# A Language-Independent Proof System for Mutual Program Equivalence [1]

Ștefan Ciobâcă[1] Dorel Lucanu[1]

*Vlad Rusu*[2] Grigore Roșu[1,3]

Alexandru Ioan Cuza University, Romania

Inria Lille, France

University of Illinois at Urbana-Champaign, USA

[1]This paper is supported by the European Sectorial Operational Programme Human Resource Development (SOP HRD), and by the Romanian Government under the contract number POSDRU/159/1.5/S/137750.

# Motivation

How to prove the equivalence of programs from different languages:

```
s := 0
i := 0
while (i /= n)
  s := s + i
  i := i + 1
return s
```

```
let f n a =
  if n = 0 then a
  else f (n - 1) (a + n)
in
  f n 0
```

*Mutual* equivalence: both programs either diverge, or both terminate and "compute the same result".

# Outline

- Programming Languages
- Language Aggregation
- Defining Mutual Equivalence
- Proof System & Example.

# Our Representation of Programming Languages

Programming Language = Syntax + Semantics
Syntax = $(Cfg, S, \Sigma, \mathcal{T})$

- $\Sigma$ - signature
- $S$ - set of sorts
- $Cfg \in S$ - the sort of program configurations
- $\mathcal{T}$ - $\Sigma$-model

# Example

```
Exp   ::=  Var | Int | Exp + Exp
Stmt  ::=  Var := Exp
         | skip | Stmt ; Stmt
         | if Exp then Stmt else Stmt
         | while Exp do Stmt
Code  ::=  Exp | Stmt
Cfg   ::= ⟨Code, Map{Var, Int}⟩
```

# Example

$$
\begin{array}{lll}
\texttt{Exp} & ::= & \texttt{Var} \mid \texttt{Int} \mid \texttt{Exp + Exp} \\
\texttt{Stmt} & ::= & \texttt{Var := Exp} \\
 & \mid & \texttt{skip} \mid \texttt{Stmt ; Stmt} \\
 & \mid & \texttt{if Exp then Stmt else Stmt} \\
 & \mid & \texttt{while Exp do Stmt} \\
\texttt{Code} & ::= & \texttt{Exp} \mid \texttt{Stmt} \\
\texttt{Cfg} & ::= & \langle \texttt{Code}, \texttt{Map}\{\texttt{Var}, \texttt{Int}\} \rangle
\end{array}
$$

$$
\begin{aligned}
\Sigma = \{ \quad & \_;\_ : \texttt{Stmt} \times \texttt{Stmt} \to \texttt{Stmt}, \\
& \texttt{if\_then\_else\_} : \texttt{Exp} \times \texttt{Stmt} \times \texttt{Stmt} \to \texttt{Stmt}, \\
& \langle \_, \ \_ \rangle : \texttt{Code} \times \texttt{Map}\{\texttt{Var}, \texttt{Int}\} \to \texttt{Cfg}, \ldots \}
\end{aligned}
$$

# Example

$$
\begin{array}{lll}
\texttt{Exp} & ::= & \texttt{Var} \mid \texttt{Int} \mid \texttt{Exp} + \texttt{Exp} \\
\texttt{Stmt} & ::= & \texttt{Var} := \texttt{Exp} \\
 & \mid & \texttt{skip} \mid \texttt{Stmt} \ ; \ \texttt{Stmt} \\
 & \mid & \texttt{if Exp then Stmt else Stmt} \\
 & \mid & \texttt{while Exp do Stmt} \\
\texttt{Code} & ::= & \texttt{Exp} \mid \texttt{Stmt} \\
\texttt{Cfg} & ::= & \langle \texttt{Code}, \ \texttt{Map}\{\texttt{Var}, \texttt{Int}\} \rangle
\end{array}
$$

