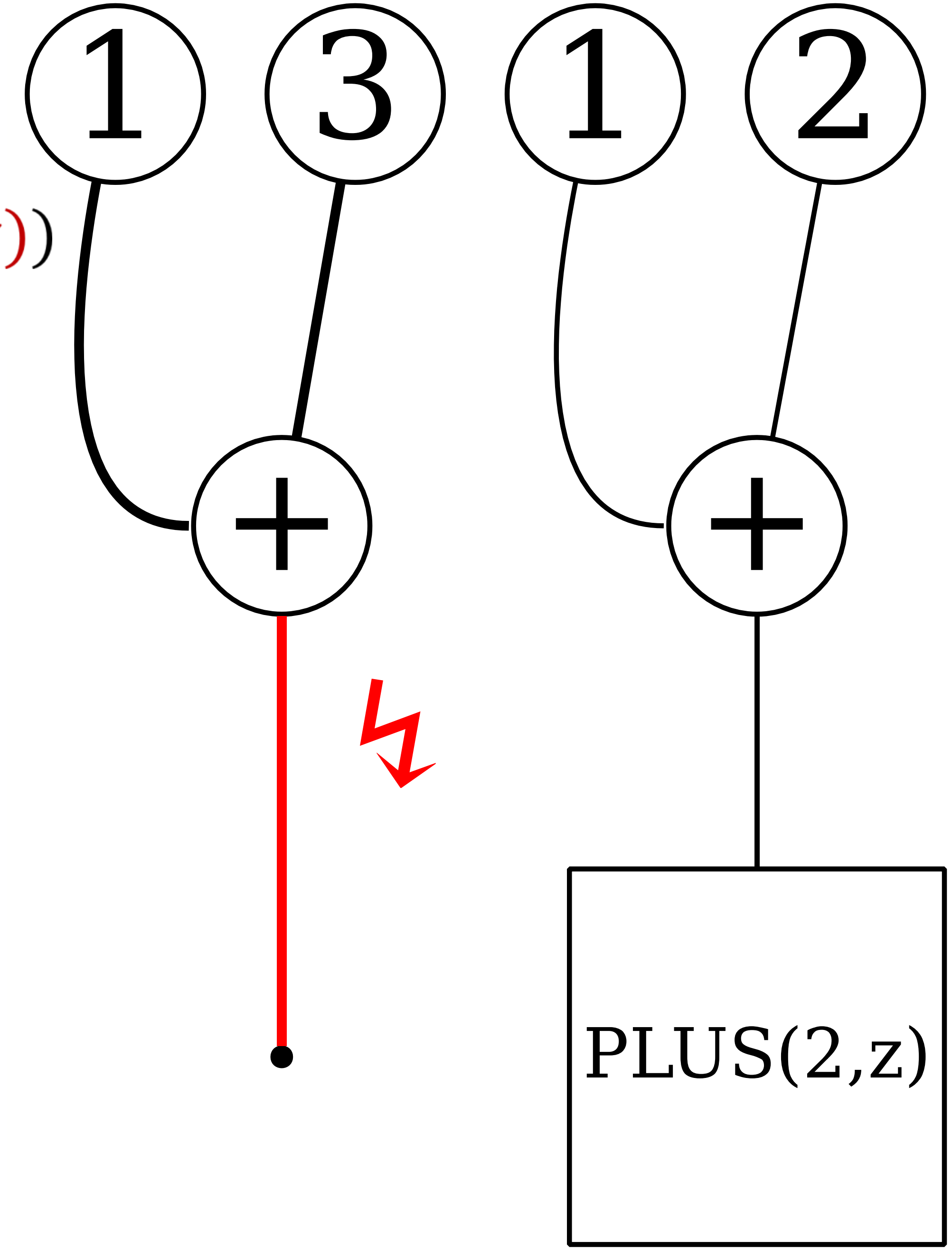
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



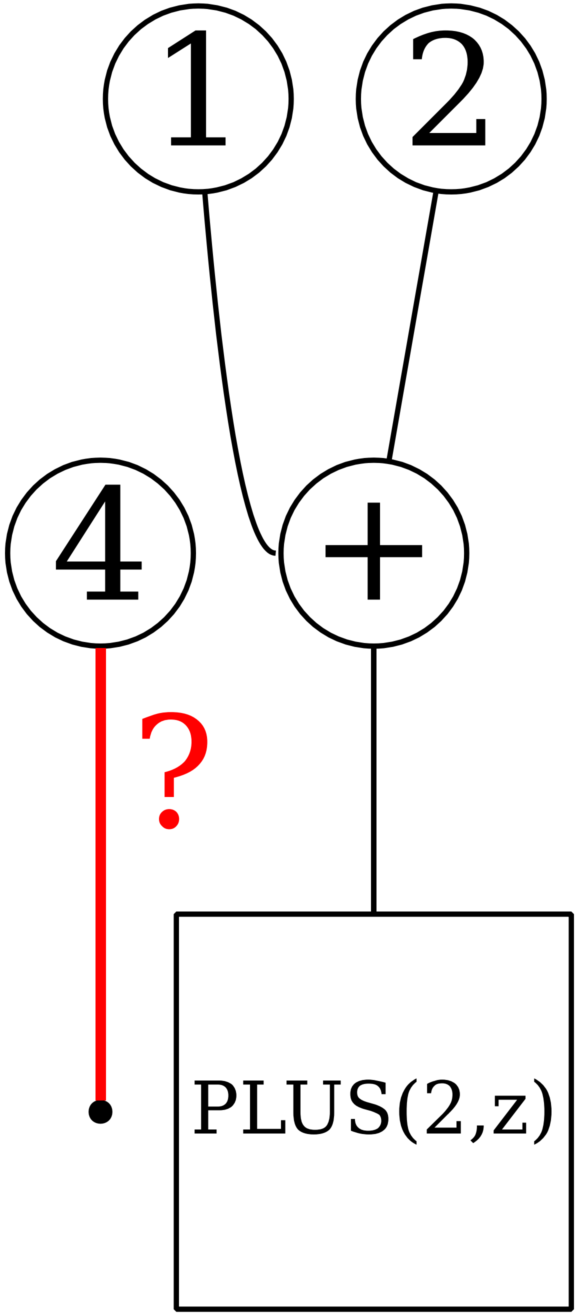
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



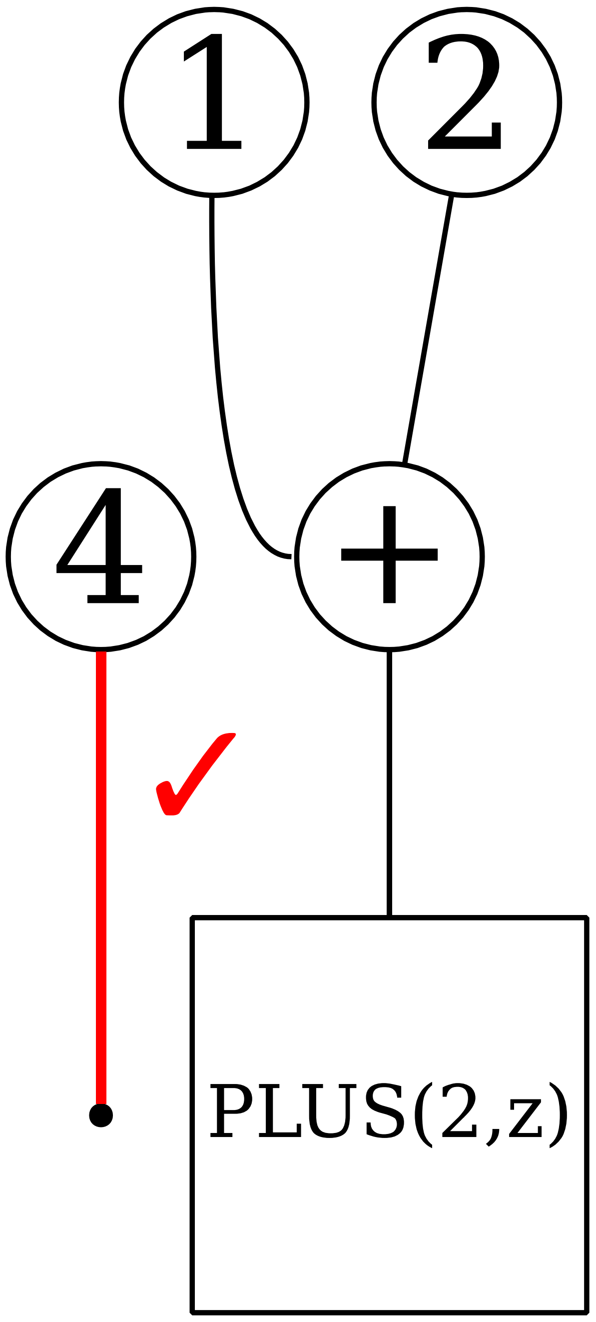
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



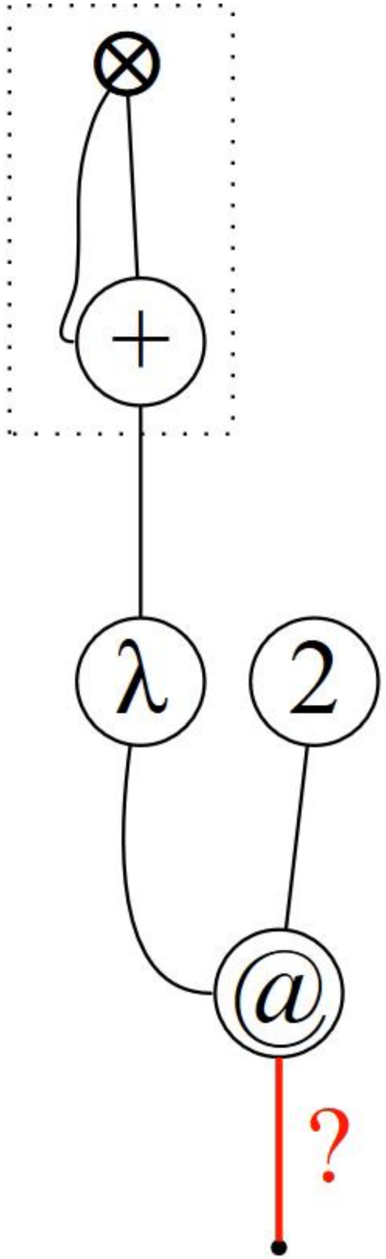
SPARTAN Semantics

$bind\ z \rightarrow 1 + 2\ in$
$(if\ (z == 3)\ then\ (1 + z)\ else\ (2 + z))$



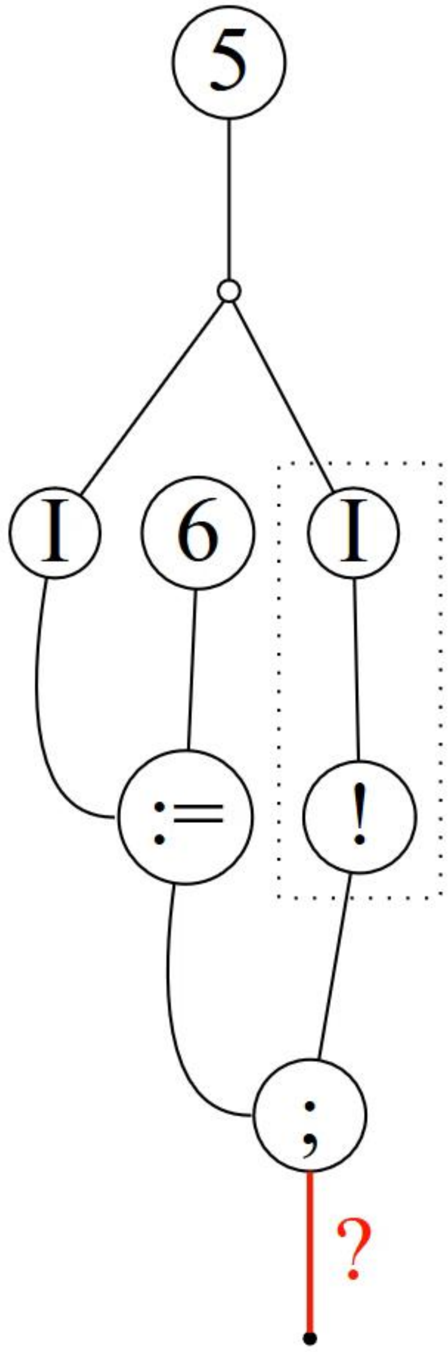
SPARTAN Semantics

$$(\lambda x.\, x + x)\ 2$$



SPARTAN Semantics

$$\begin{array}{l} \textit{new } a \to 5 \textit{ in} \\ \quad a := 6; \\ \quad !\,a \end{array}$$



The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and *refocusing*

The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and *refocusing*

Recall: Terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[t_1]) \simeq Init(C^{bf}[t_2])$$

The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and *refocusing*

Recall: Terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[t_1]) \simeq Init(C^{bf}[t_2])$$

Terms ($\boldsymbol{t_1}, \boldsymbol{t_2}$): $t \coloneqq x \mid bind\ x \to t'\ in\ t'' \mid a \mid new\ a \multimap t'\ in\ t'' \mid \vec{y}.t' \mid \phi(\overrightarrow{t'}; \overrightarrow{t''})$

The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and *refocusing*

Recall: Terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[t_1]) \simeq Init(C^{bf}[t_2])$$

Terms ($\boldsymbol{t_1}, \boldsymbol{t_2}$): $t \coloneqq x \mid bind\ x \to t'\ in\ t'' \mid a \mid new\ a \multimap t'\ in\ t'' \mid \vec{y}.t' \mid \phi(\overrightarrow{t'}; \overrightarrow{t''})$

All contexts ($\forall \boldsymbol{C}$): $C^{bf} \coloneqq \square \mid bind\ x \to t\ in\ C' \mid new\ a \multimap t\ in\ C' \mid \vec{y}.C' \mid$
$\phi(\vec{t}, C', \overrightarrow{t'}; \overrightarrow{t''}) \mid \phi(\vec{t}; \overrightarrow{t'}, C', \overrightarrow{t''})$

The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and *refocusing*

Recall: Terms t_1, t_2 are equivalent if they are *observationally equivalent*:

$$t_1 \equiv t_2 \Leftrightarrow \forall C^{bf}. Init(C^{bf}[t_1]) \simeq Init(C^{bf}[t_2])$$

Terms ($\boldsymbol{t_1, t_2}$): $t \coloneqq x \mid bind\ x \to t'\ in\ t'' \mid a \mid new\ a \multimap t'\ in\ t'' \mid \vec{y}.t' \mid \phi(\overrightarrow{t'}; \overrightarrow{t''})$

All contexts ($\forall \boldsymbol{C}$): $C^{bf} \coloneqq \square \mid bind\ x \to t\ in\ C' \mid new\ a \multimap t\ in\ C' \mid \vec{y}.C' \mid$
$\phi(\vec{t}, C', \overrightarrow{t'}; \overrightarrow{t''}) \mid \phi(\vec{t}; \overrightarrow{t'}, C', \overrightarrow{t''})$

Equivalence of output behaviour ($\simeq$): Untyped setting!
Terms have the same output behaviour if their initial states are *state equivalent*:

$\dot{G}_1 \simeq \dot{G}_2 \Leftrightarrow \dot{G}_1 \to^* \dot{N}_1$ (final state) if and only if $\dot{G}_2 \to^* \dot{N}_2$ (final state)

The Characterisation Theorem

SPARTAN graphs g_1, g_2 are equivalent if their initial states are *state equivalent*:

$$g_1 \equiv g_2 \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[g_1]) \simeq Init(C^{bf}[g_2])$$

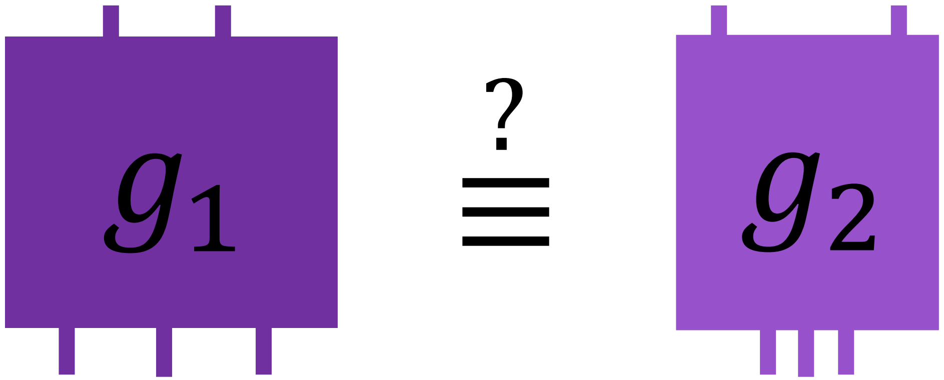
Contexts in the hypernet model are more expressive than those in the term model

Can immediately identify relevant direct interactions arising in a term

Thus, there are not hidden interactions between the program and context

This leads to a notion of *local reasoning* about programs

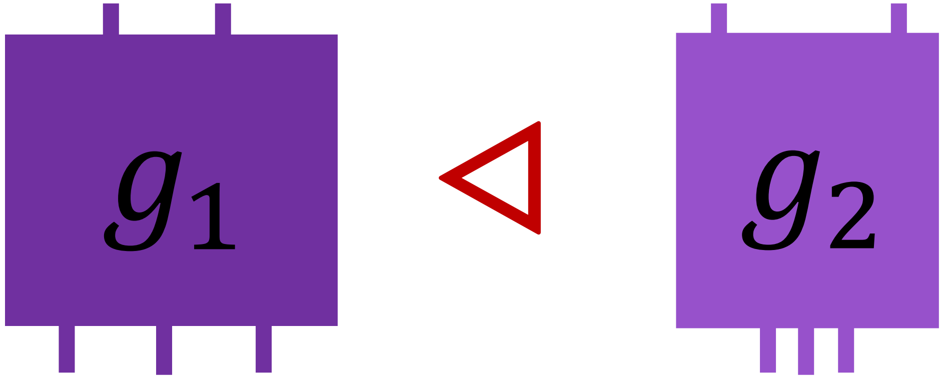
The Characterisation Theorem



$$g_1 \equiv g_2 \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[g_1]) \simeq Init(C^{bf}[g_2])$$

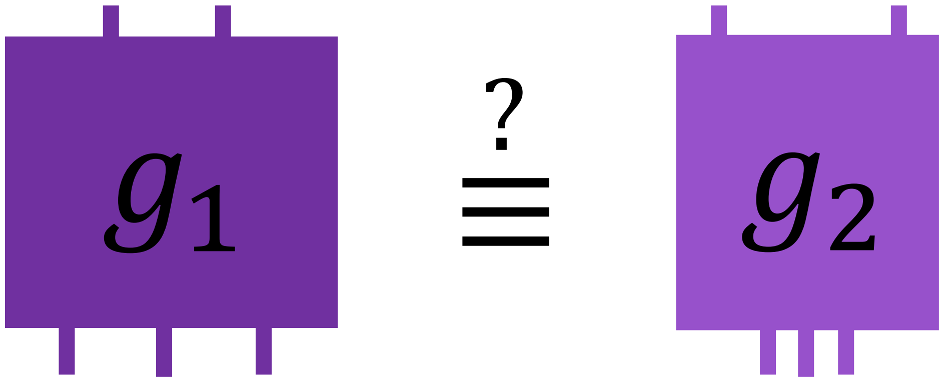
The Characterisation Theorem

Pre-template: A family of binary relations on hypernets with the same interface



$$g_1 \equiv g_2 \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[g_1]) \simeq Init(C^{bf}[g_2])$$

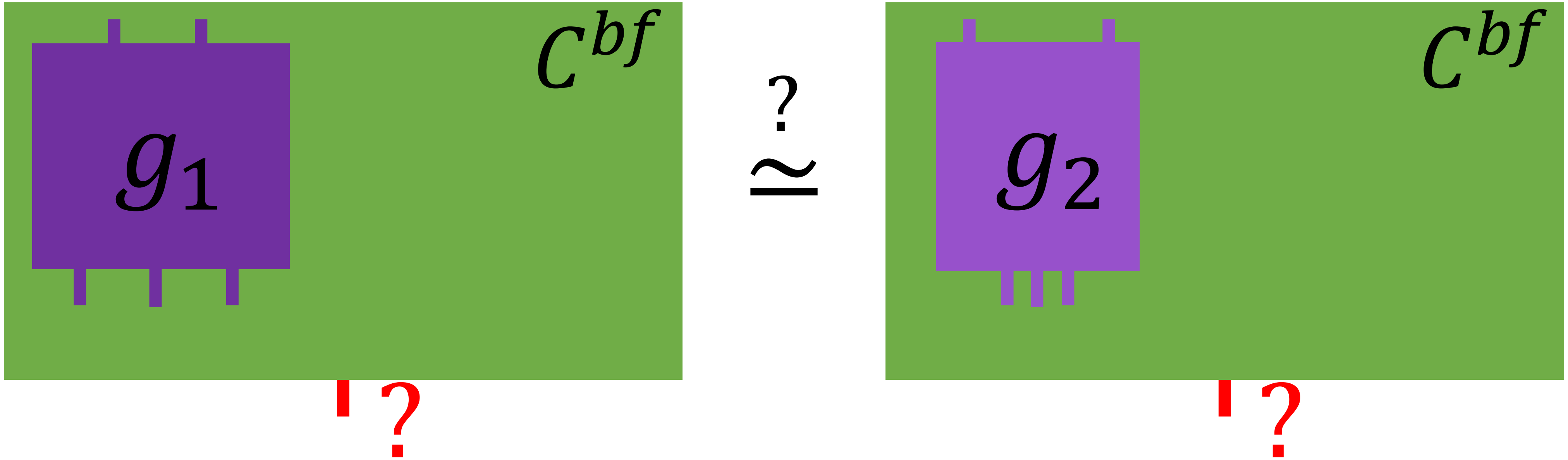
The Characterisation Theorem



$$\boldsymbol{g_1} \equiv \boldsymbol{g_2} \Leftrightarrow \forall C^{bf}.\, Init(C^{bf}[g_1]) \simeq Init(C^{bf}[g_2])$$