$$
\begin{aligned}
\Sigma = \{ \quad & \_;\_ : \texttt{Stmt} \times \texttt{Stmt} \to \texttt{Stmt}, \\
& \texttt{if\_then\_else\_} : \texttt{Exp} \times \texttt{Stmt} \times \texttt{Stmt} \to \texttt{Stmt}, \\
& \langle \_, \ \_ \rangle : \texttt{Code} \times \texttt{Map}\{\texttt{Var}, \texttt{Int}\} \to \texttt{Cfg}, \ldots \}
\end{aligned}
$$

$$
[\![ t_1 ; t_2 ]\!] \tau \equiv [\![ t_1 ]\!] \tau ; [\![ t_2 ]\!] \tau
$$

# Semantics of Programming Languages

Syntax $= (Cfg, S, \Sigma, \mathcal{T})$
Semantics $= A$

- $A$ - set of rewrite rules (the operational semantics)

# Semantics of Programming Languages

Syntax $= (Cfg, S, \Sigma, \mathcal{T})$
Semantics $= A$

- $A$ - set of rewrite rules (the operational semantics)

- $\langle \texttt{skip;}s\texttt{,}\ env\rangle \Rightarrow \langle s\texttt{,}\ env\rangle$
- $\langle \texttt{if i then}\ s_1\ \texttt{else}\ s_2\texttt{,}\ env\rangle \wedge env[i] \neq 0 \Rightarrow \langle s_1\texttt{,}\ env\rangle$
- $\ldots$

# Semantics of Programming Languages

Syntax $= (Cfg, S, \Sigma, \mathcal{T})$
Semantics $= A$

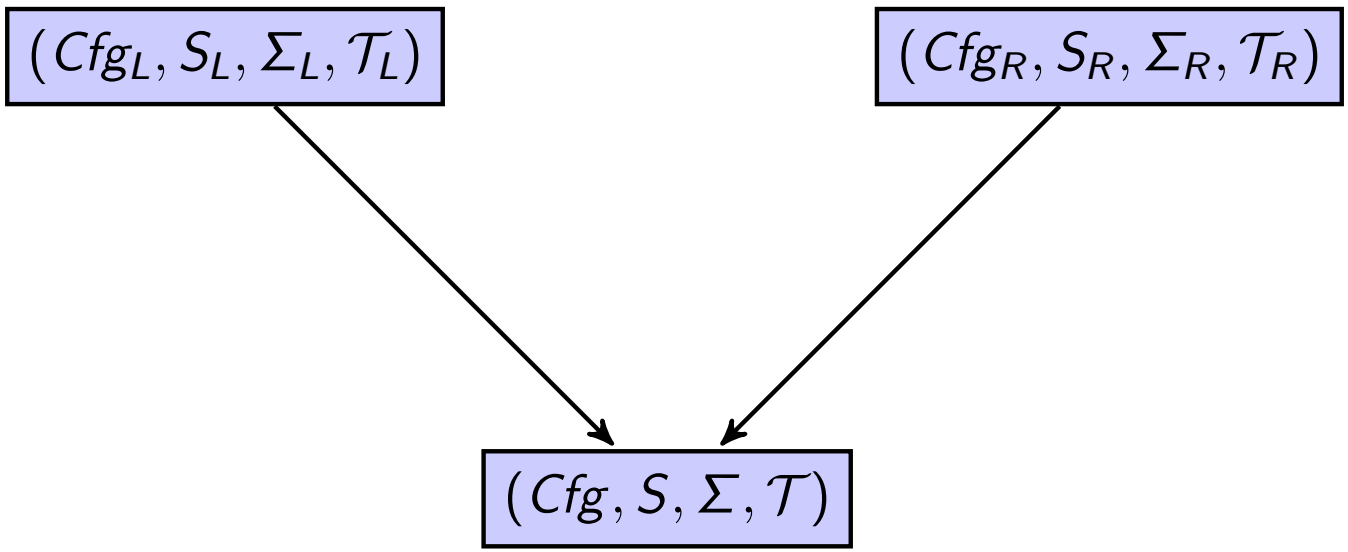
- $A$ - set of rewrite rules (the operational semantics)

- $\langle \texttt{skip;}s\texttt{,}\ env \rangle \Rightarrow \langle s\texttt{,}\ env \rangle$
- $\langle \texttt{if i then}\ s_1\ \texttt{else}\ s_2\texttt{,}\ env \rangle \wedge env[i] \neq 0 \Rightarrow \langle s_1\texttt{,}\ env \rangle$
- ...