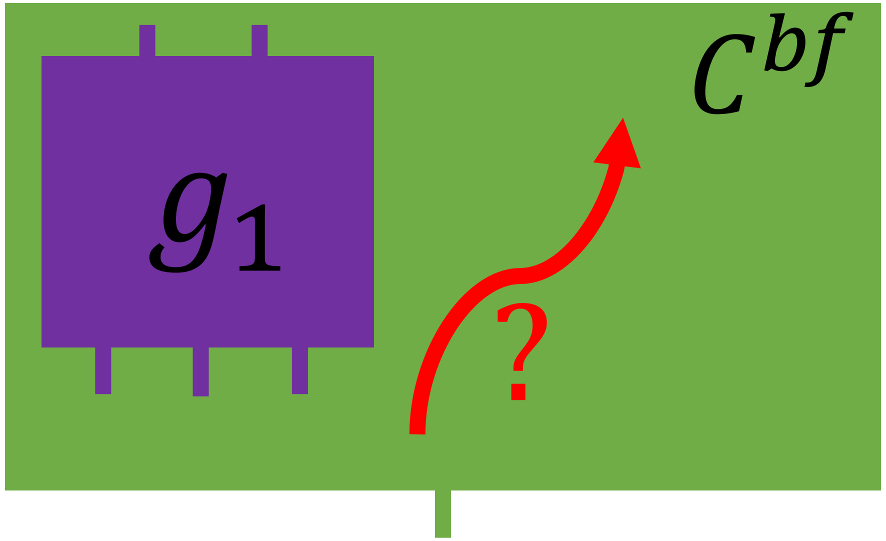
The Characterisation Theorem

$\forall C^{bf}.$

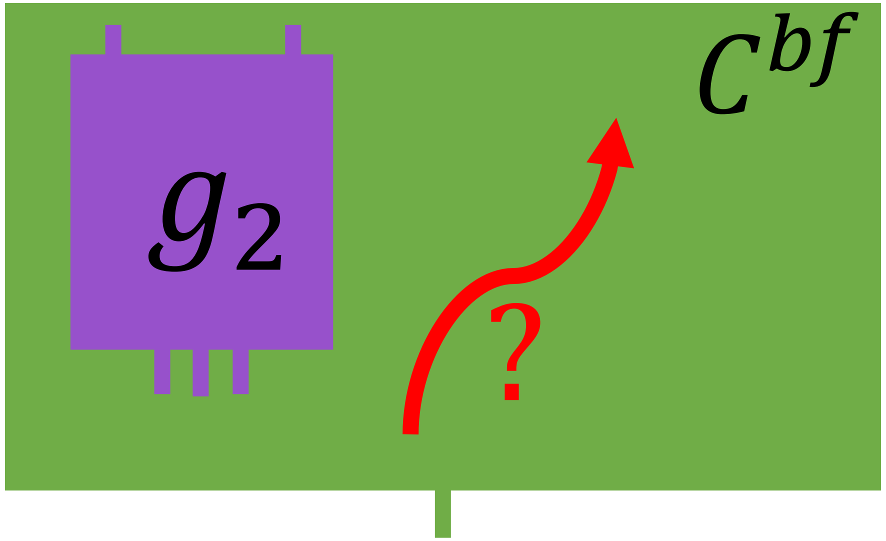


$g_1 \equiv g_2 \Leftrightarrow \forall C^{bf}.\, \boldsymbol{Init}(C^{bf}[g_1]) \simeq \boldsymbol{Init}(C^{bf}[g_2])$

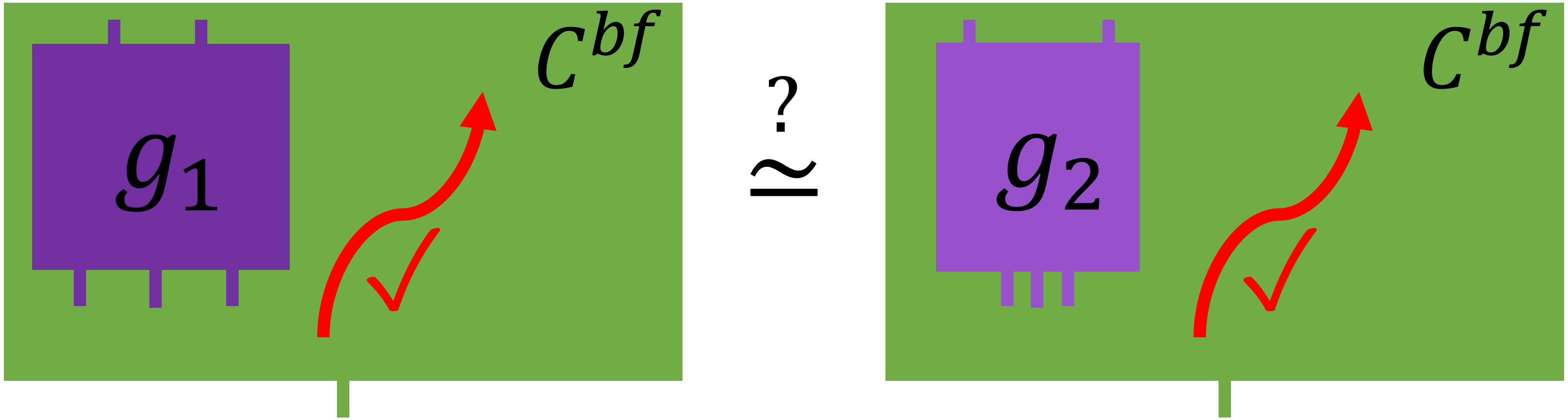
The Characterisation Theorem



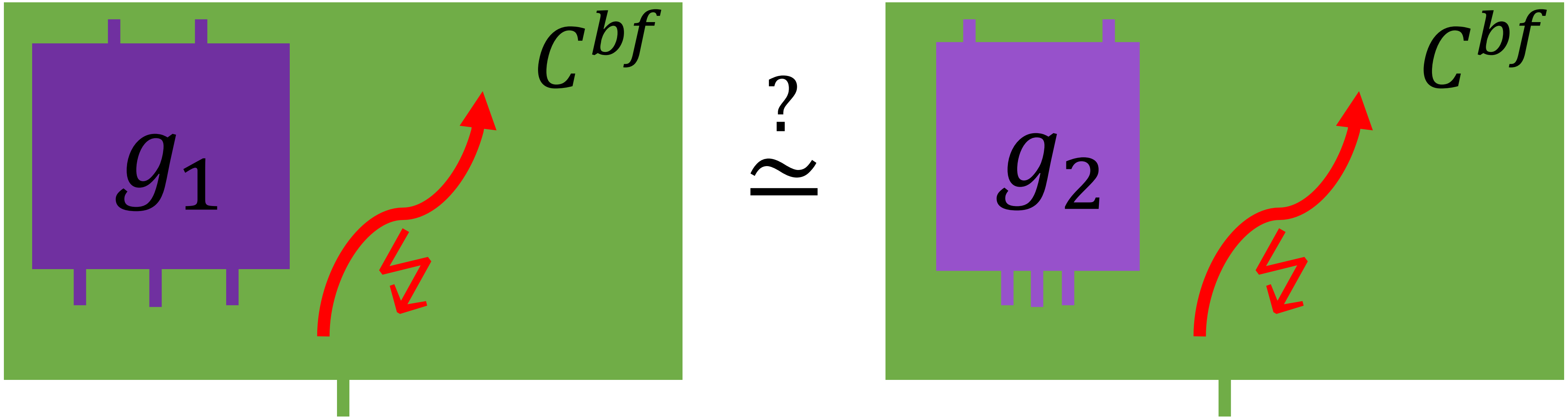
$\stackrel{?}{\simeq}$



The Characterisation Theorem

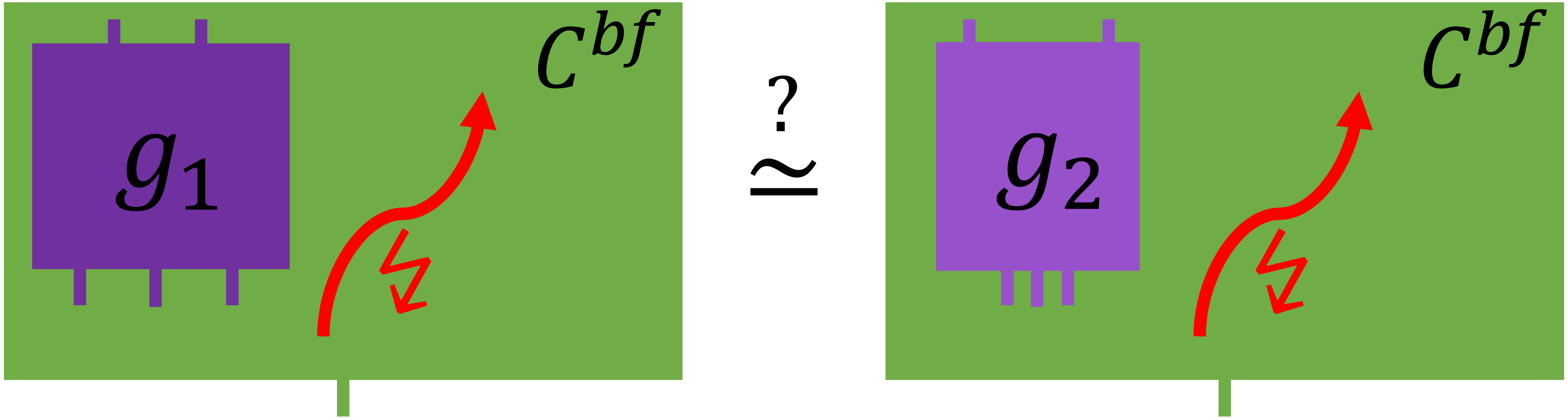


The Characterisation Theorem

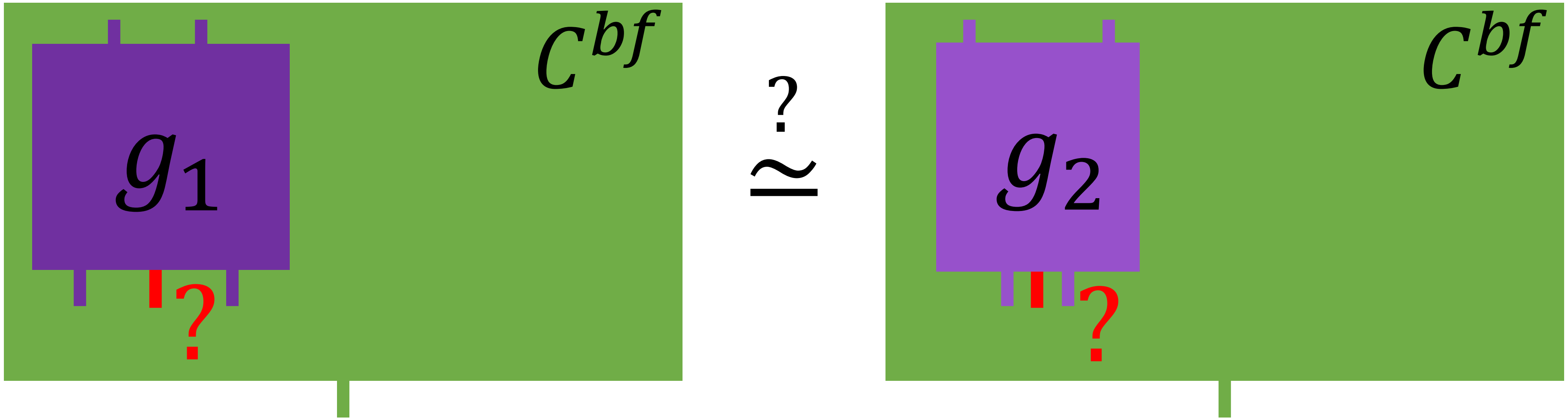


The Characterisation Theorem

Robust: A pre-template that is preserved by any rewrite in the context

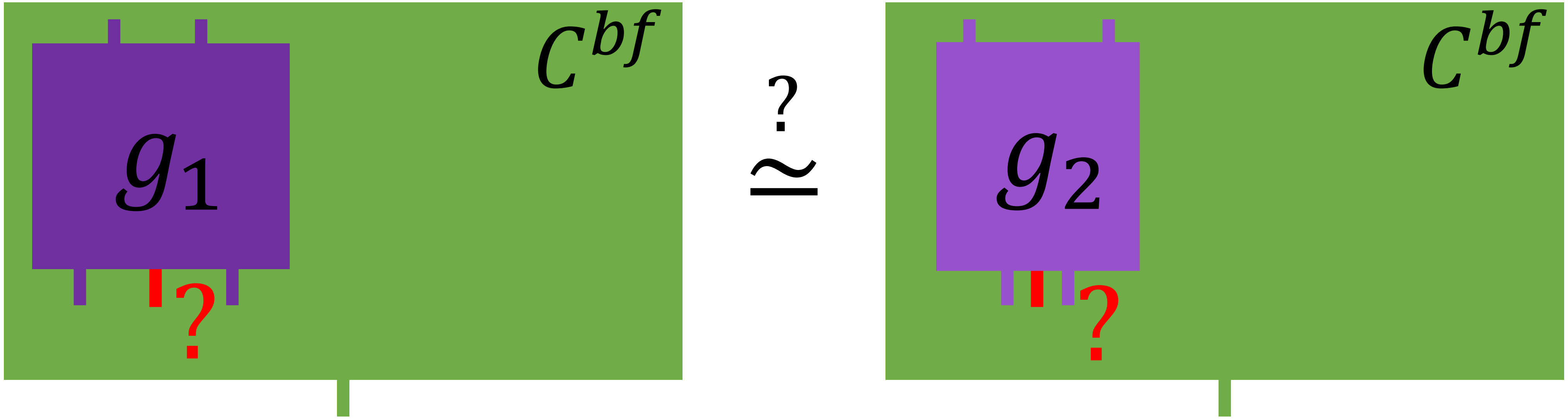


The Characterisation Theorem

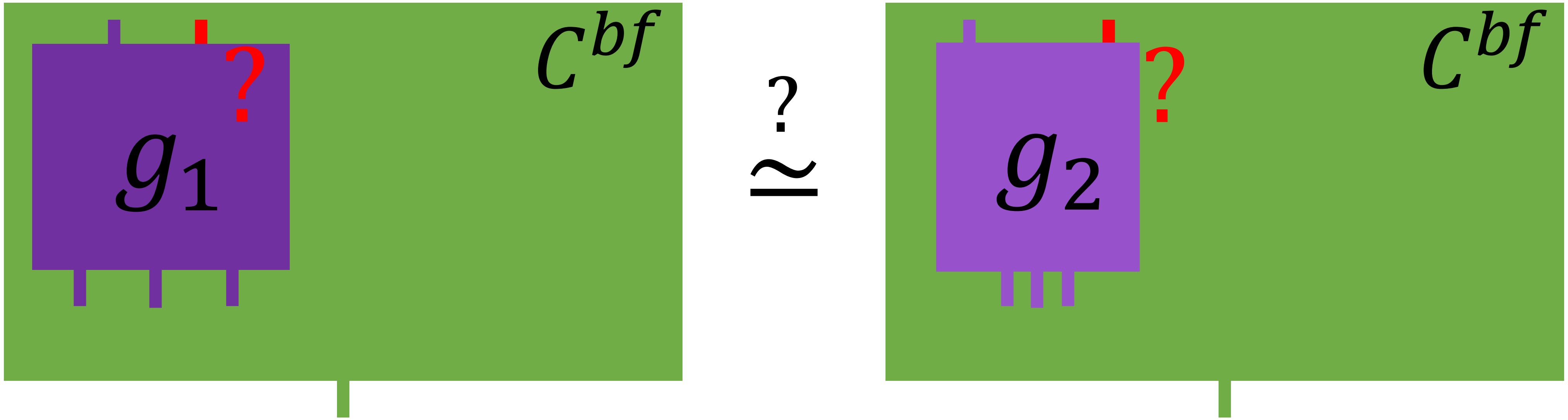


The Characterisation Theorem

Input-safe: A pre-template that is preserved by any input search token

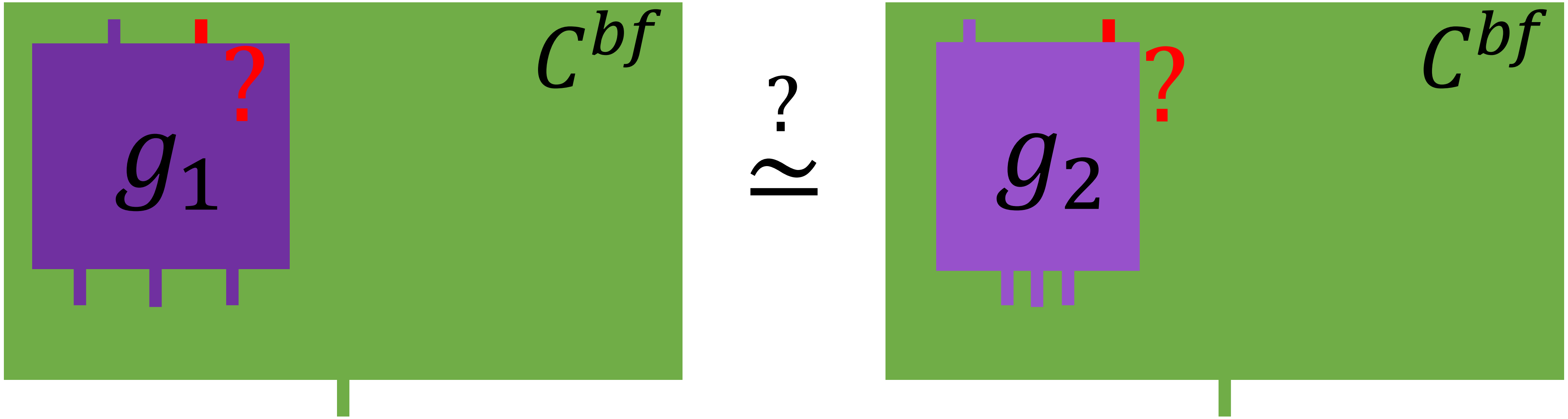


The Characterisation Theorem



The Characterisation Theorem

Output-closed: A pre-template where the token cannot reach an output from the initial state



The Characterisation Theorem

A *pre-template* is a family of binary relations on graphs with the same interface

A *template* is a pre-template that is both input-safe and output-closed

Input-safe: The pre-template is preserved by any input search token

Output-closed: The pre-template prevents any output search token

A (pre-)template is *robust* if it is preserved by any rewrite in any context

Characterisation Theorem.

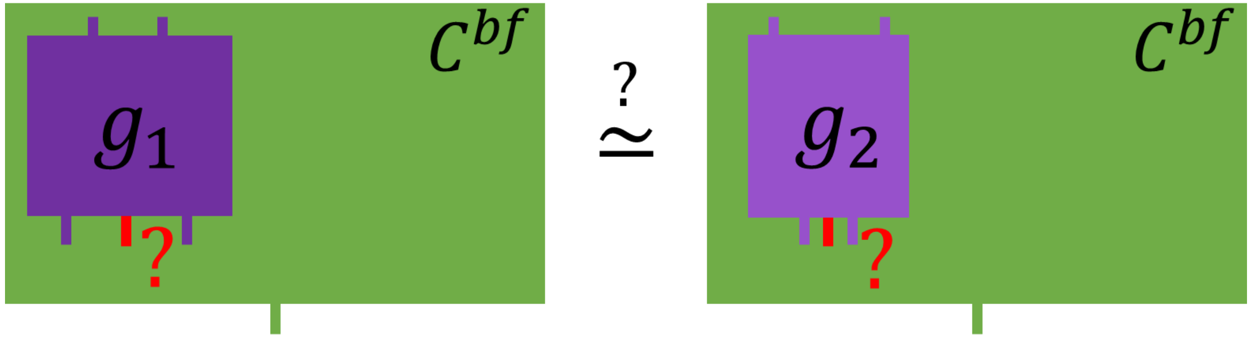
Robust templates induce observational equivalences

The Characterisation Theorem

Definition. A *pre-template* $g_1 \lhd g_2$ is *input-safe* if, for any valid *focussed context* $\dot{C}$ with an input search token, one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}_1$ and $\dot{C}[g_2] \to^* \dot{N}_2$
   for two stuck states $\dot{N}_1, \dot{N}_2$

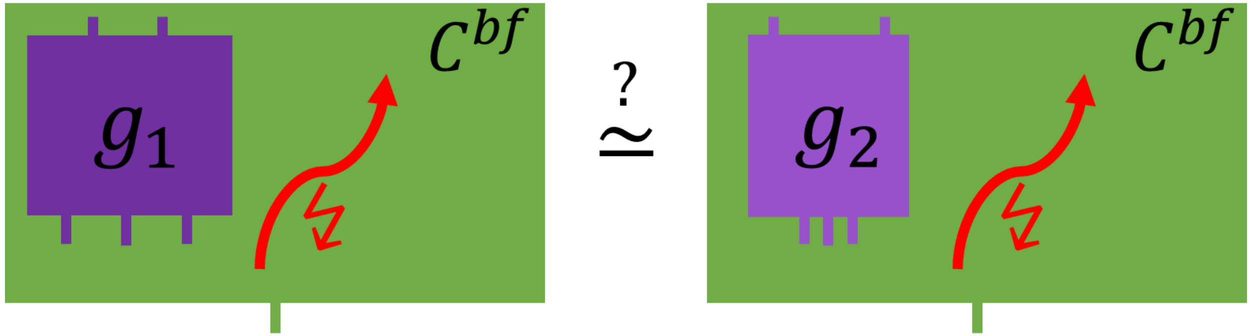
2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
   for two hypernets g_1', g_2' such that $g_1' \lhd g_2'$
   and a valid *focussed context* $\dot{C}'$
   such that the token is not in rewrite status



Definition. A *pre-template* $g_1 \lhd g_2$ is *robust* if, for any valid *focussed context* $\dot{C}$ with a rewrite token, one of the following holds:

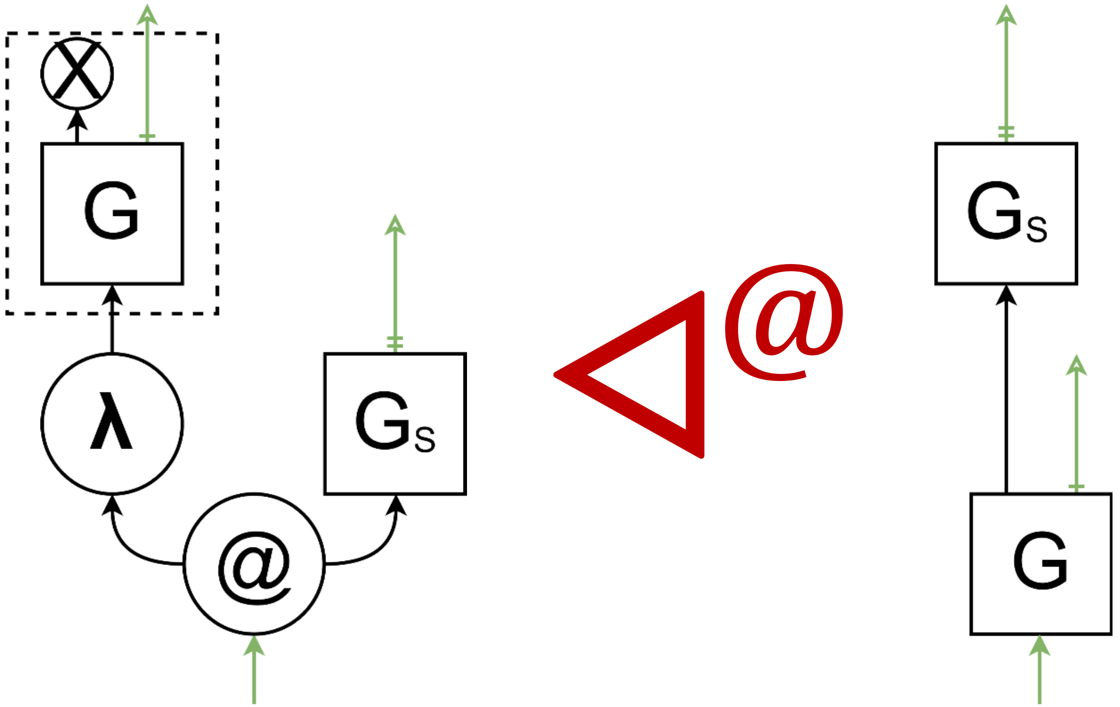
1. $\dot{C}[g_1] \to^+ \dot{N}_1$ and $\dot{C}[g_2] \to^* \dot{N}_2$
   for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^+ \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
   for two hypernets g_1', g_2' such that $g_1' \lhd g_2'$
   and a valid *focussed context* $\dot{C}'$
   such that the token is not in rewrite status



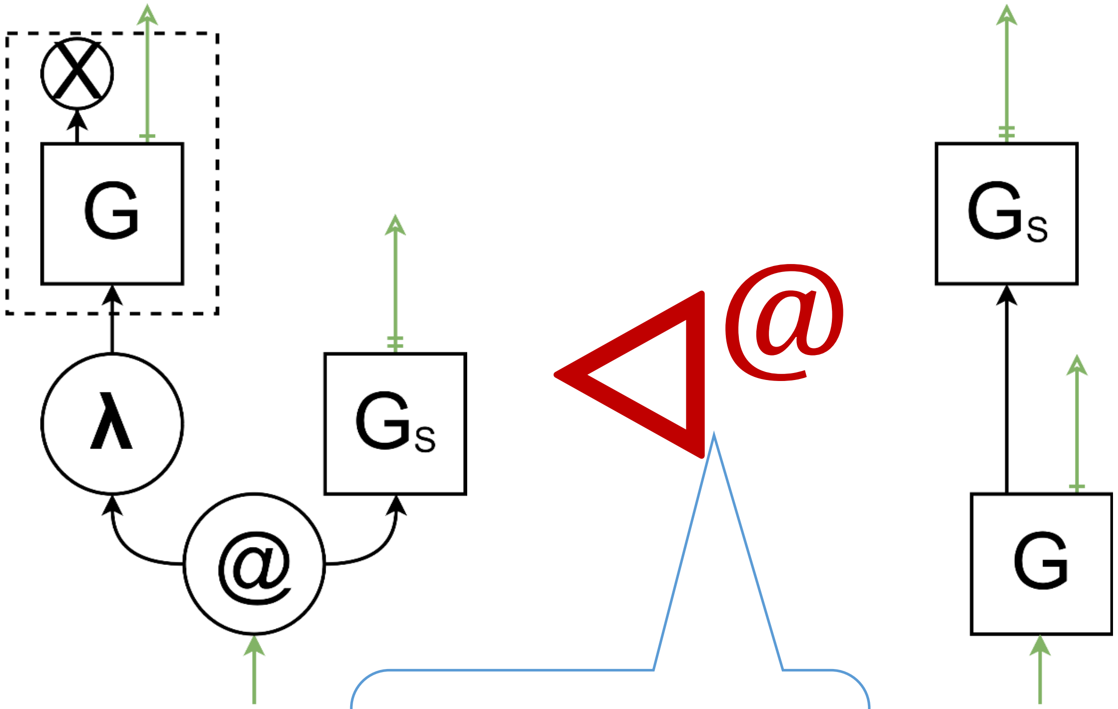
The Characterisation Theorem

We want to show that the
β-Law pre-template
is a *robust template*



The Characterisation Theorem

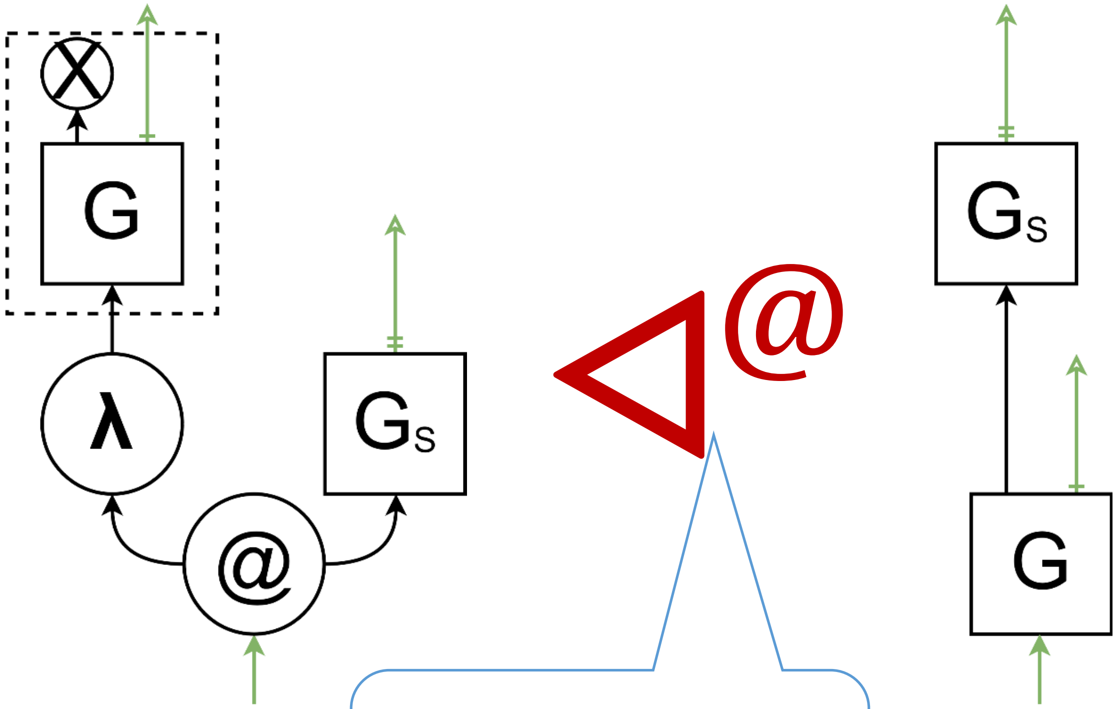
We want to show that the
β-Law pre-template
is a *robust template*



The *β-Law pre-template* is a family of binary relations

The Characterisation Theorem

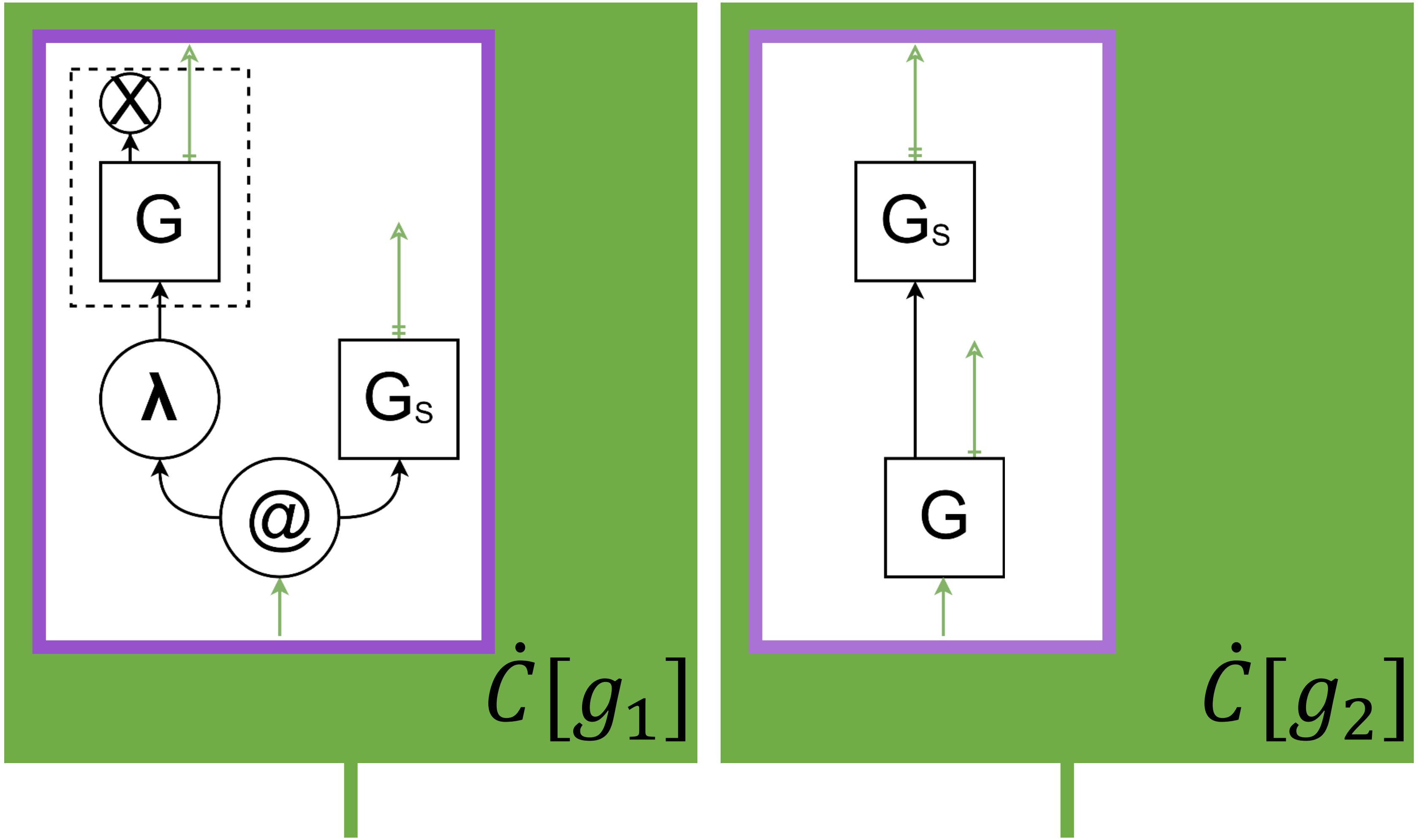
We want to show that the
β-Law pre-template
is a *robust template*



The *β-Law pre-template* is a family of binary relations

Input-safe	?
Output-closed	?
Robust	?

The Characterisation Theorem



Input-safe ?

Output-closed ?

Robust ?

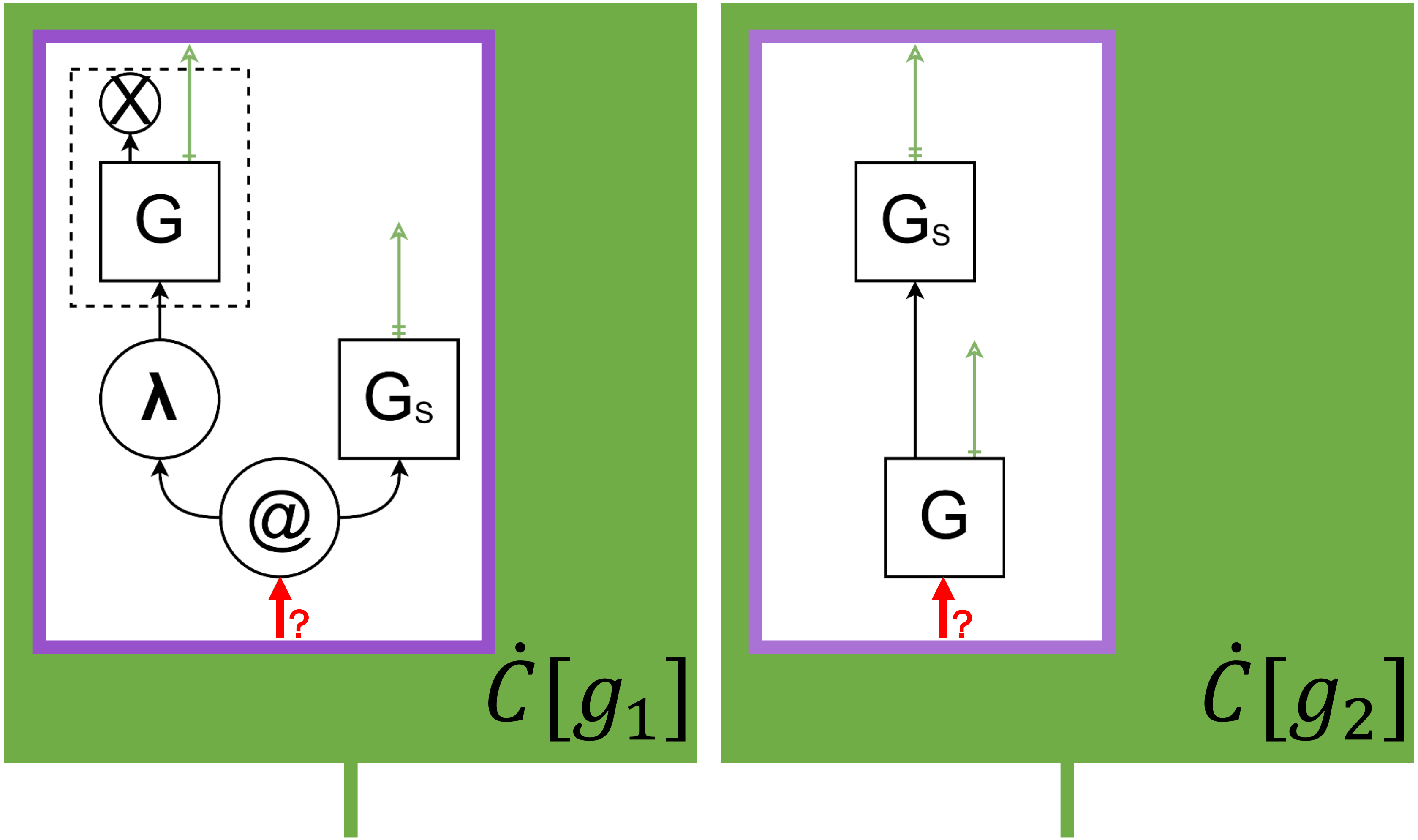
The Characterisation Theorem

Def. The pre-template is *input-safe* if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$ for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$ for two hypernets g_1', g_2' s. t. $g_1' \lhd g_2'$ and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status

Input-safe	?
Output-closed	✓
Robust	?

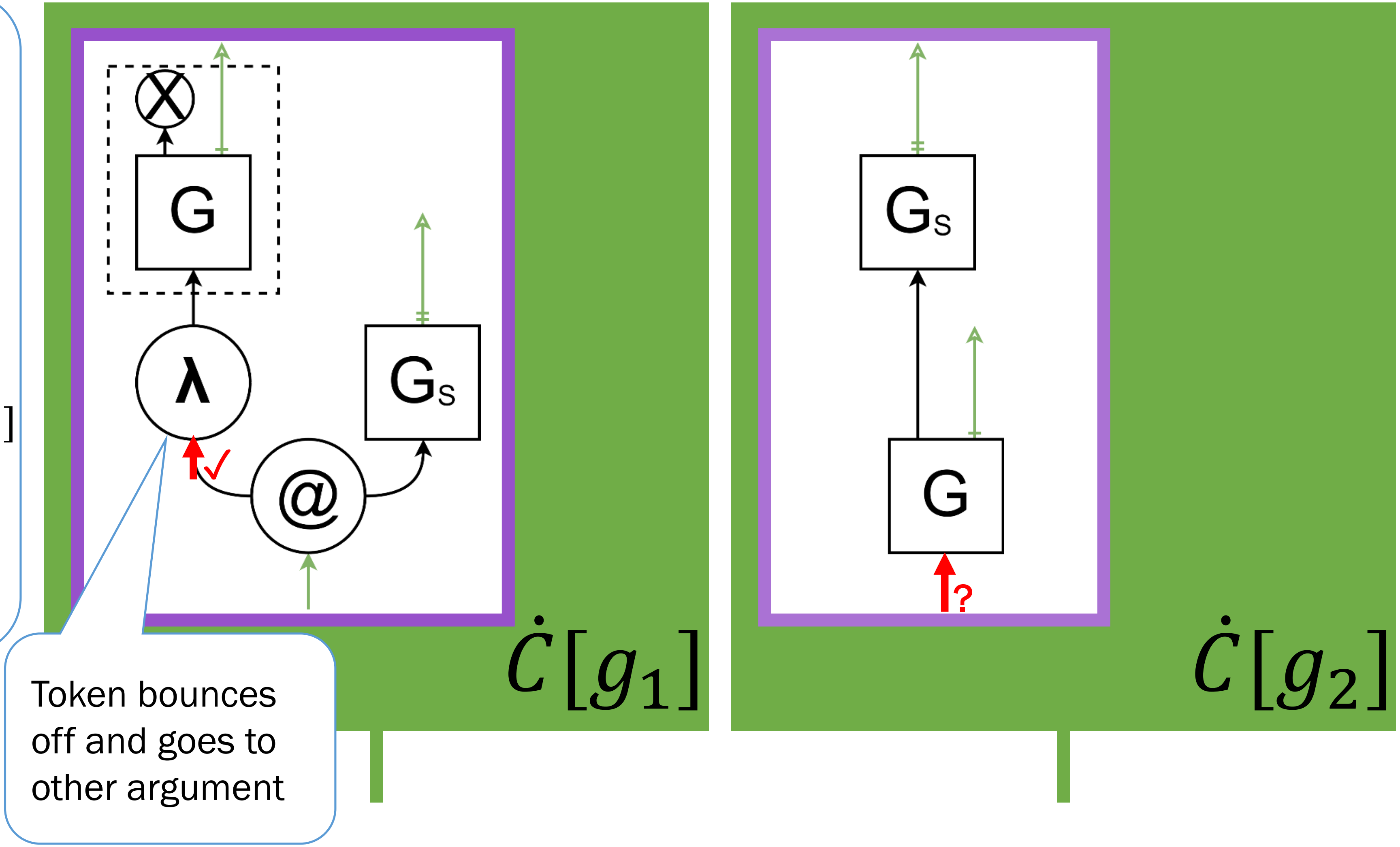


The Characterisation Theorem

Def. The pre-template is *input-safe*
if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token
one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status



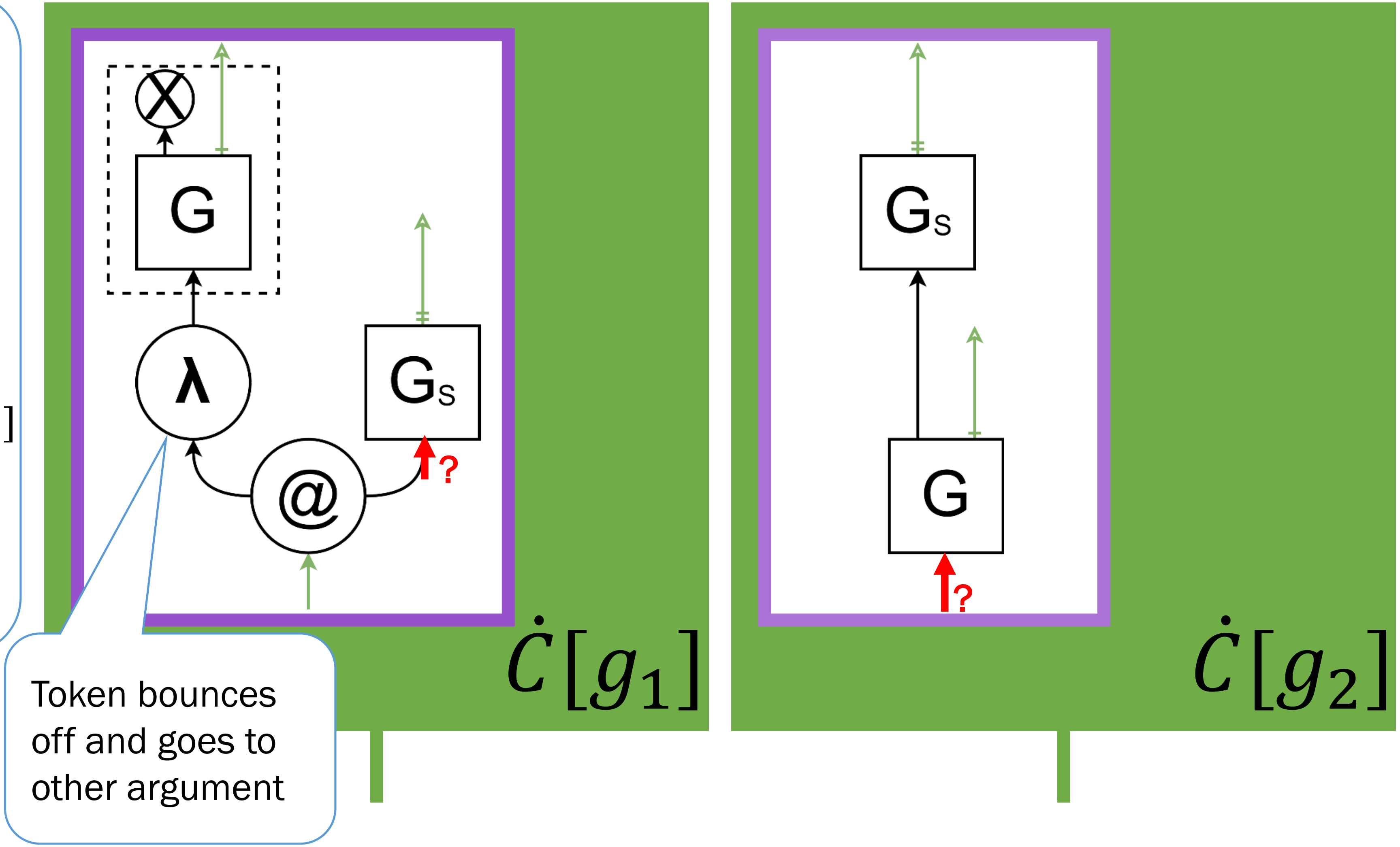
Input-safe	?
Output-closed	✓
Robust	?

The Characterisation Theorem

Def. The pre-template is *input-safe* if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$ for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$ for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$ and a valid *focussed context* $\dot{C}'$
   such that the token is
   not in rewrite status



Input-safe	?
Output-closed	✓
Robust	?

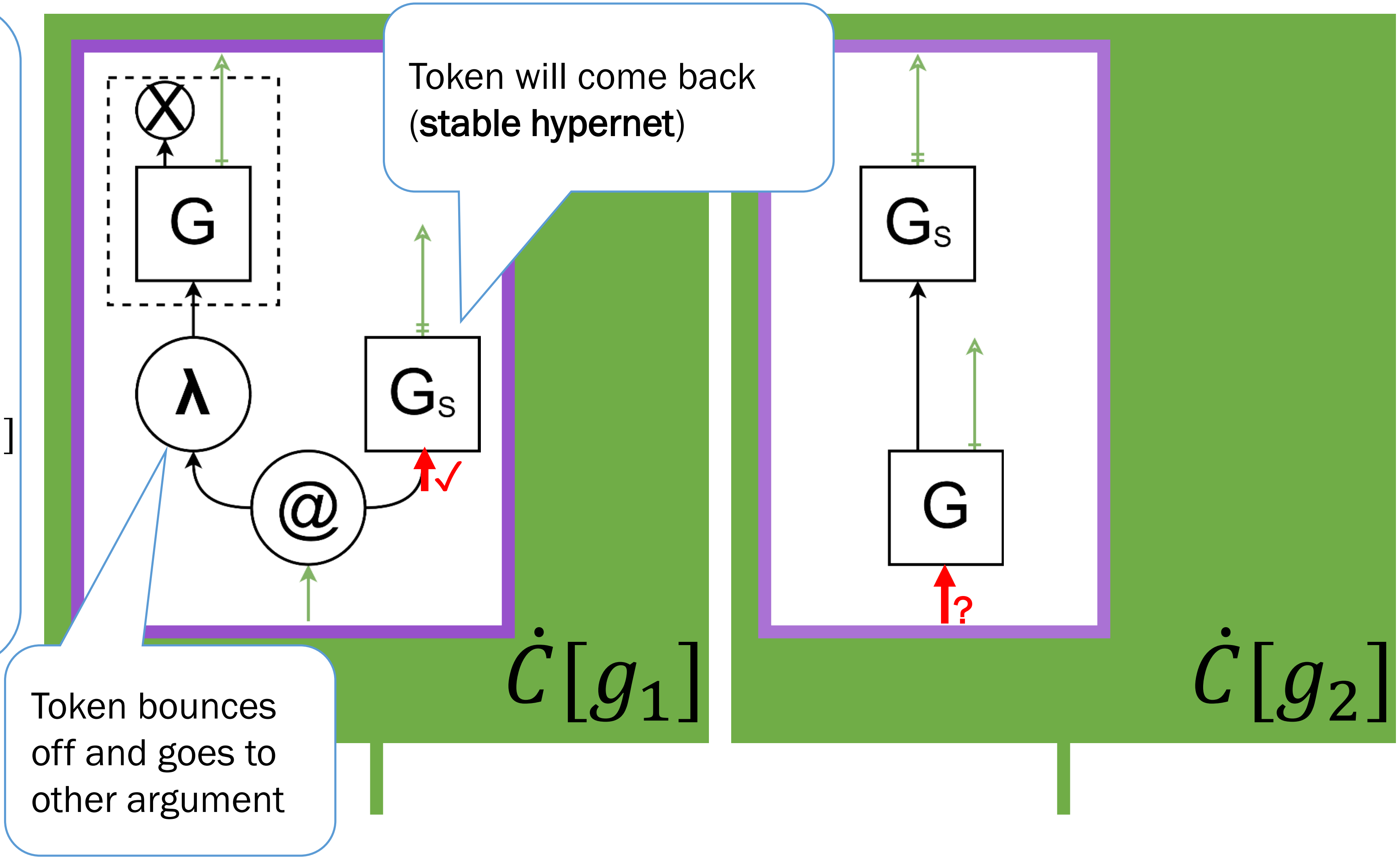
The Characterisation Theorem

Def. The pre-template is *input-safe*
if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token
one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status

Input-safe	?
Output-closed	✓
Robust	?