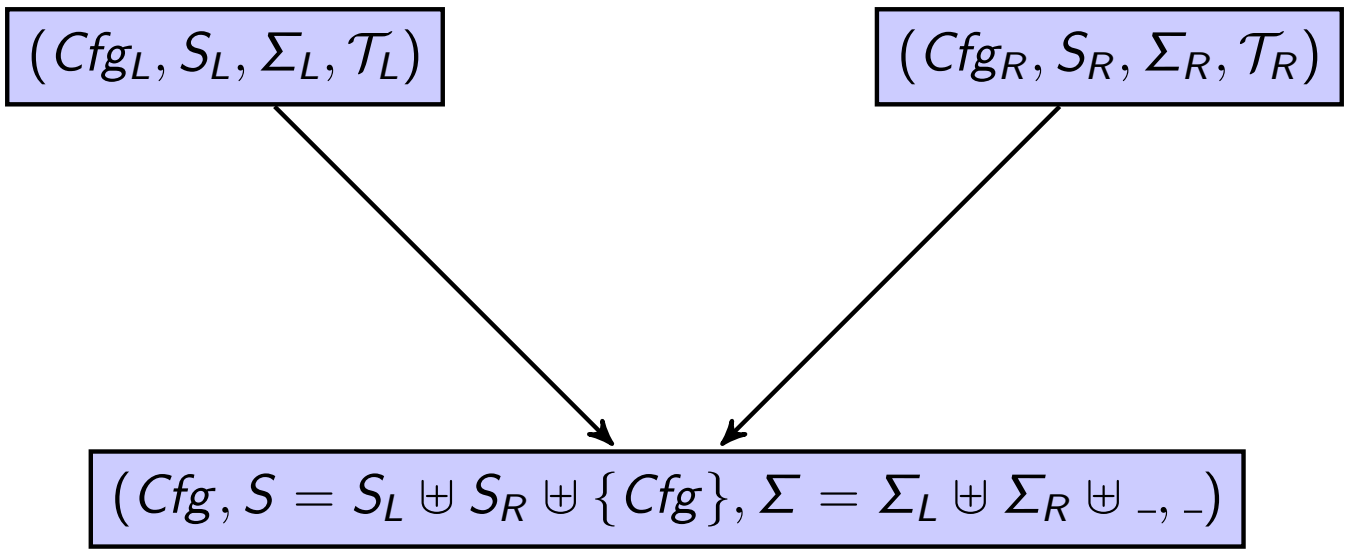
  Can *faithfully* encode most operational semantics styles (small step, big step, reduction with evaluation contexts, ...)

  Have been used for defining complete semantics of real languages: C [POPL2012] and Java [POPL2014].