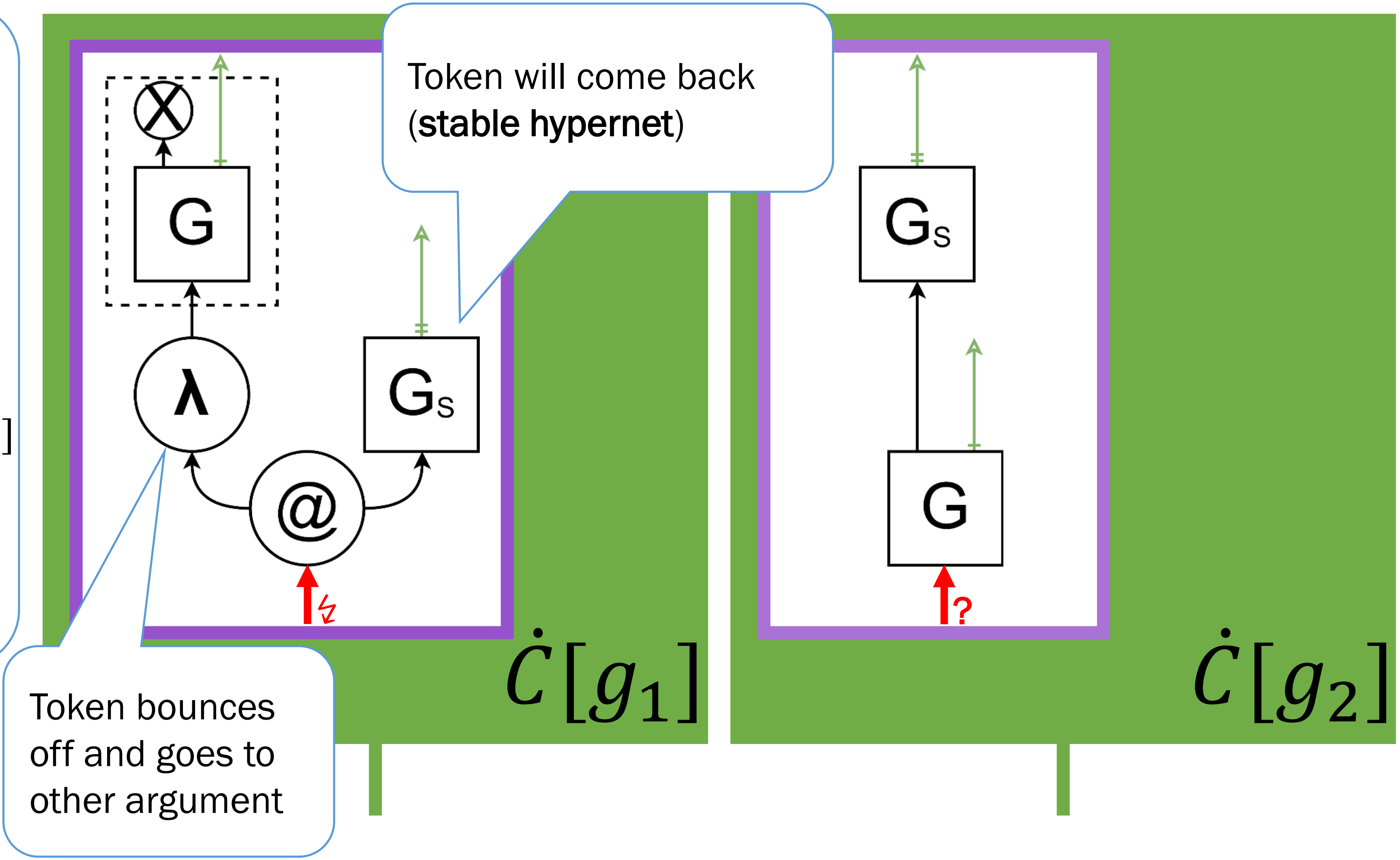
The Characterisation Theorem

Def. The pre-template is *input-safe* if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$ for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$ for two hypernets g_1', g_2' s. t. $g_1' \lhd g_2'$ and a valid *focussed context* $\dot{C}'$ such that the token is not in rewrite status

Input-safe ?
Output-closed ✓
Robust ?

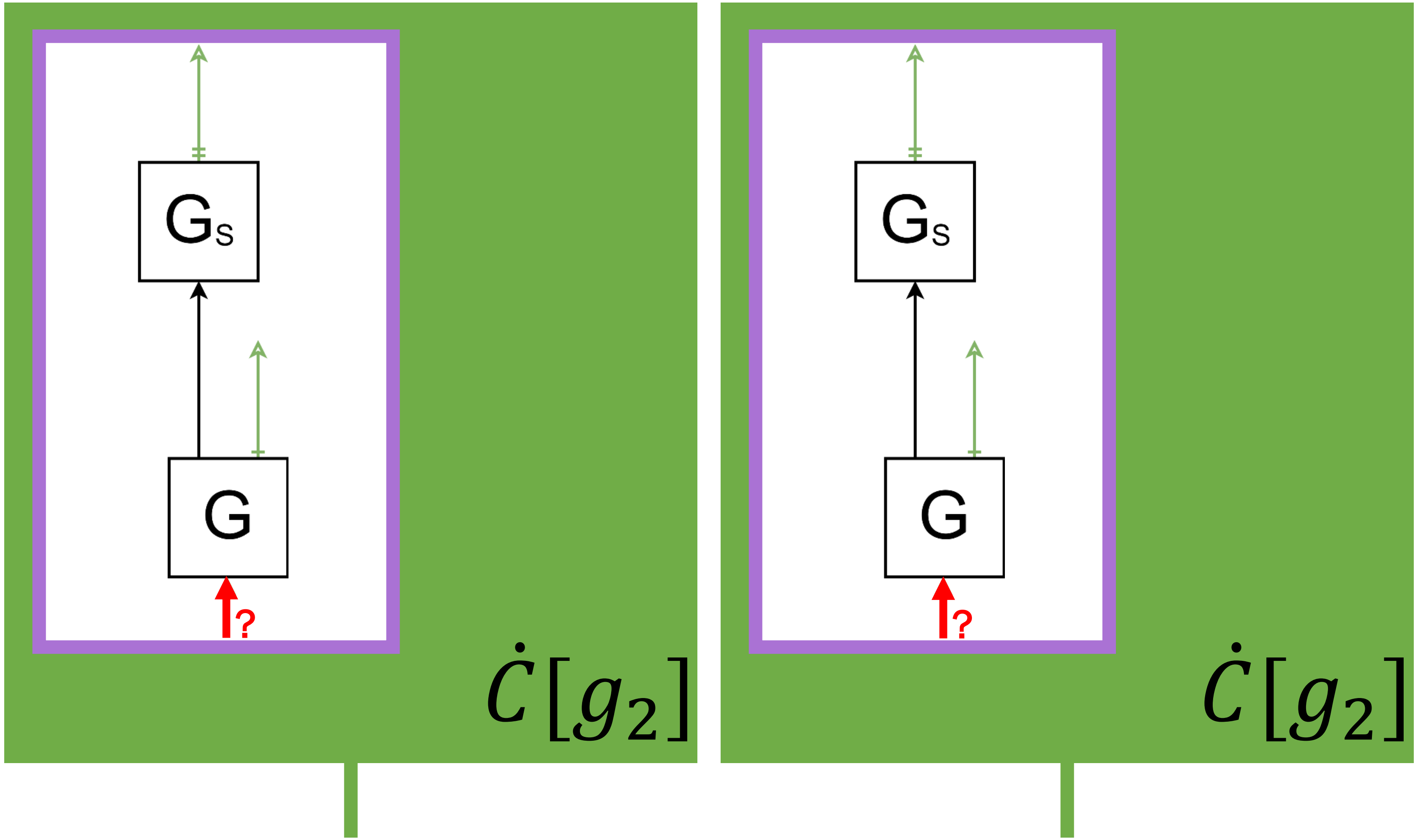


The Characterisation Theorem

Def. The pre-template is *input-safe*
if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token
one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status



Input-safe ?
Output-closed ✓
Robust ?

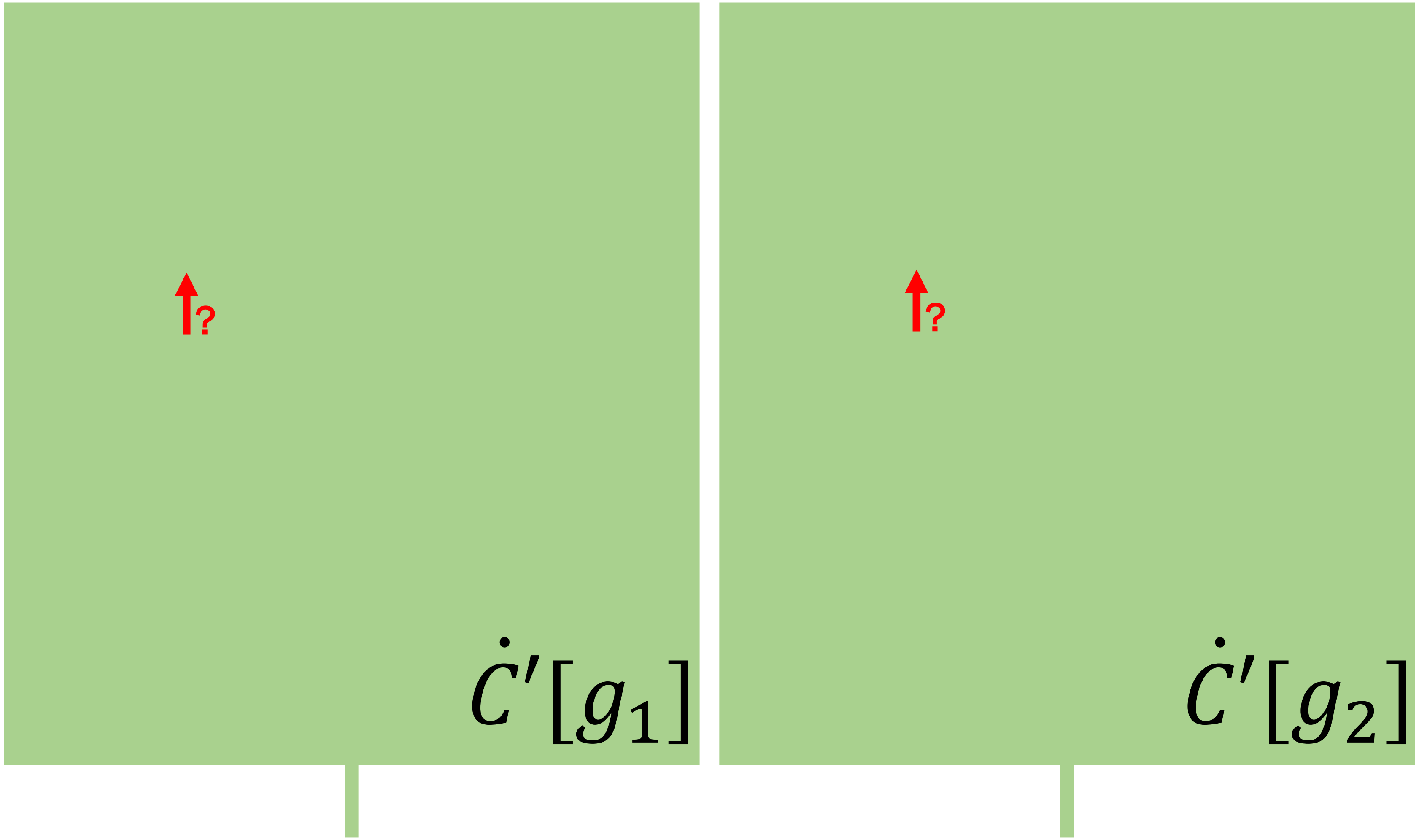
The Characterisation Theorem

Def. The pre-template is *input-safe*
if *input-safe* if for any valid *focussed context* $\dot{C}$ with an input search token
one of the following holds:

1. $\dot{C}[g_1] \to^* \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^* \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status

Input-safe ✓
Output-closed ✓
Robust ?



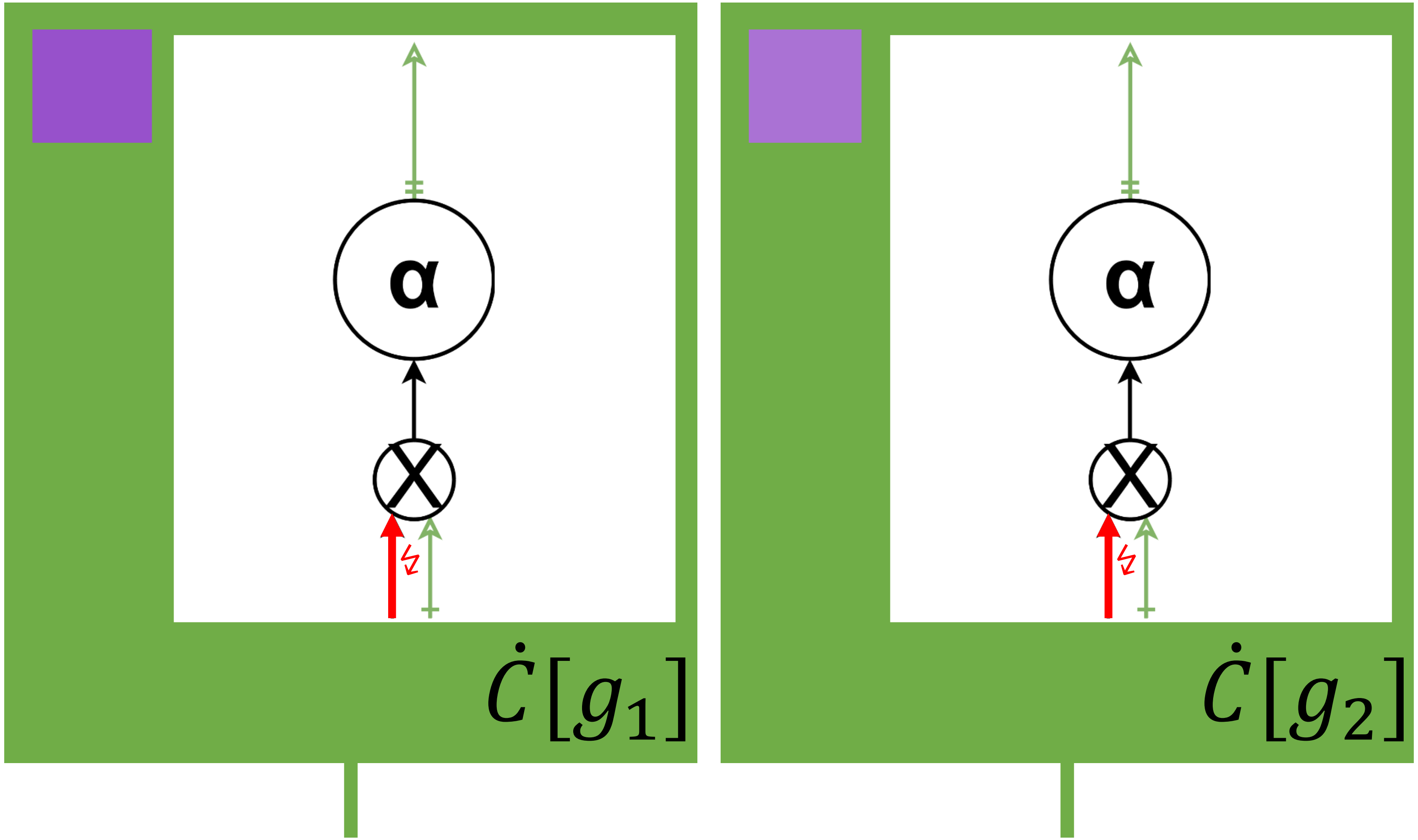
The Characterisation Theorem

Def. The pre-template is *robust* if, for any valid *focussed context* $\dot{C}$ with a rewrite token, one of the following holds:

1. $\dot{C}[g_1] \to^+ \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$ for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^+ \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$ for two hypernets g_1', g_2' s. t. $g_1' \lhd g_2'$ and a valid *focussed context* $\dot{C}'$ such that the token is not in rewrite status

Input-safe	✓
Output-closed	✓
Robust (⊗)	?



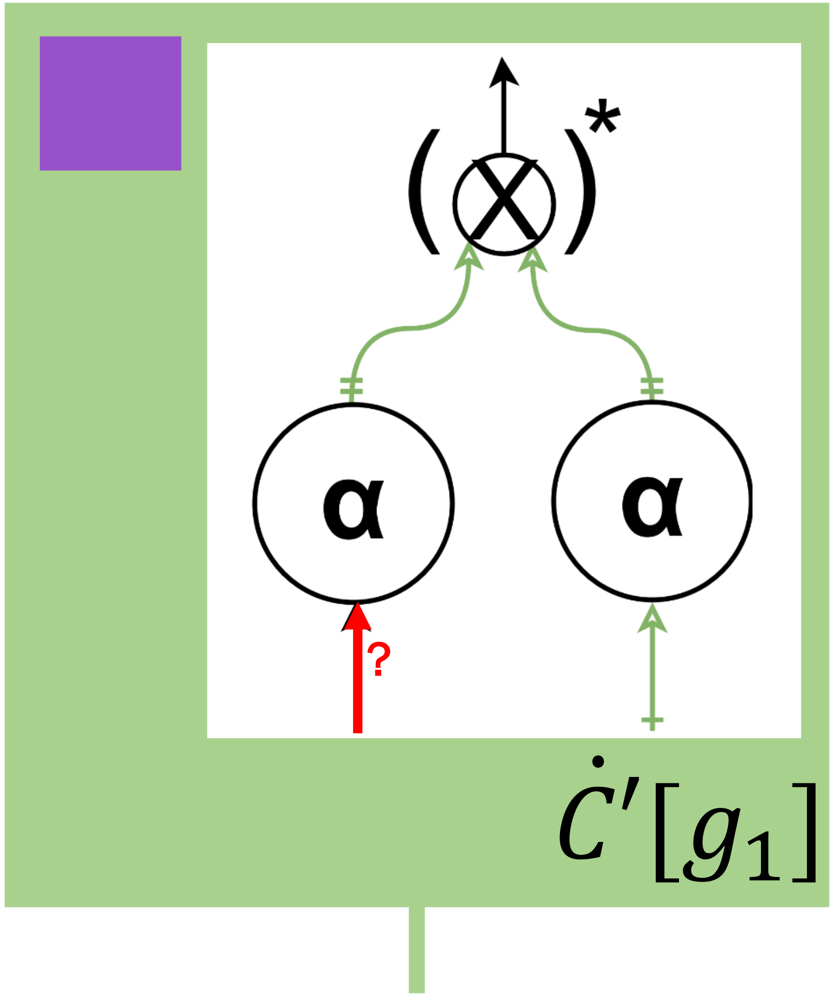
The Characterisation Theorem

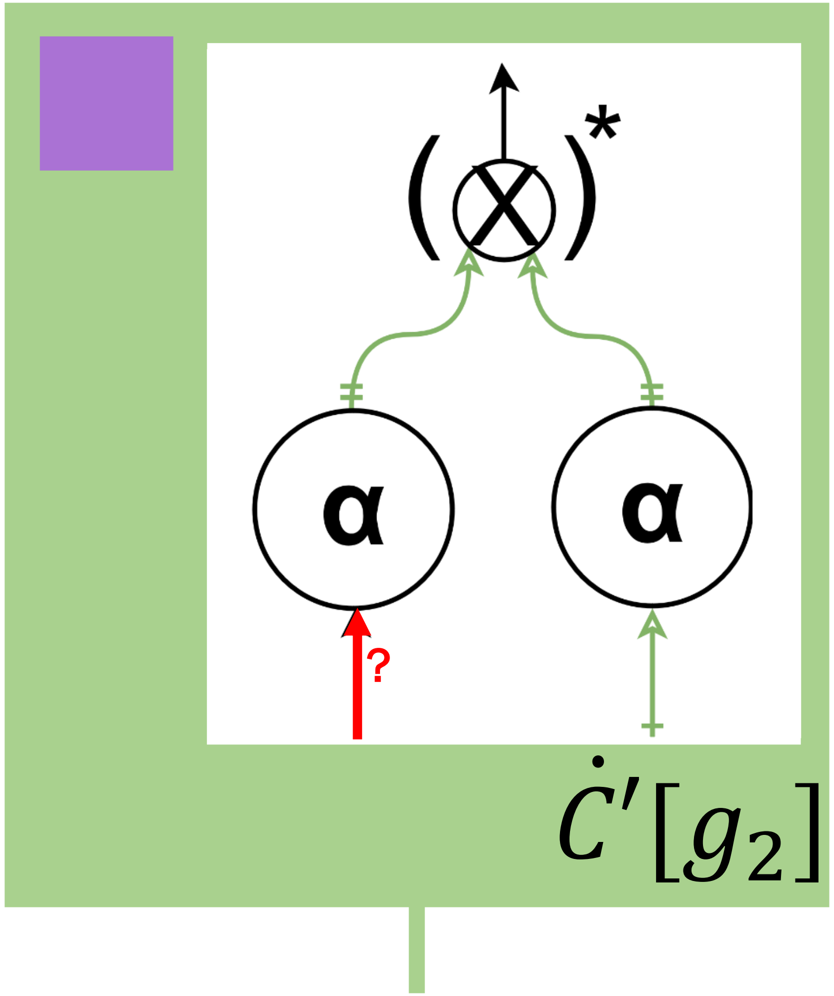
Def. The pre-template is *robust* if, for any valid *focussed context* $\dot{C}$ with a rewrite token, one of the following holds:

1. $\dot{C}[g_1] \to^+ \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$ for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^+ \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$ for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$ and a valid *focussed context* $\dot{C}'$ such that the token is not in rewrite status

Input-safe	✓
Output-closed	✓
Robust (⊗)	✓





The Characterisation Theorem

Def. The pre-template is *robust* if, for any valid *focussed context* $\dot{C}$ with a rewrite token, one of the following holds:

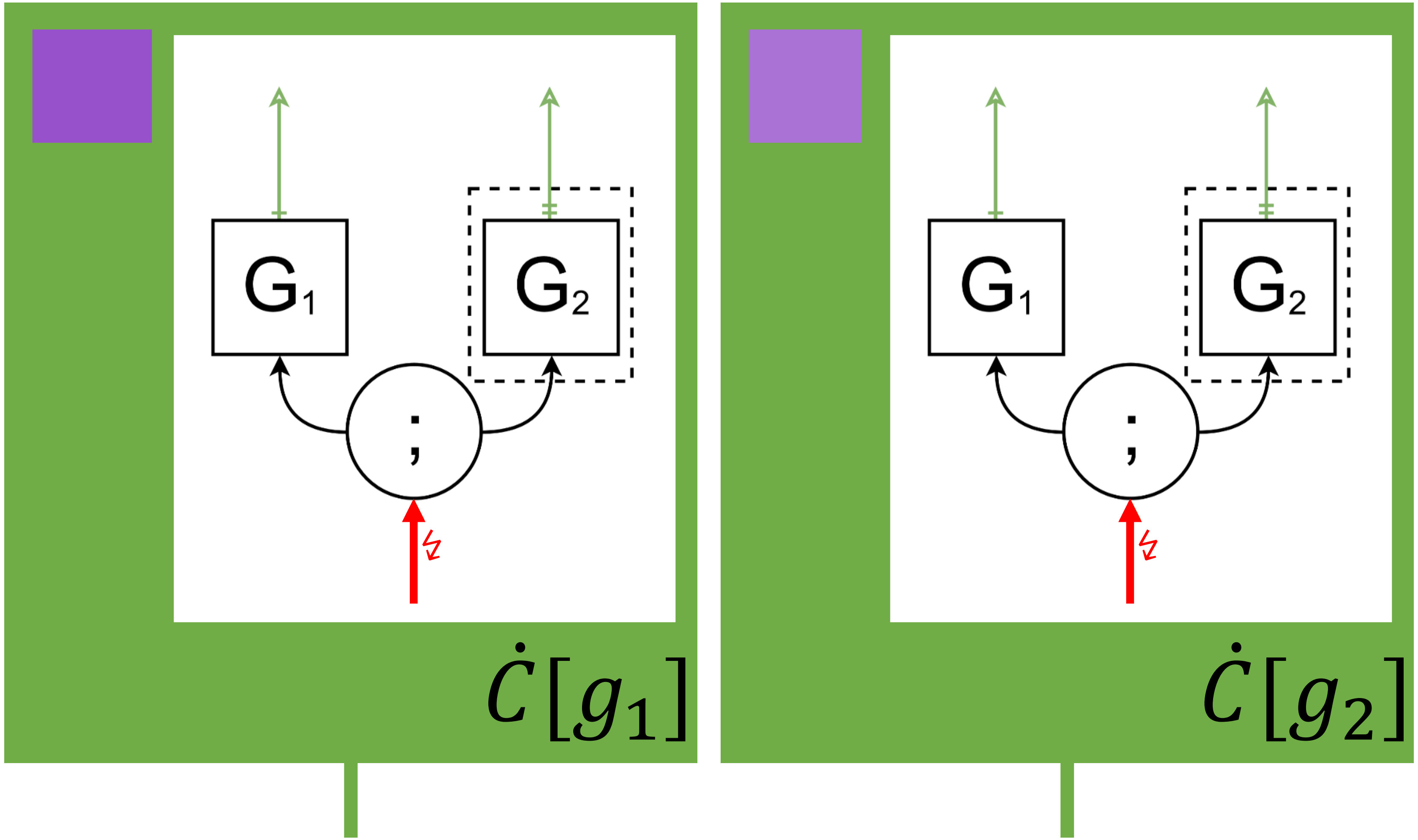
1. $\dot{C}[g_1] \to^+ \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$ for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^+ \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$ for two hypernets g_1', g_2' s. t. $g_1' \lhd g_2'$ and a valid *focussed context* $\dot{C}'$ such that the token is not in rewrite status

Input-safe ✓

Output-closed ✓

Robust (;) ?

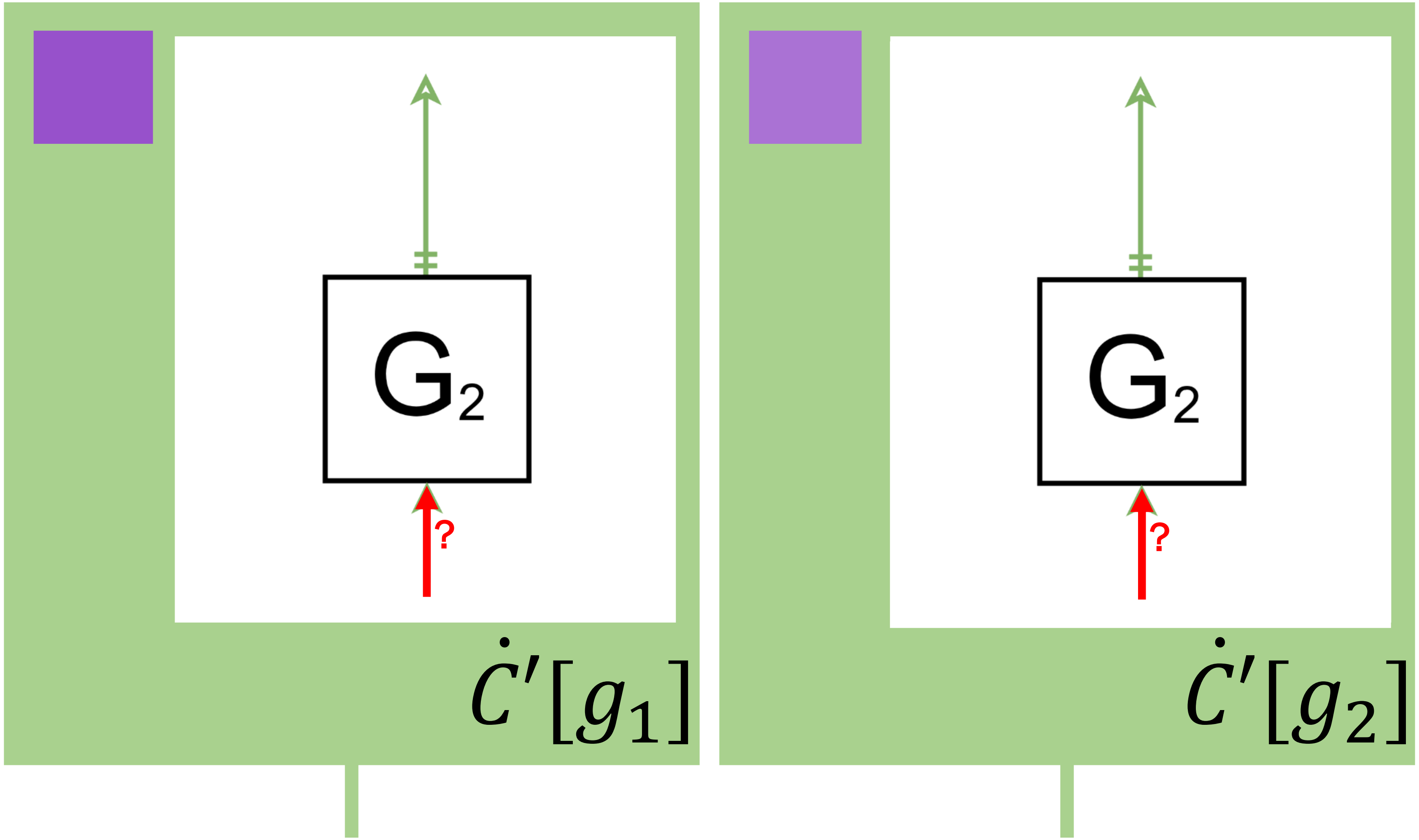


The Characterisation Theorem

Def. The pre-template is *robust* if, for any valid *focussed context* $\dot{C}$
with a rewrite token,
one of the following holds:

1. $\dot{C}[g_1] \to^+ \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^+ \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
for two hypernets g_1', g_2' s. t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status



Input-safe	✓
Output-closed	✓
Robust (;)	✓

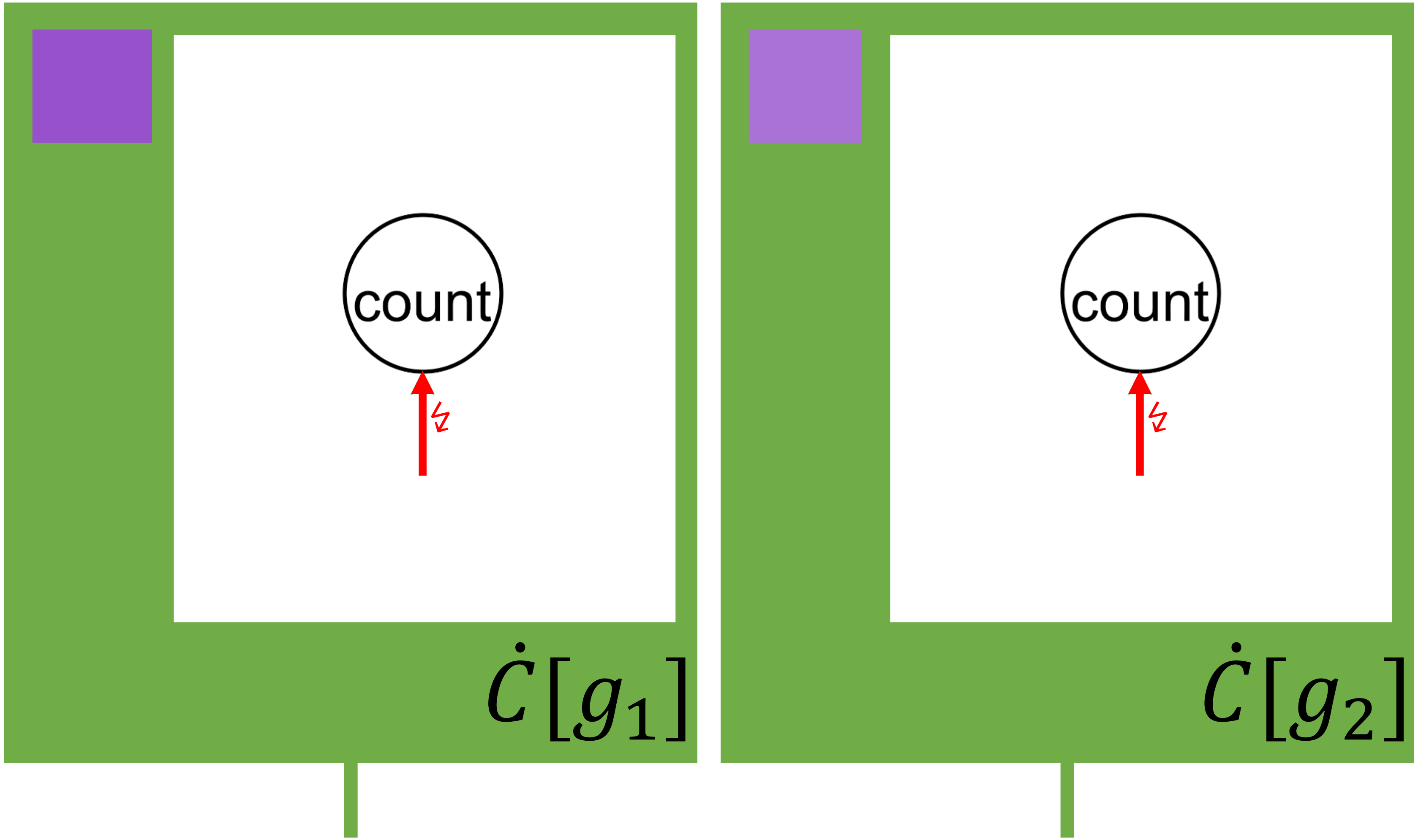
The Characterisation Theorem

Def. The pre-template is *robust* if, for any valid *focussed context* $\dot{C}$
with a rewrite token,
one of the following holds:

1. $\dot{C}[g_1] \to^+ \dot{N}$ and $\dot{C}[g_2] \to^* \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \to^+ \dot{C}'[g_1']$ and $\dot{C}[g_2] \to^* \dot{C}'[g_2']$
for two hypernets g_1', g_2' s. t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status

Input-safe	✓
Output-closed	✓
Robust (`count`)	**?**



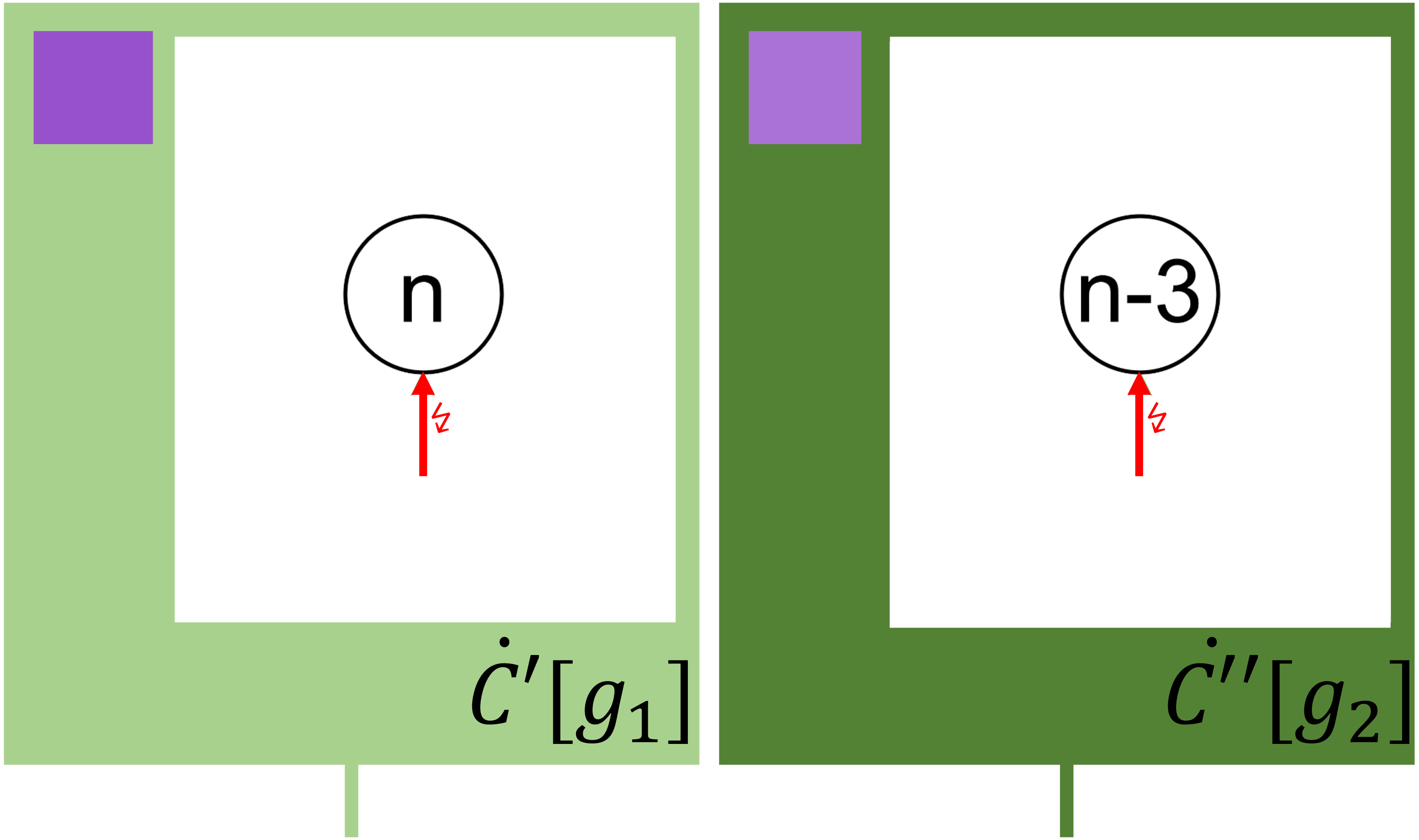
The Characterisation Theorem

Def. The pre-template is *robust* if, for any valid *focussed context* $\dot{C}$
with a rewrite token,
one of the following holds:

1. $\dot{C}[g_1] \rightarrow^{+} \dot{N}$ and $\dot{C}[g_2] \rightarrow^{*} \dot{N}_2$
for two stuck states $\dot{N}_1, \dot{N}_2$

2. $\dot{C}[g_1] \rightarrow^{+} \dot{C}'[g_1']$ and $\dot{C}[g_2] \rightarrow^{*} \dot{C}'[g_2']$
for two hypernets g_1', g_2' s.t. $g_1' \lhd g_2'$
and a valid *focussed context* $\dot{C}'$
such that the token is
not in rewrite status

Input-safe ✓
Output-closed ✓
Robust (`count`) ✗



An Application of The Characterisation Theorem

$$\big(\lambda x.\, \lambda f.\, (\lambda z.\, !\, x)(f\, ()\,)\big)(\mathrm{ref}\ 1) \;\equiv^{?}\; \lambda g.\, (\lambda y.\, 1)(f\, ()\,)$$

An Application of The Characterisation Theorem

$$\forall C.\, C\big[\big(\lambda x.\, \lambda f.\, (\lambda z.\, !\, x)(f\,())\big)(\mathrm{ref}\ 1)\big] \;\simeq^{?}\; C[\lambda g.\, (\lambda y.\, 1)(g\,())]$$

An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.\,!\,x)(f\,()))(\mathrm{ref}\,1)] \;\simeq^{?}\; C[\lambda g.(\lambda y.1)(g\,())]$$

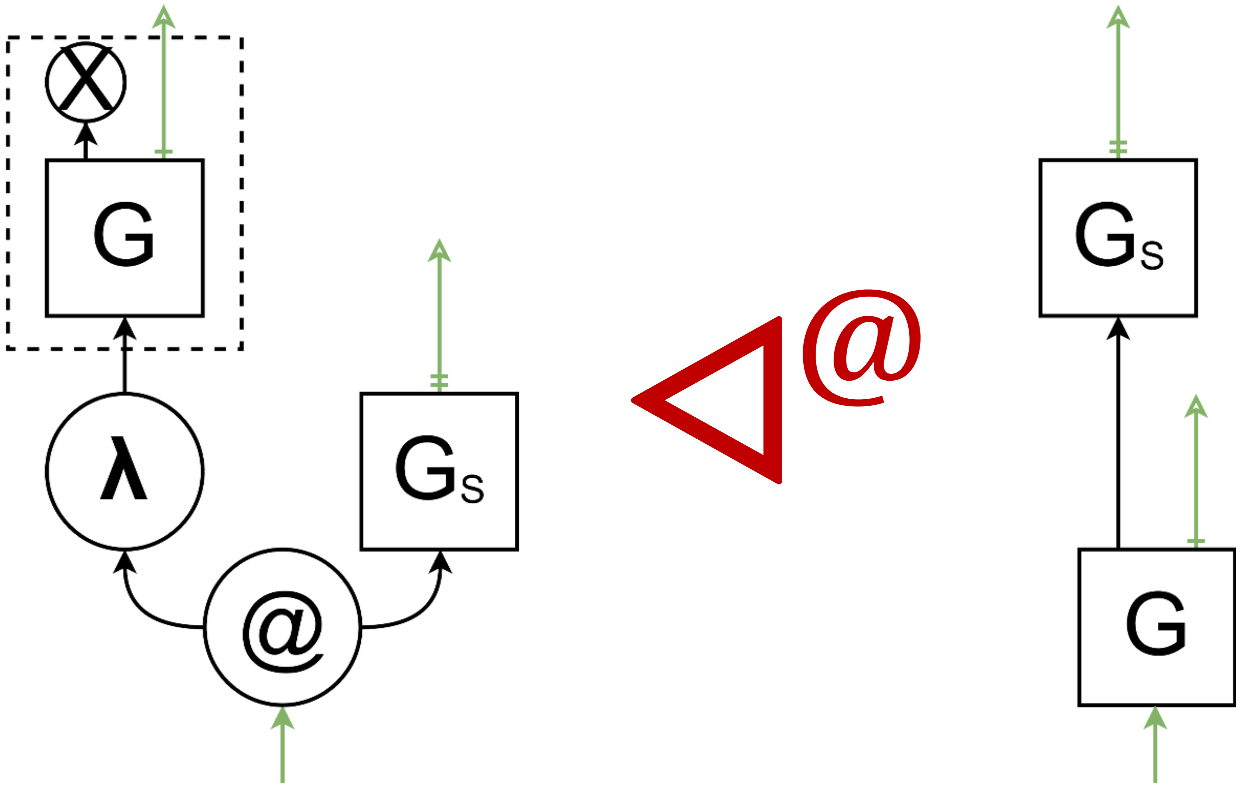
Characterisation Theorem.
Robust templates induce observational equivalences

An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: β-Law

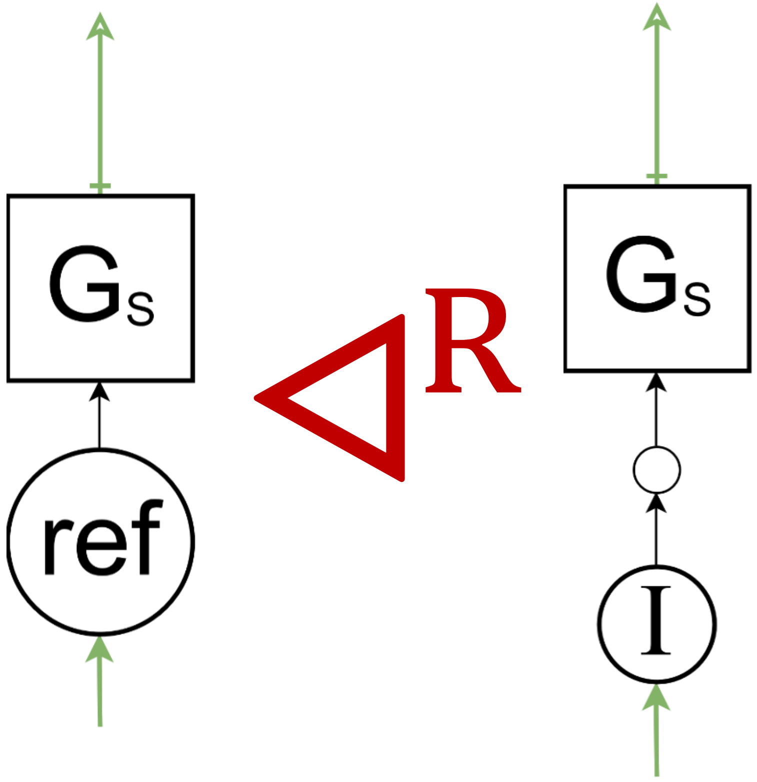


An Application of The Characterisation Theorem

$$\forall C. C[(\lambda x. \lambda f. (\lambda z. !\, x)(f\, ()))(\text{ref } 1)] \simeq^{?} C[\lambda g. (\lambda y. 1)(g\, ())]$$

Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: β-Law
Robust Template 2: ‘Ref’ Rewrite