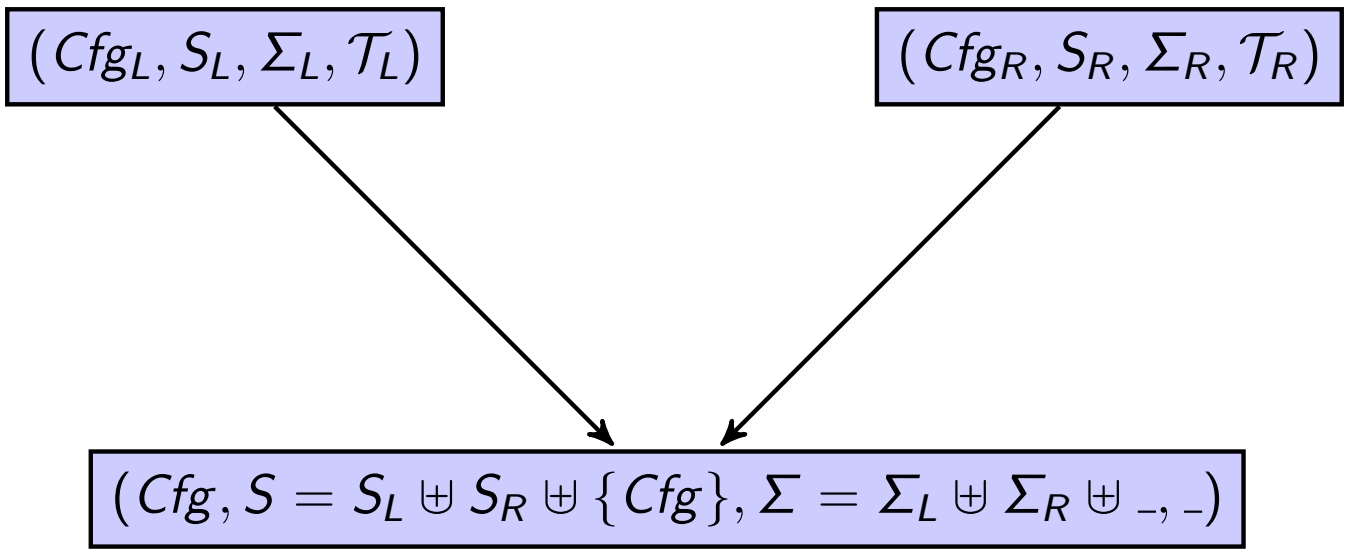
# Programing Language Aggregation



# First Idea: Disjoint Union

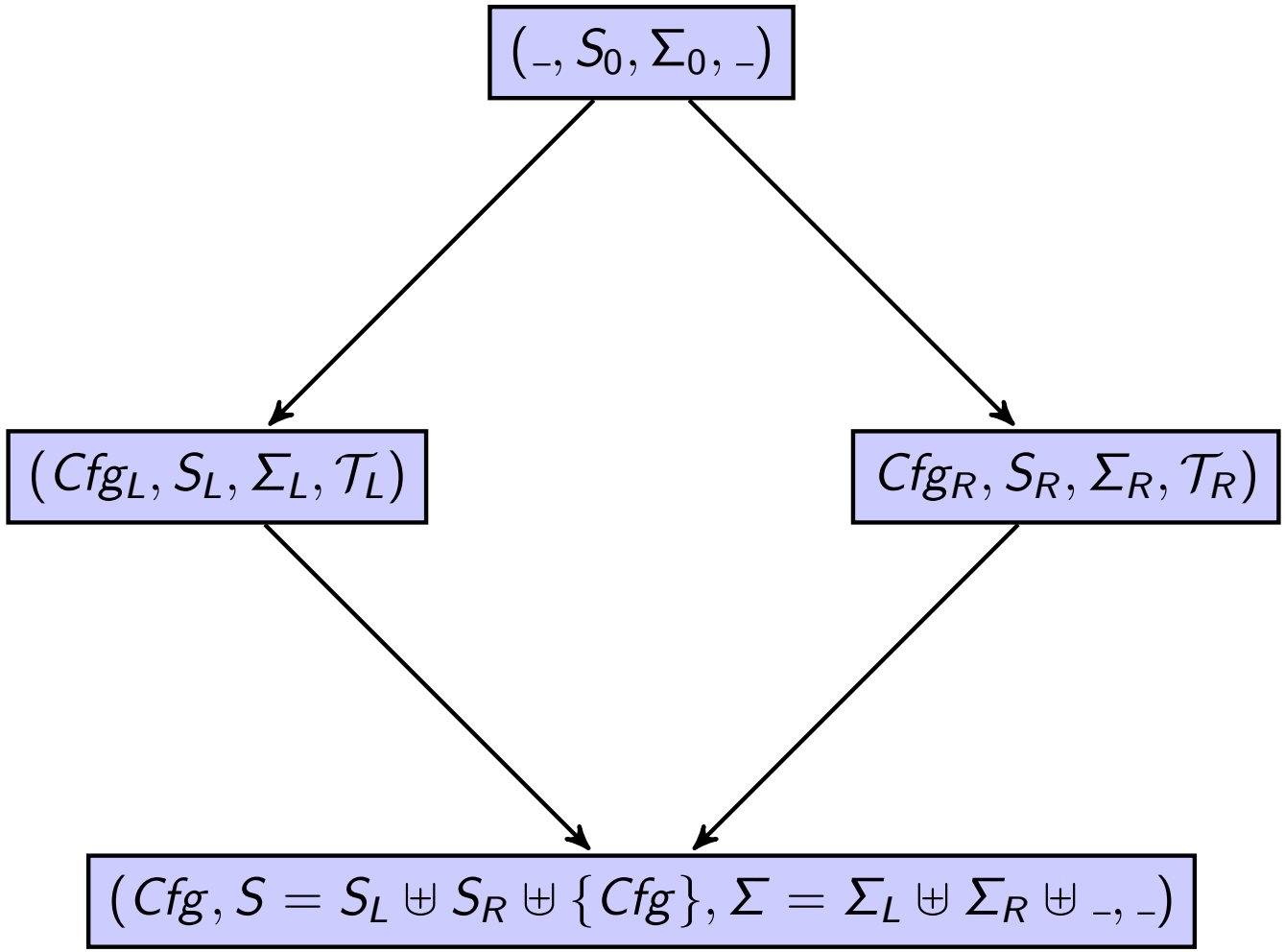


# First Idea: Disjoint Union