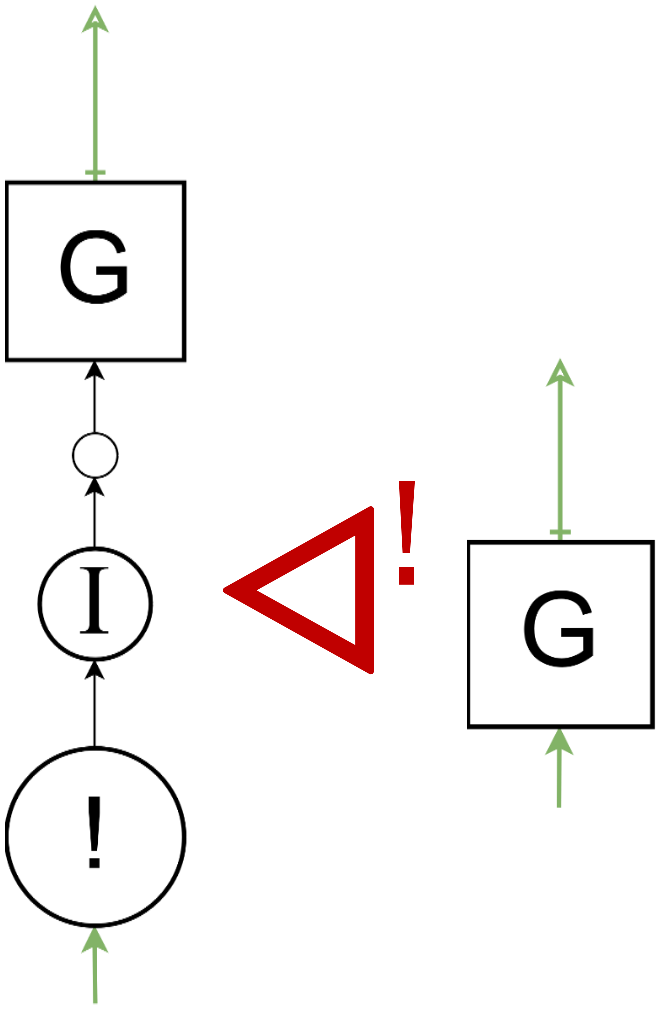


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\text{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: β-Law
Robust Template 2: ‘Ref’ Rewrite
Robust Template 3: ‘!’ Rewrite

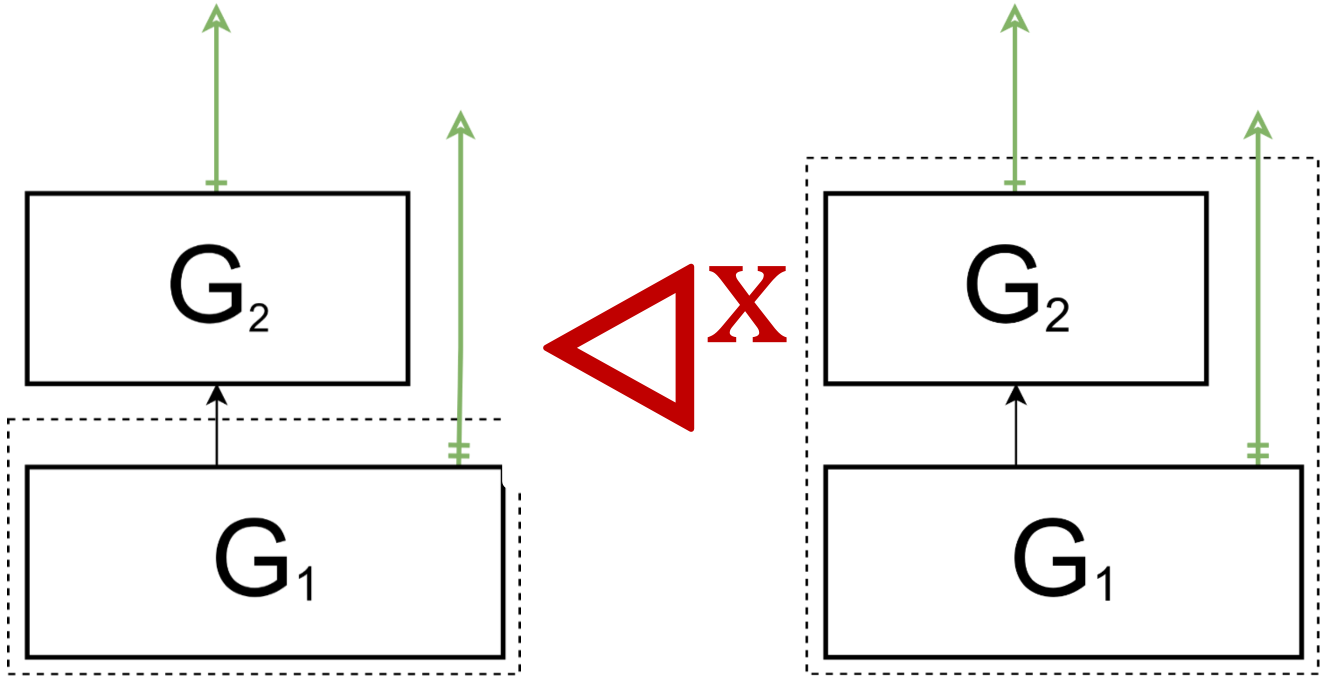


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

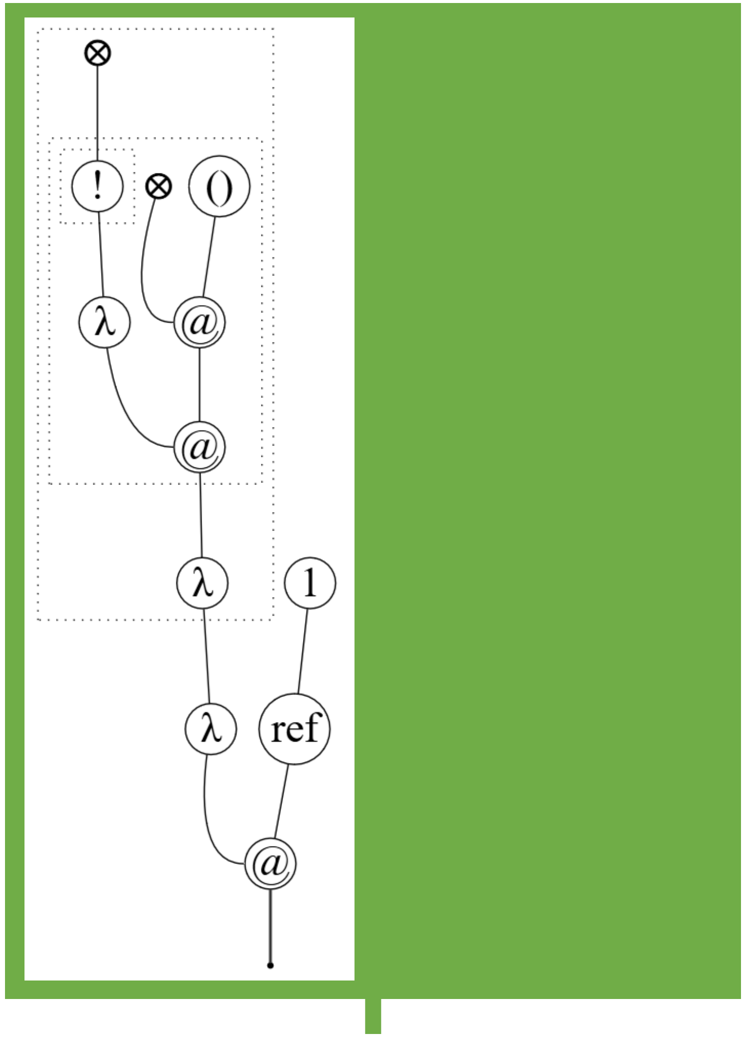
Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: β-Law
Robust Template 2: ‘Ref’ Rewrite
Robust Template 3: ‘!’ Rewrite
Robust Template 4: Extend Thunk

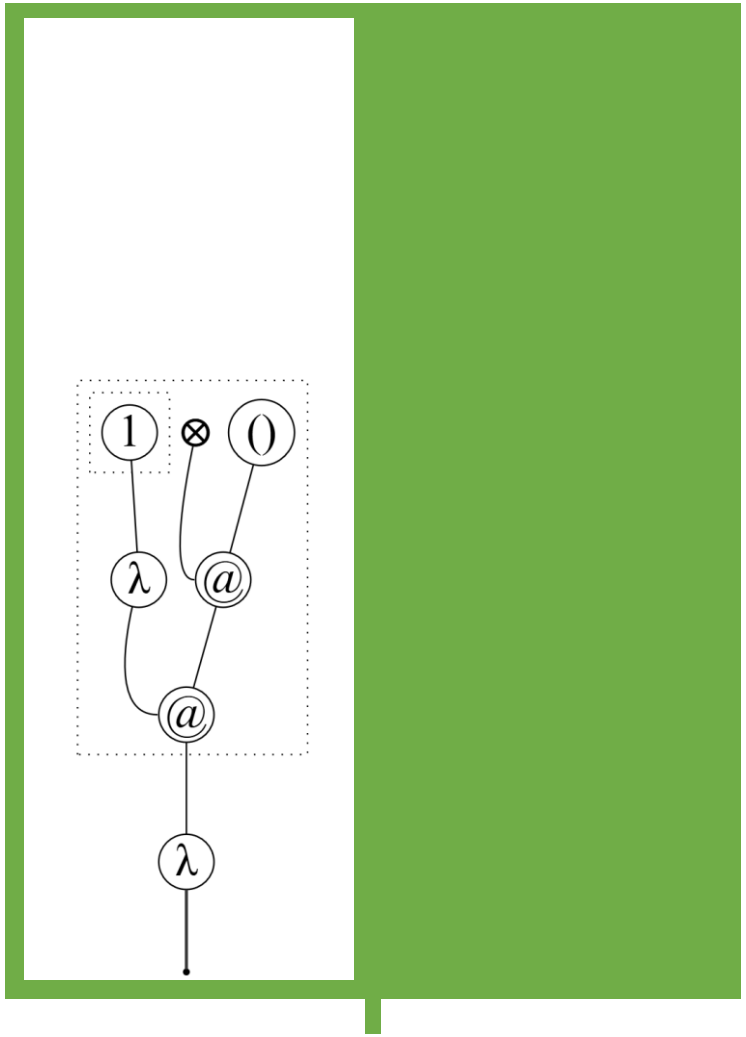


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

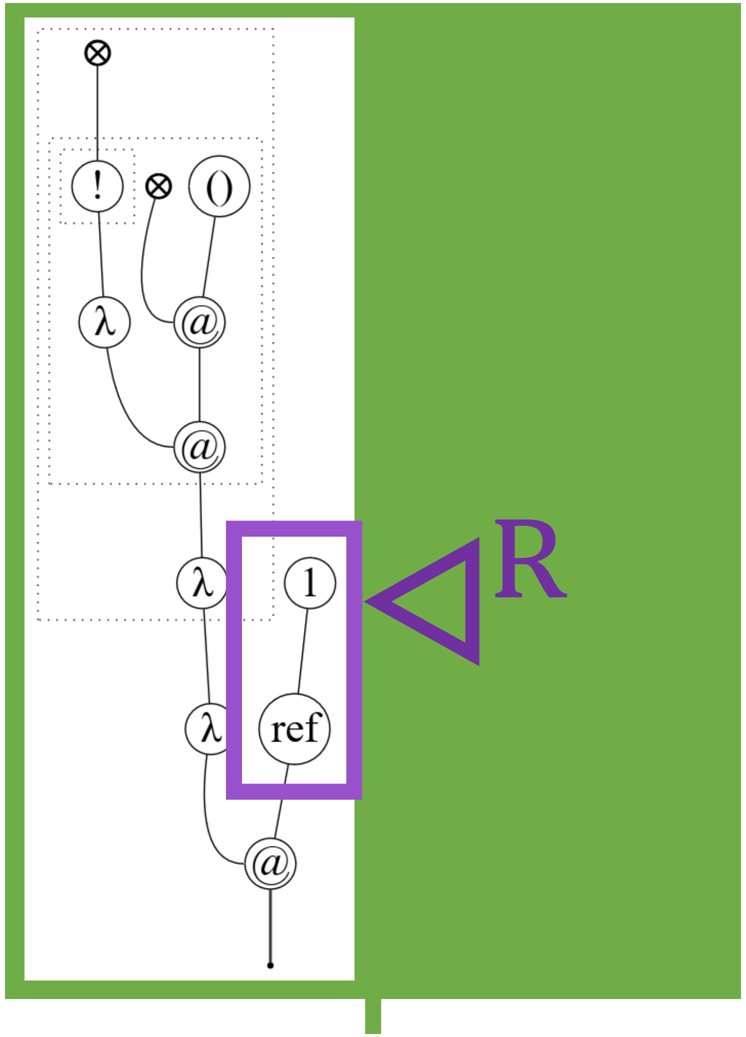


$\simeq^{?}$

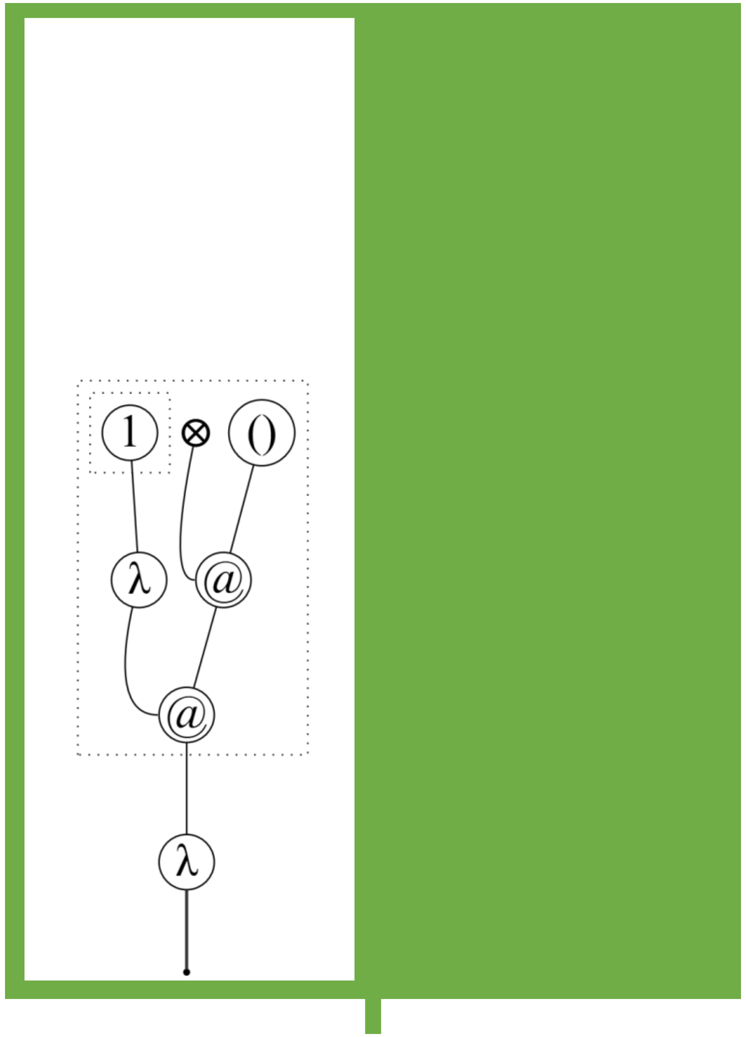


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x. \lambda f. (\lambda z. !\, x)(f\,()))(\mathrm{ref}\ 1)] \simeq^{?} C[\lambda g. (\lambda y. 1)(g\,())]$$

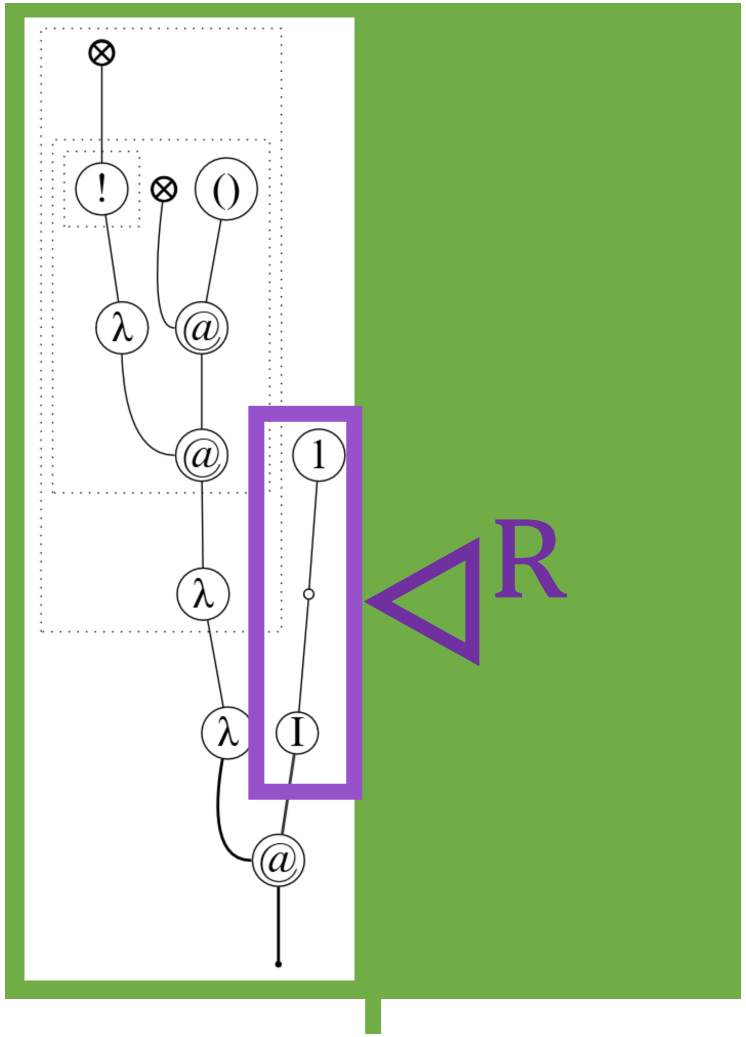


$\simeq^{?}$

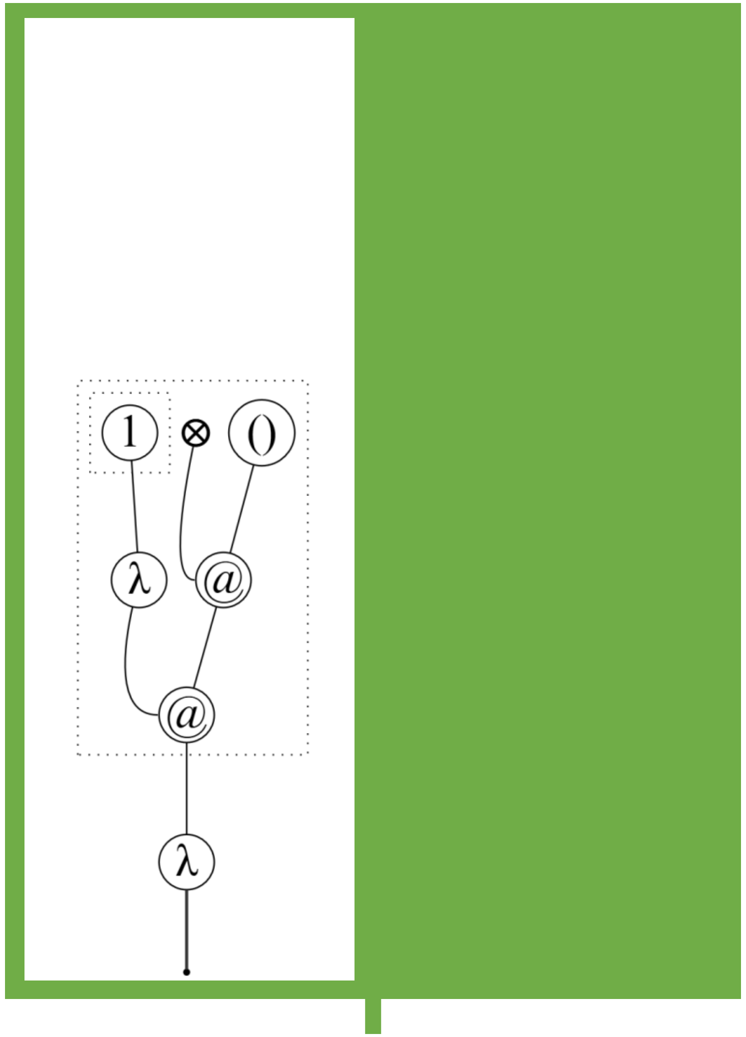


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!x)(f\,())) (\mathrm{ref}\,1)] \simeq^? C[\lambda g.(\lambda y.1)(g\,())]$$

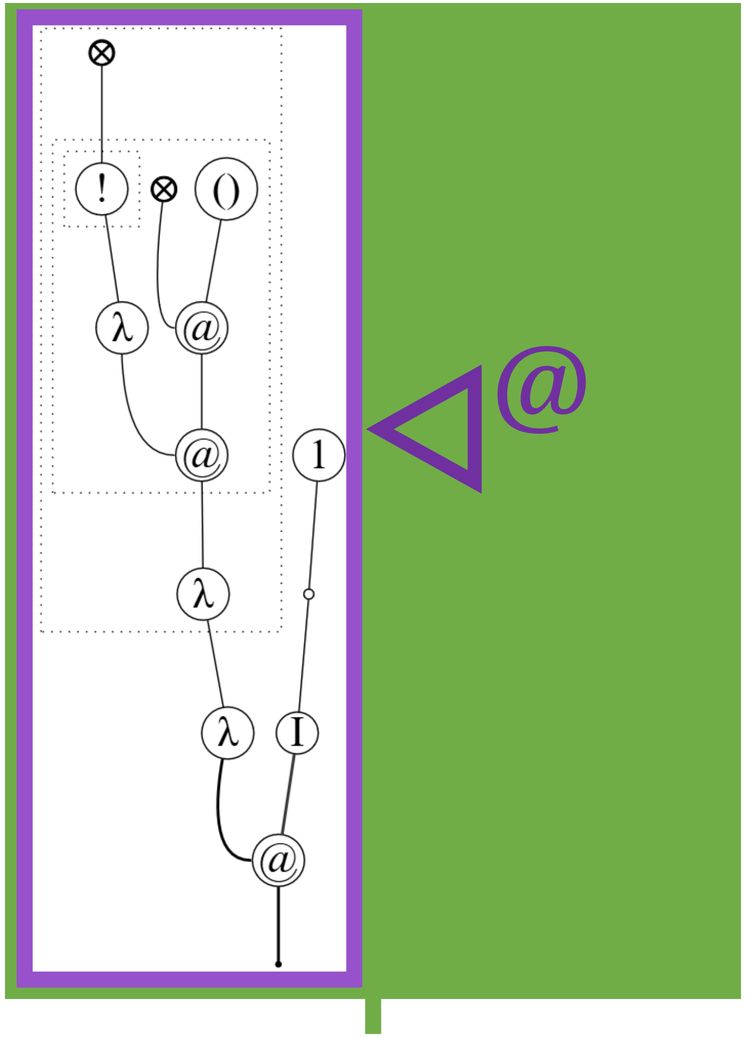


$\simeq^?$

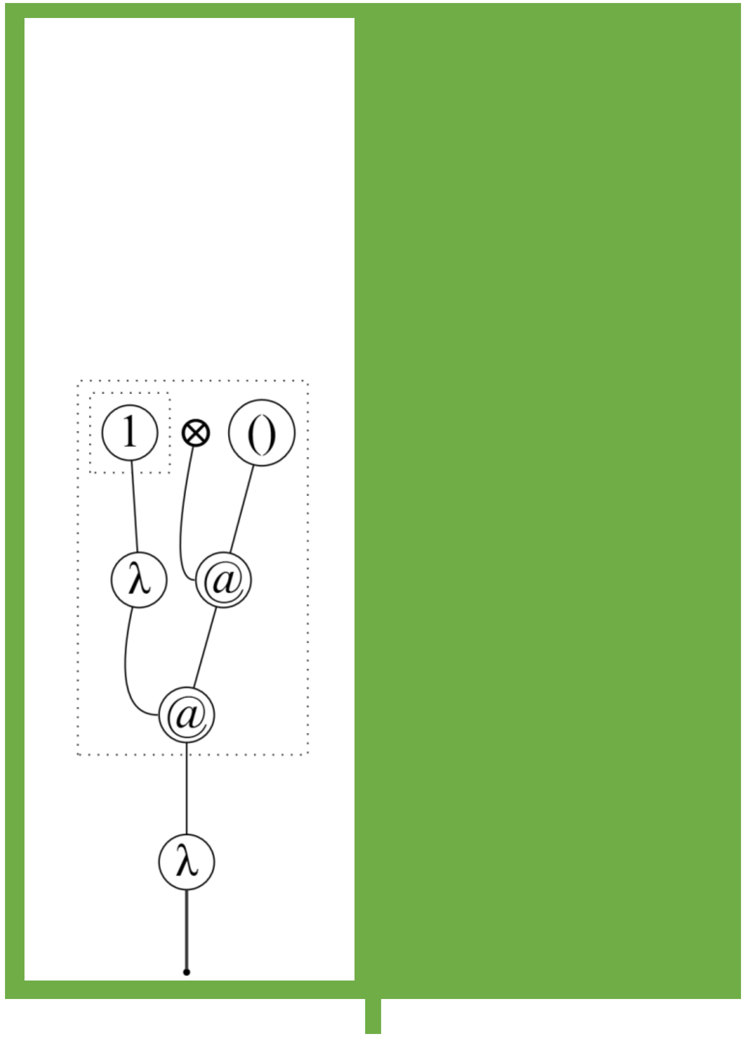


An Application of The Characterisation Theorem

$$\forall \mathcal{C}.\, \mathcal{C}[(\lambda x.\lambda f.(\lambda z.!x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} \mathcal{C}[\lambda g.(\lambda y.1)(g\,())]$$

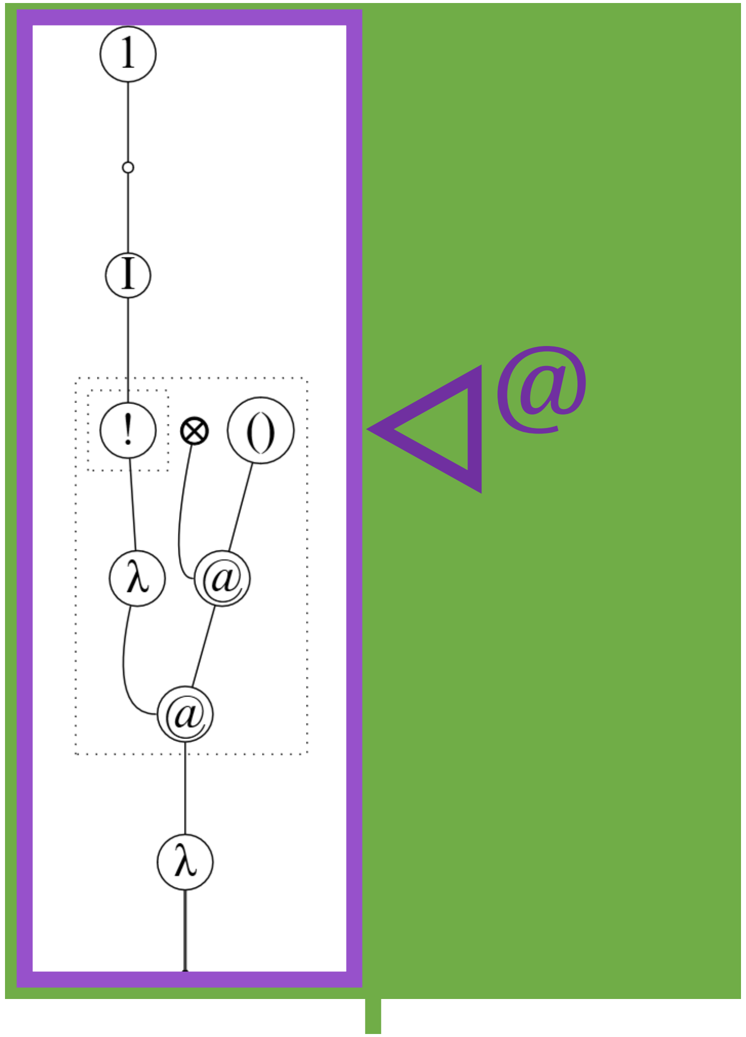


$\simeq^{?}$

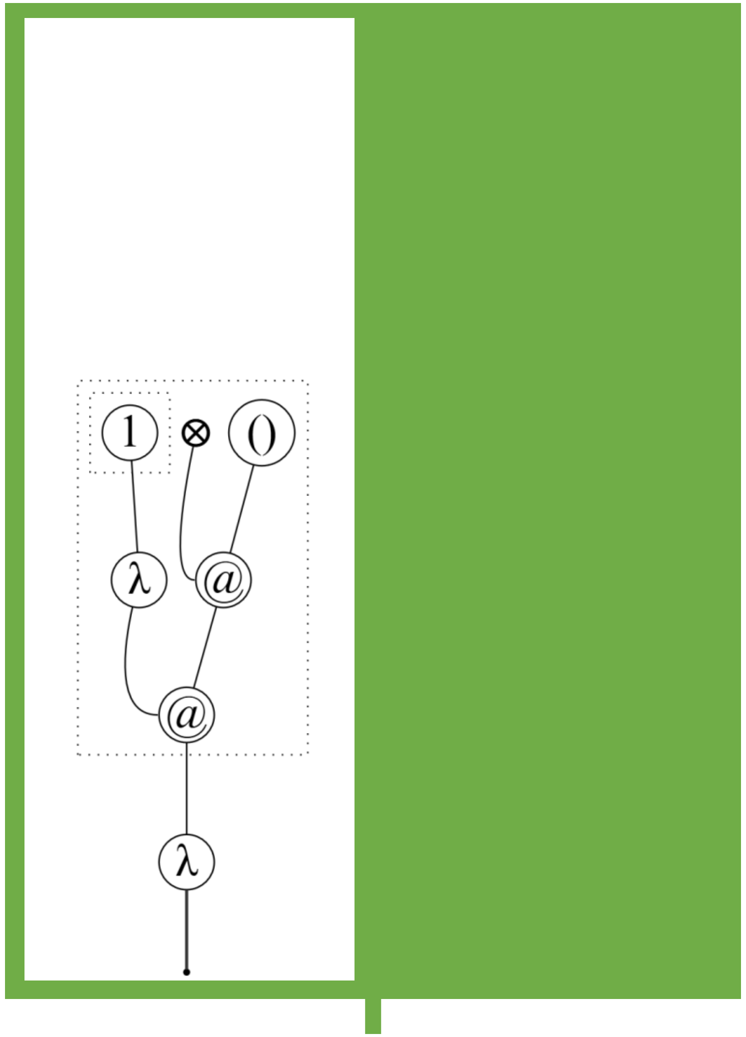


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

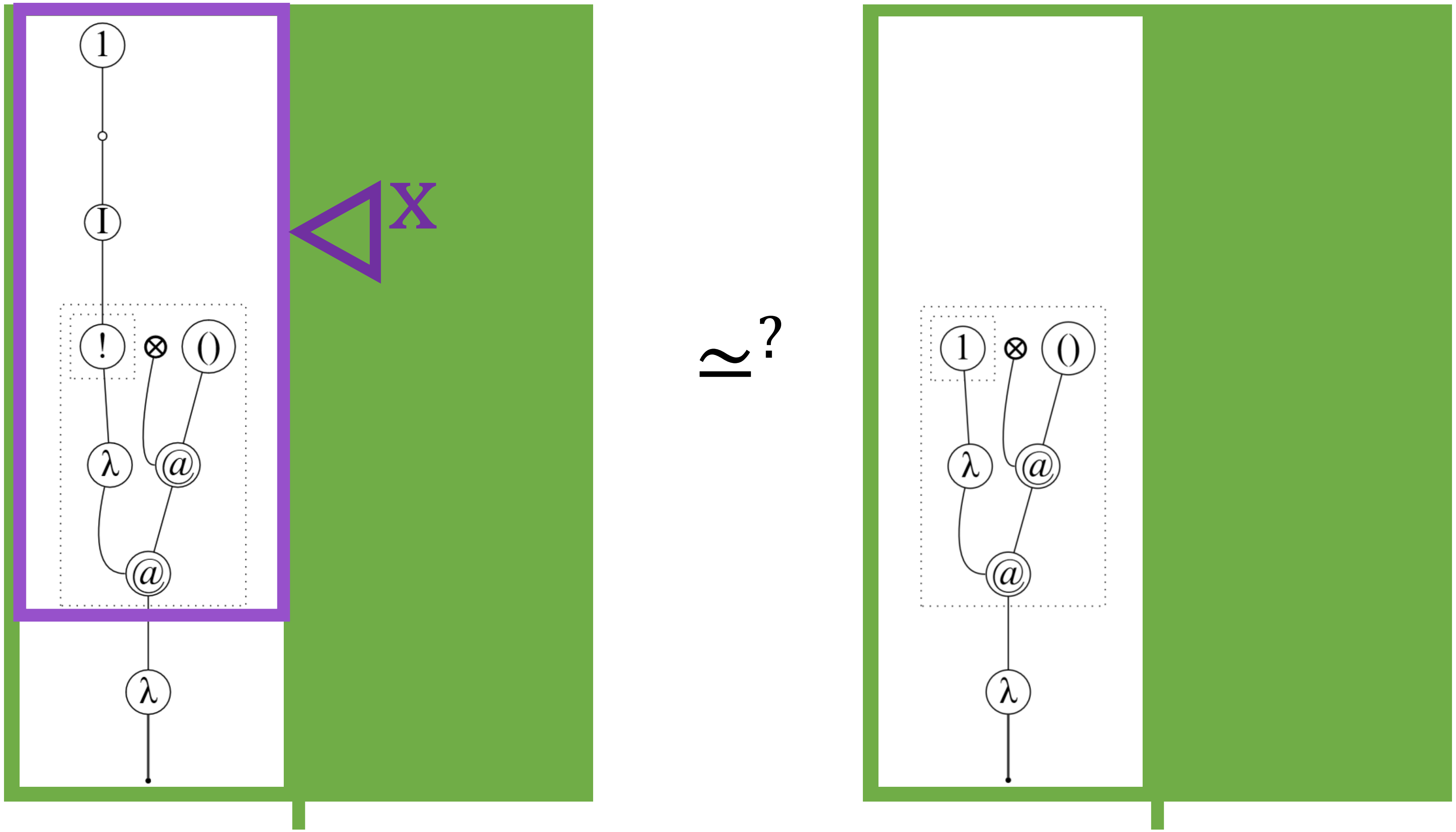


$\simeq^{?}$



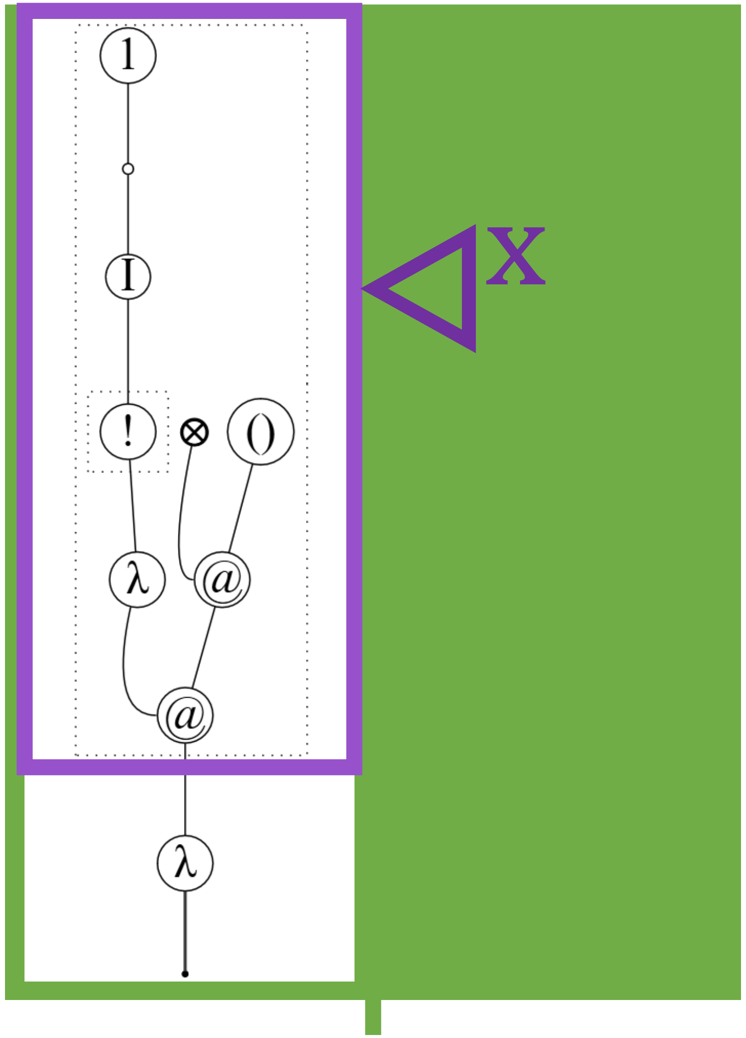
An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

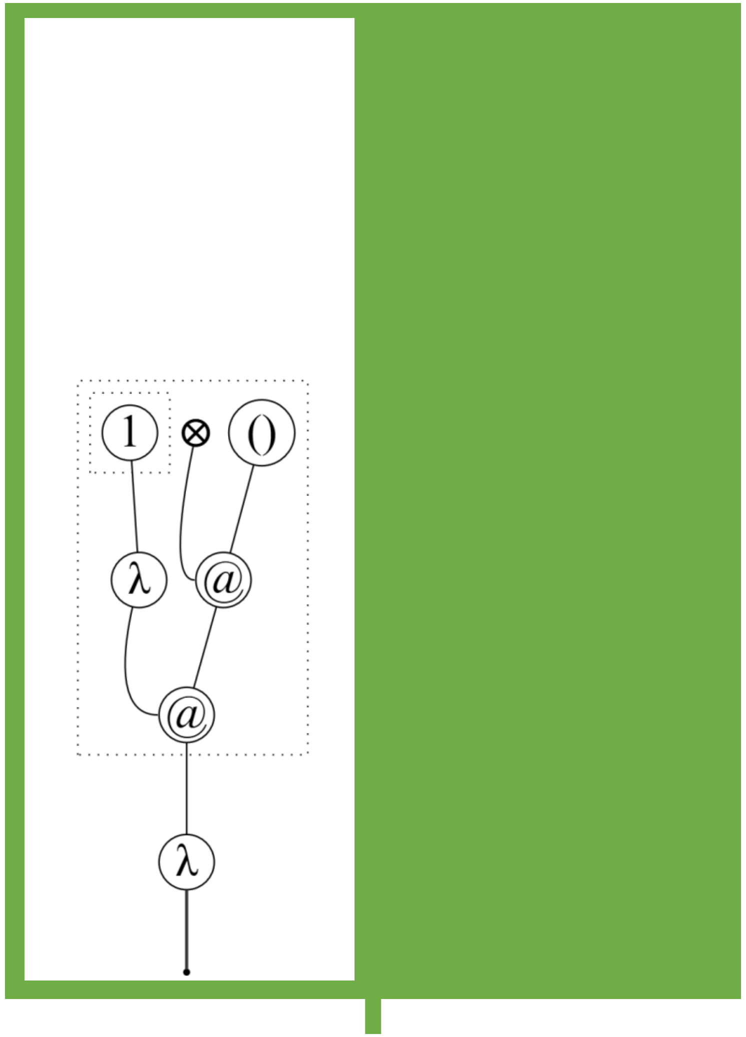


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.\,!\,x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