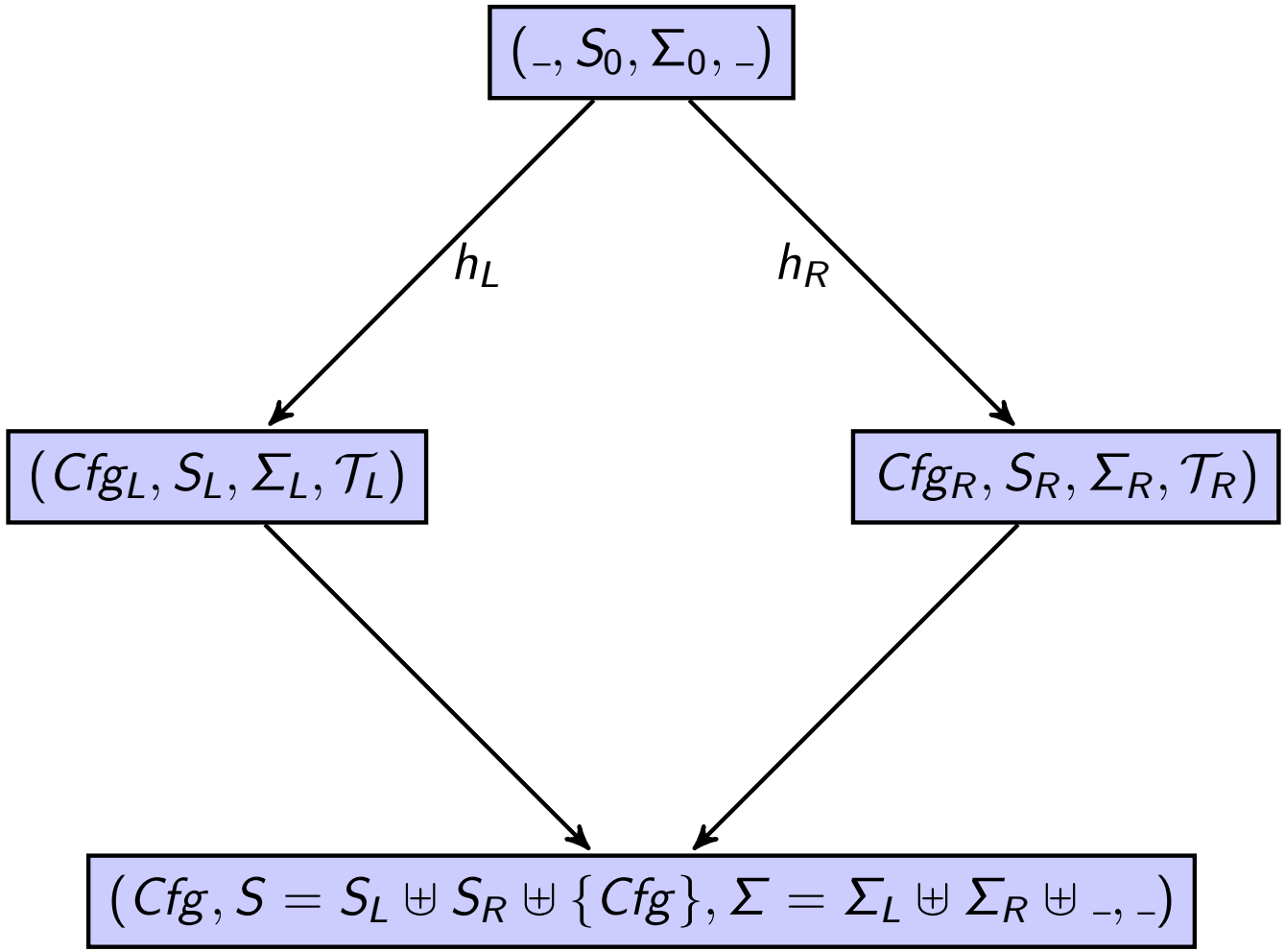



No relations between $Int_L$ and $Int_R$!