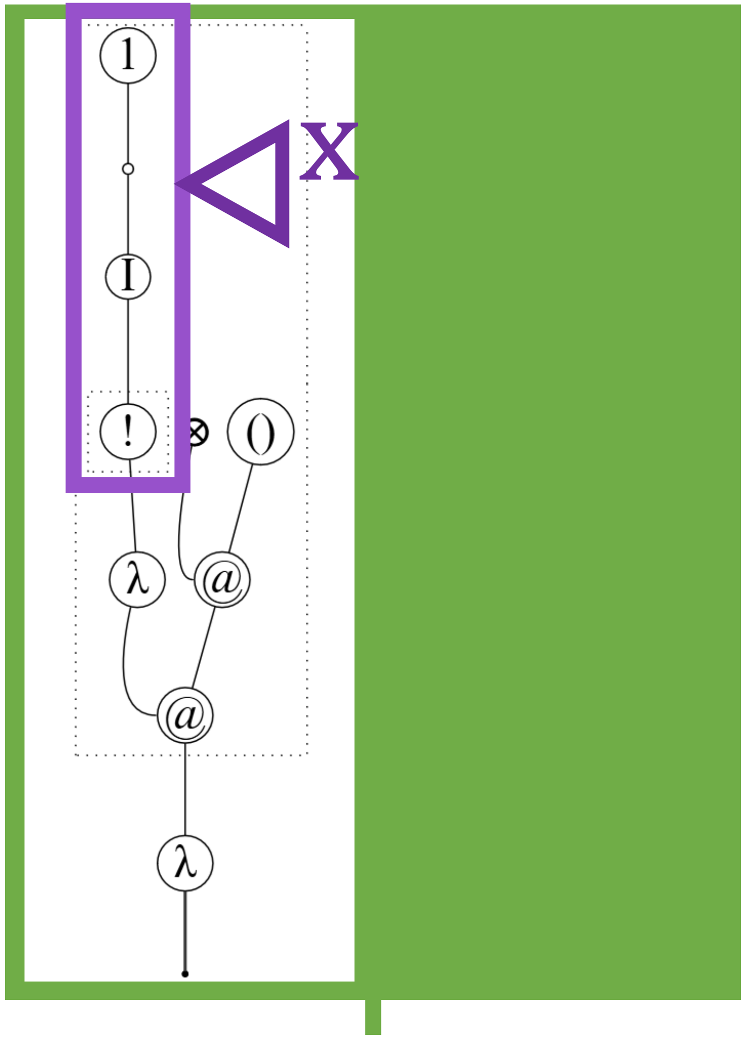


$\simeq^{?}$

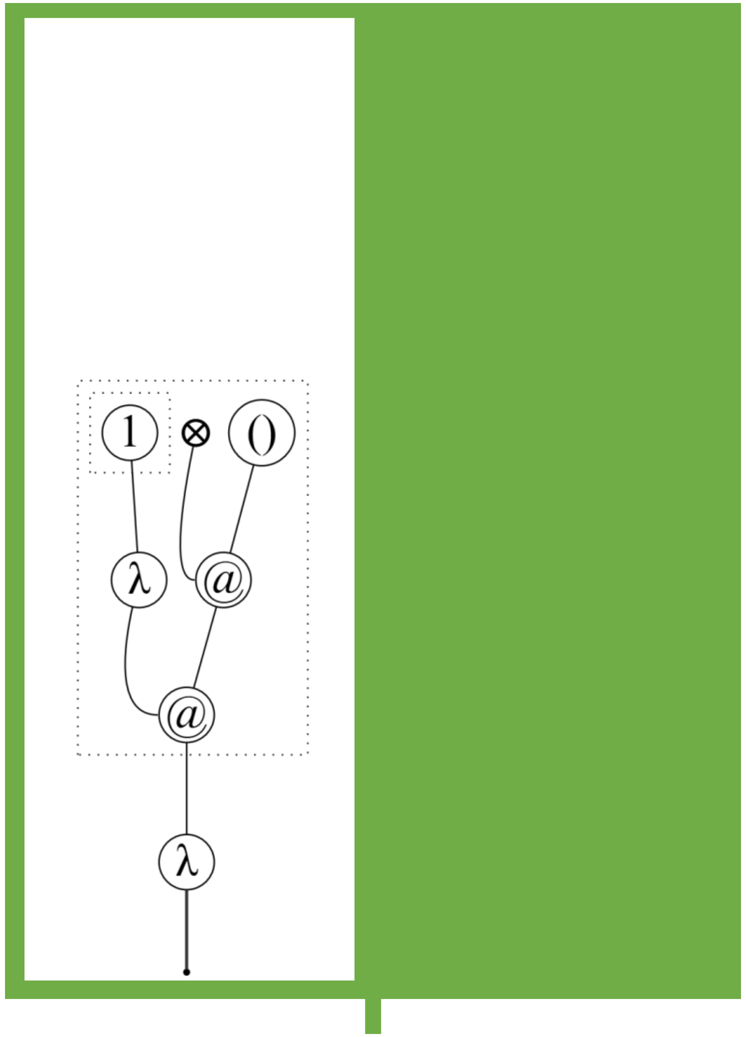


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

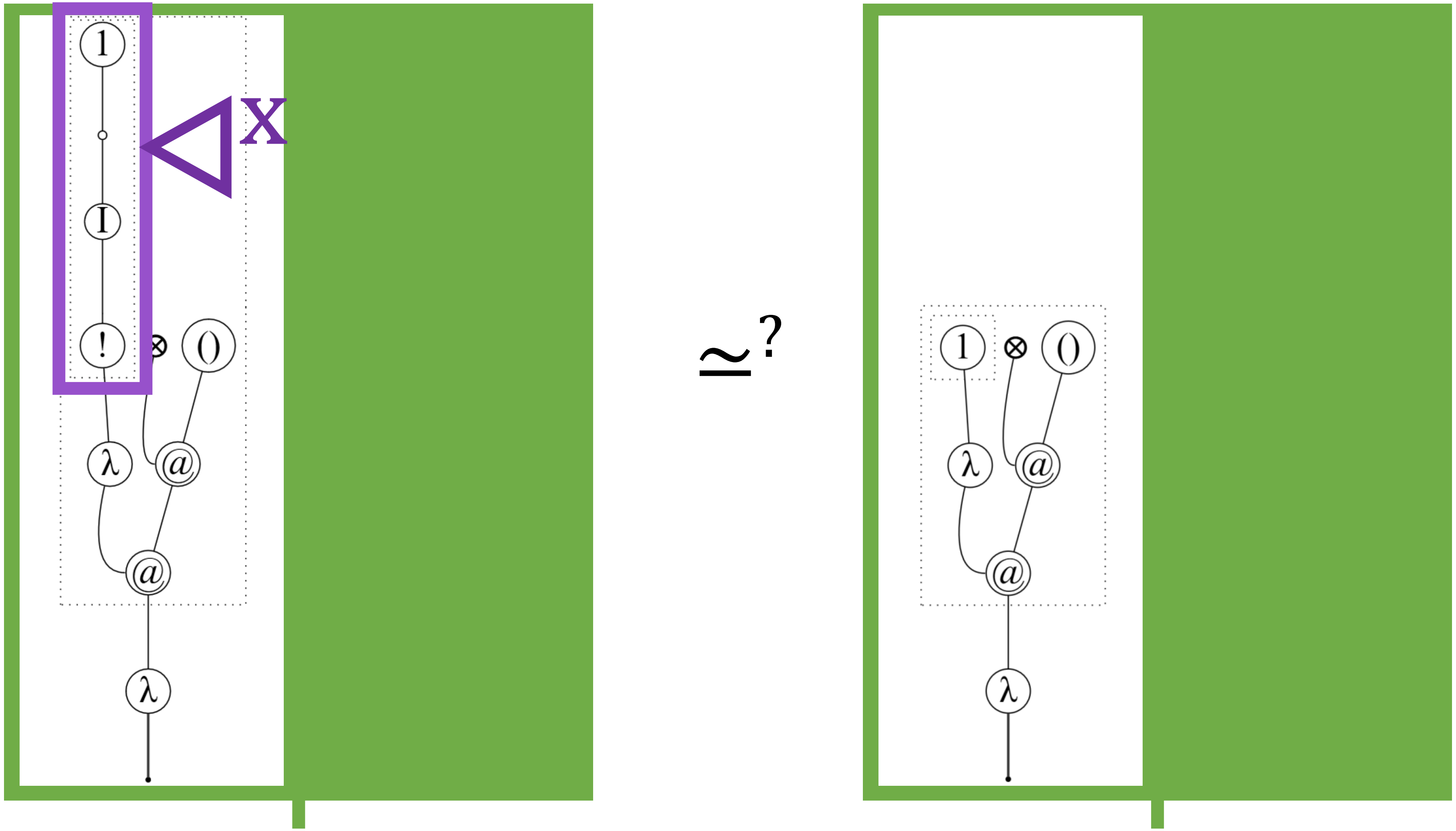


$\simeq^{?}$



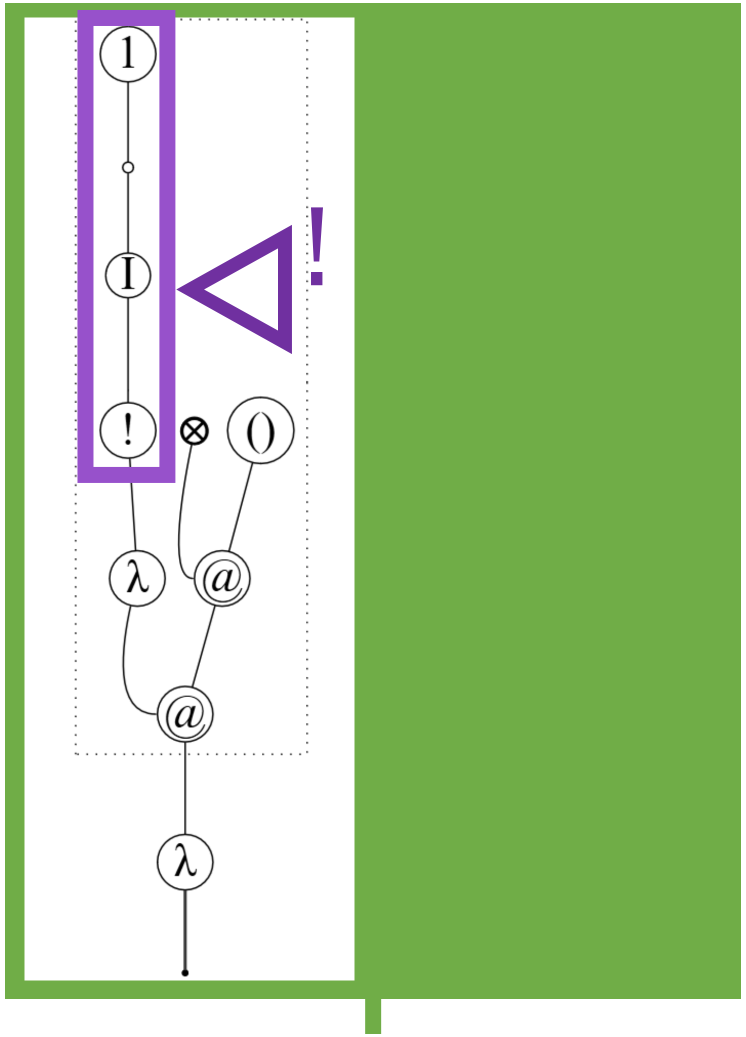
An Application of The Characterisation Theorem

$$\forall \mathcal{C}.\, \mathcal{C}[(\lambda x.\lambda f.(\lambda z.!x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} \mathcal{C}[\lambda g.(\lambda y.1)(g\,())]$$

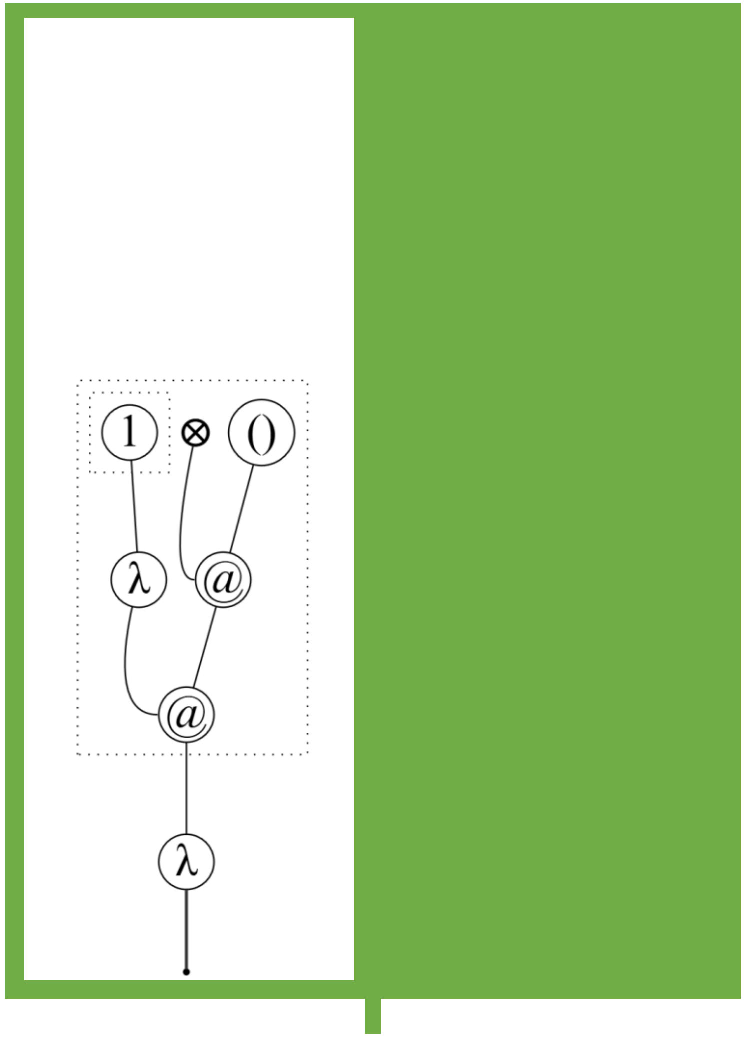


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\mathrm{ref}\,1)] \simeq^{?} C[\lambda g.(\lambda y.1)(g\,())]$$

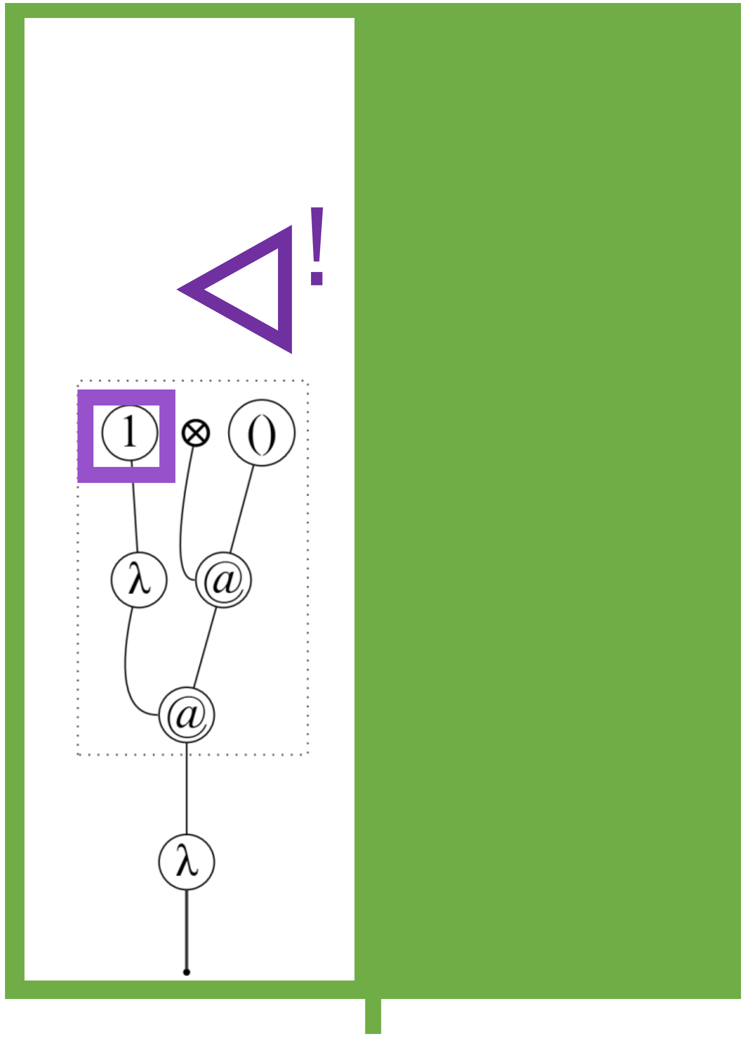


$\simeq^{?}$

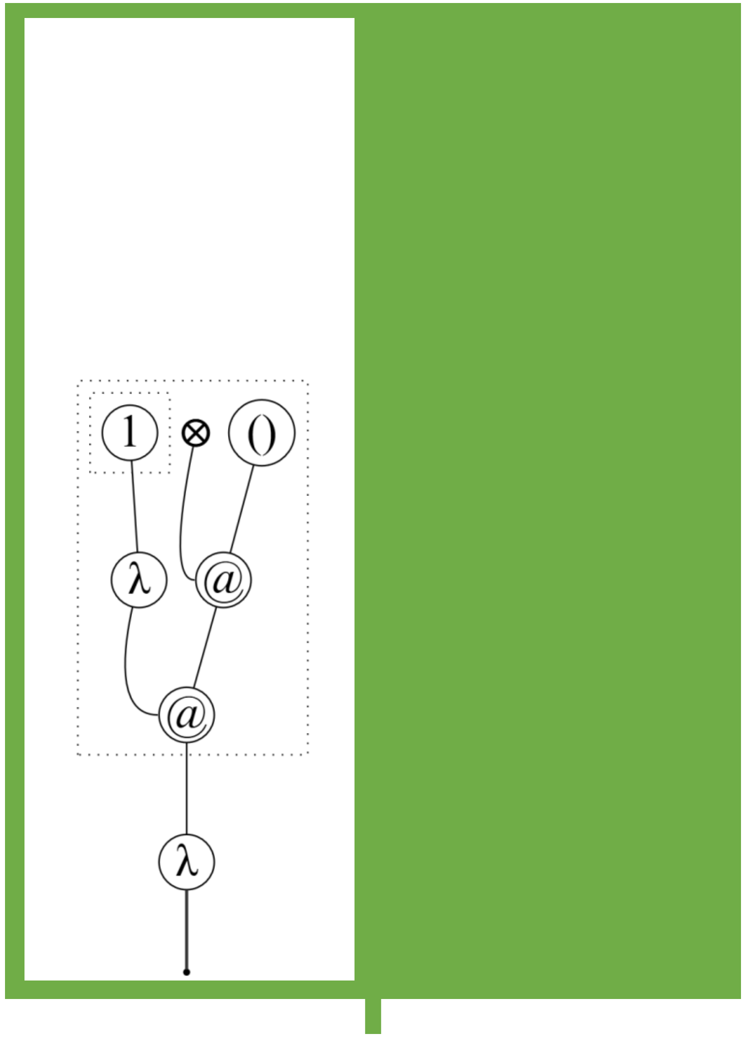


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.!\,x)(f\,()))(\mathrm{ref}\,1)] \ \simeq^{?} \ C[\lambda g.(\lambda y.1)(g\,())]$$

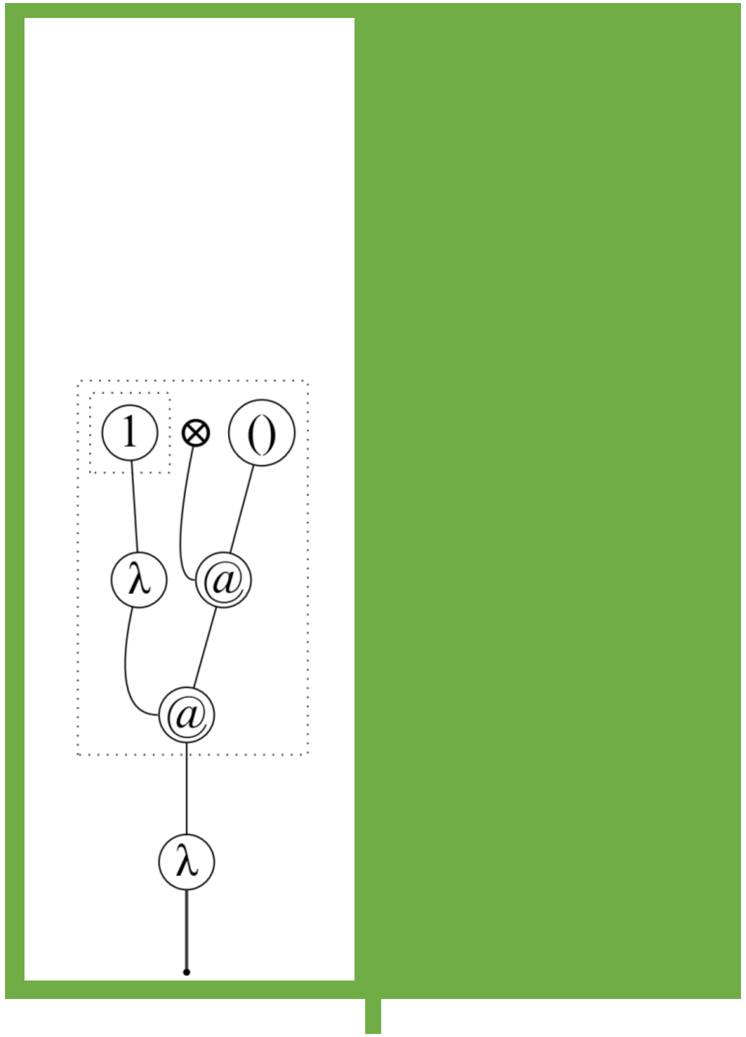


$\simeq^{?}$

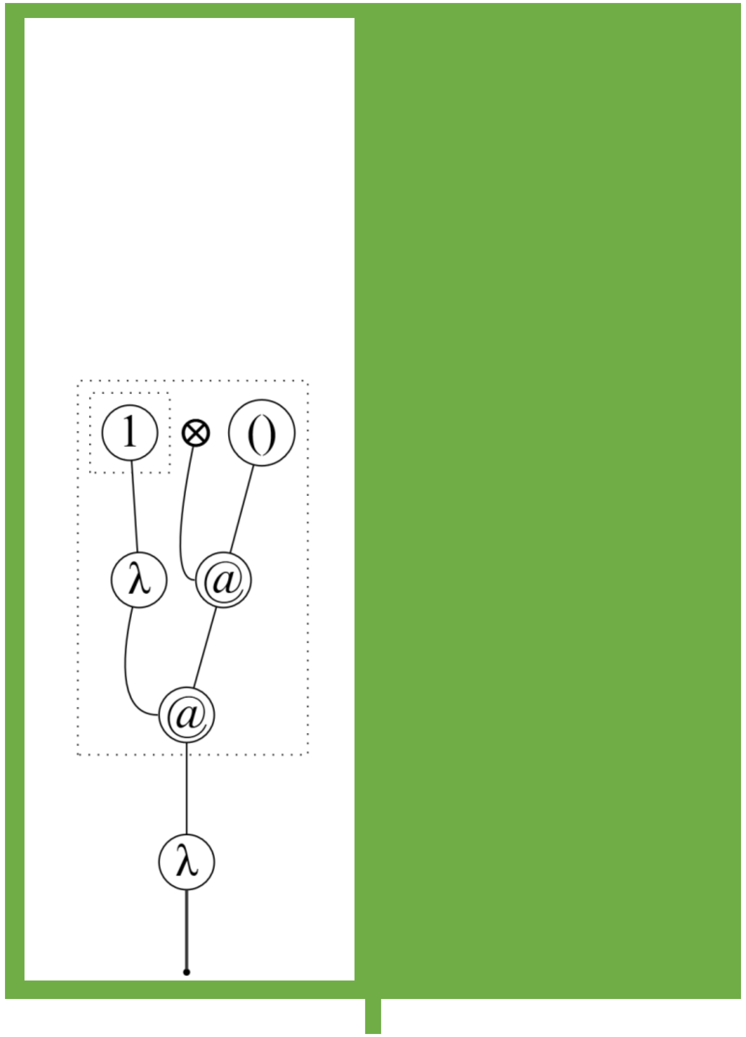


An Application of The Characterisation Theorem

$$\forall C.\, C[(\lambda x.\lambda f.(\lambda z.\,!\,x)(f\,())) (\mathrm{ref}\,1)] \simeq^{\checkmark} C[\lambda g.(\lambda y.1)(g\,())]$$

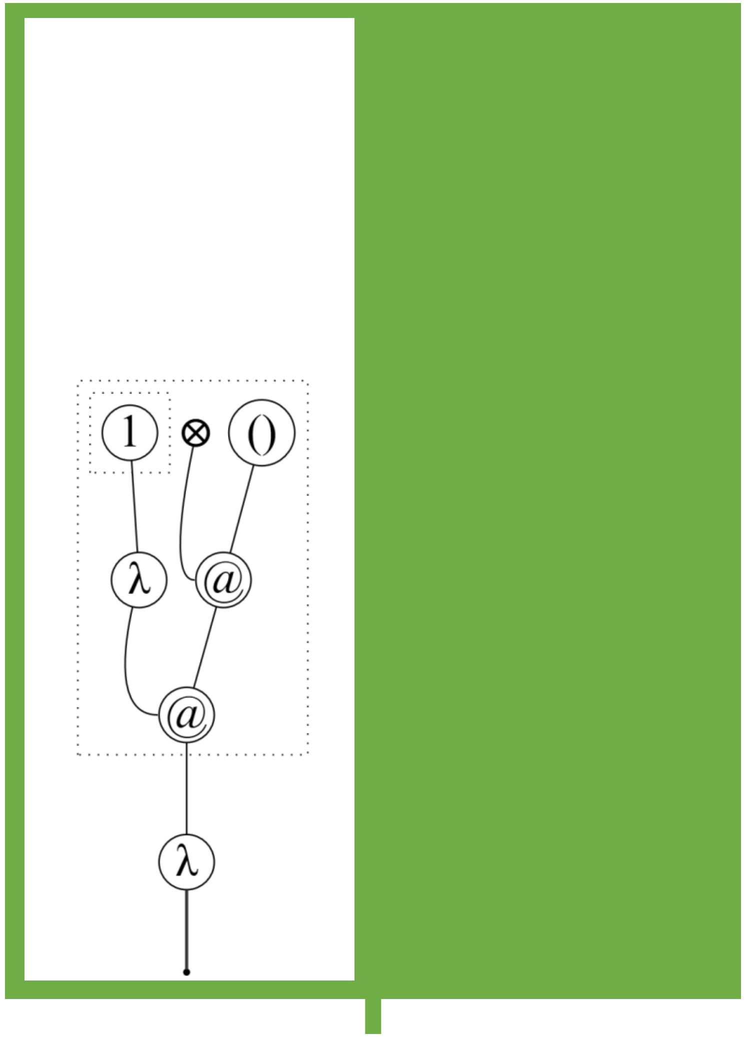


$\simeq^{\checkmark}$

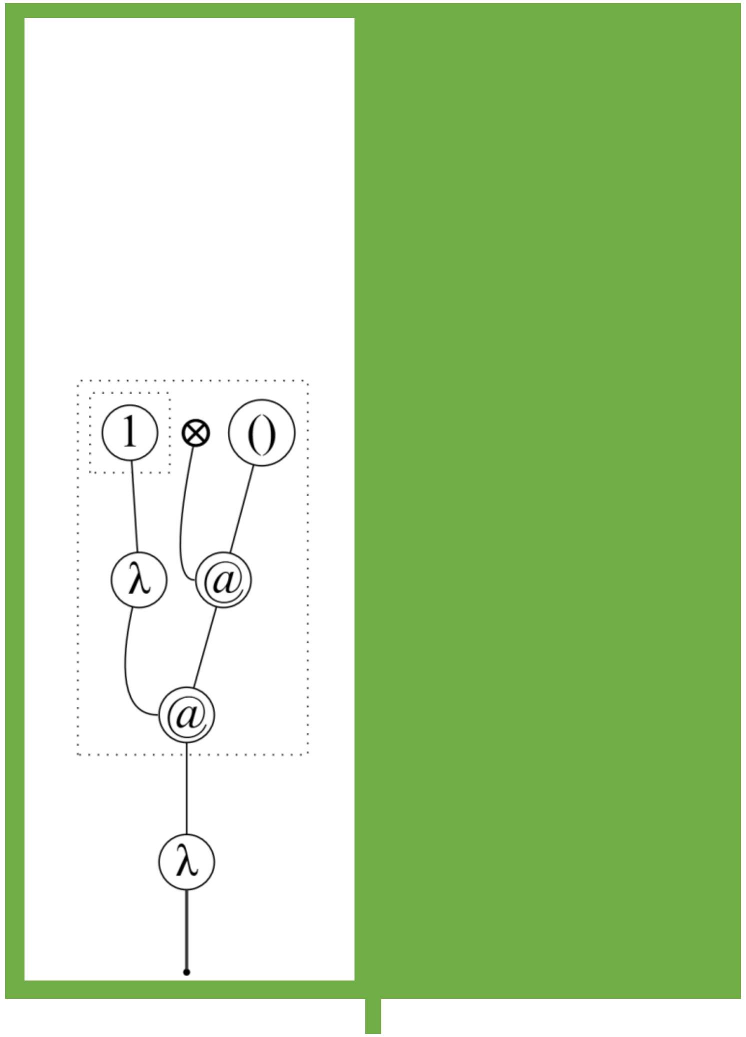


An Application of The Characterisation Theorem

$$\big(\lambda x.\lambda f.(\lambda z.!x)(f\,())\big)(\mathrm{ref}\,1) \equiv^{\checkmark} \lambda g.(\lambda y.1)(f\,())$$



$\simeq^{\checkmark}$



Closing Remarks & Further Work

Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

SPARTAN

Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

Spartan

Parameterisation of contexts for observational equivalence ✓

Binding-free & Robustness

Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

SPARTAN

Parameterisation of contexts for observational equivalence ✓

Binding-free & Robustness

Characterisation of effects from POV of equational properties of the language ✓

Characterisation Theorem

Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

SPARTAN

Parameterisation of contexts for observational equivalence ✓

Binding-free & Robustness

Characterisation of effects from POV of equational properties of the language ✓

Characterisation Theorem

Control operations ?

Non-deterministic operations ?

Concurrency ?

Type system ?

Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

SPARTAN

Parameterisation of contexts for observational equivalence ✓

Binding-free & Robustness

Characterisation of effects from POV of equational properties of the language ✓

Characterisation Theorem

Control operations ?

Non-deterministic operations ?

Concurrency ?

Type system ?

SPARTAN Visualiser & Paper:
tnttodda.github.io/Spartan-Visualiser