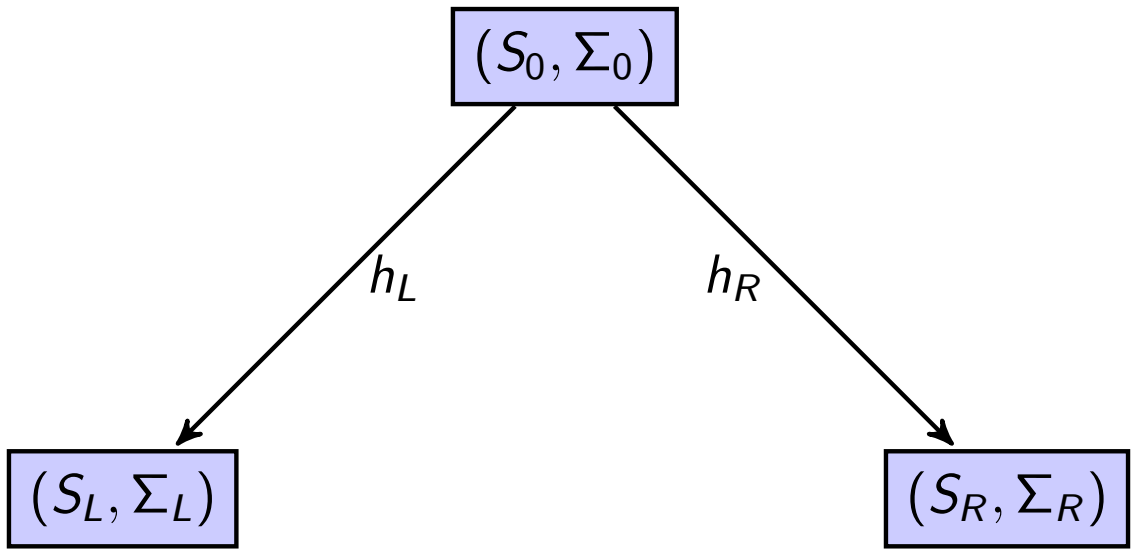
# Second Idea - Sharing Common Parts



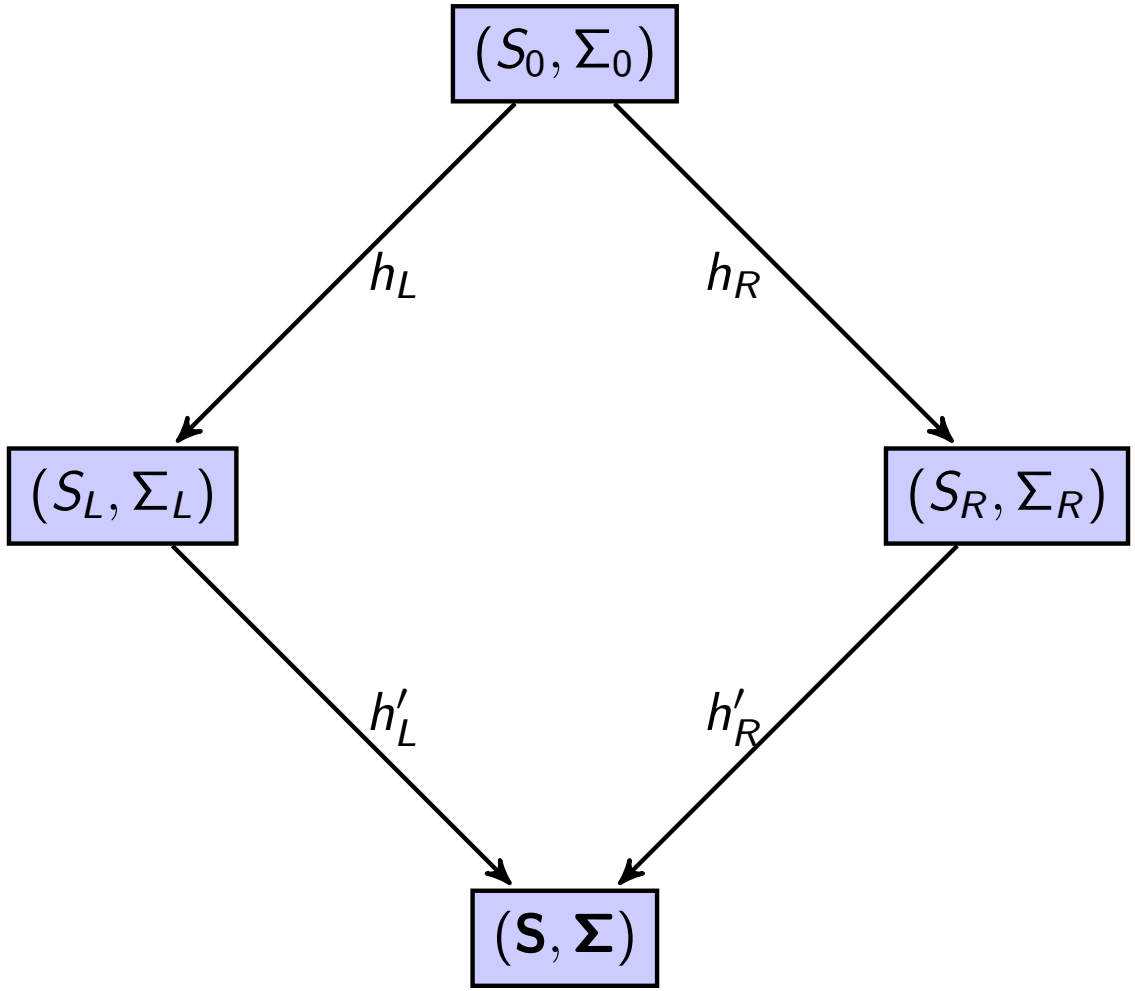
# Second Idea - Sharing Common Parts



# (1) Using the Push-out Theorem

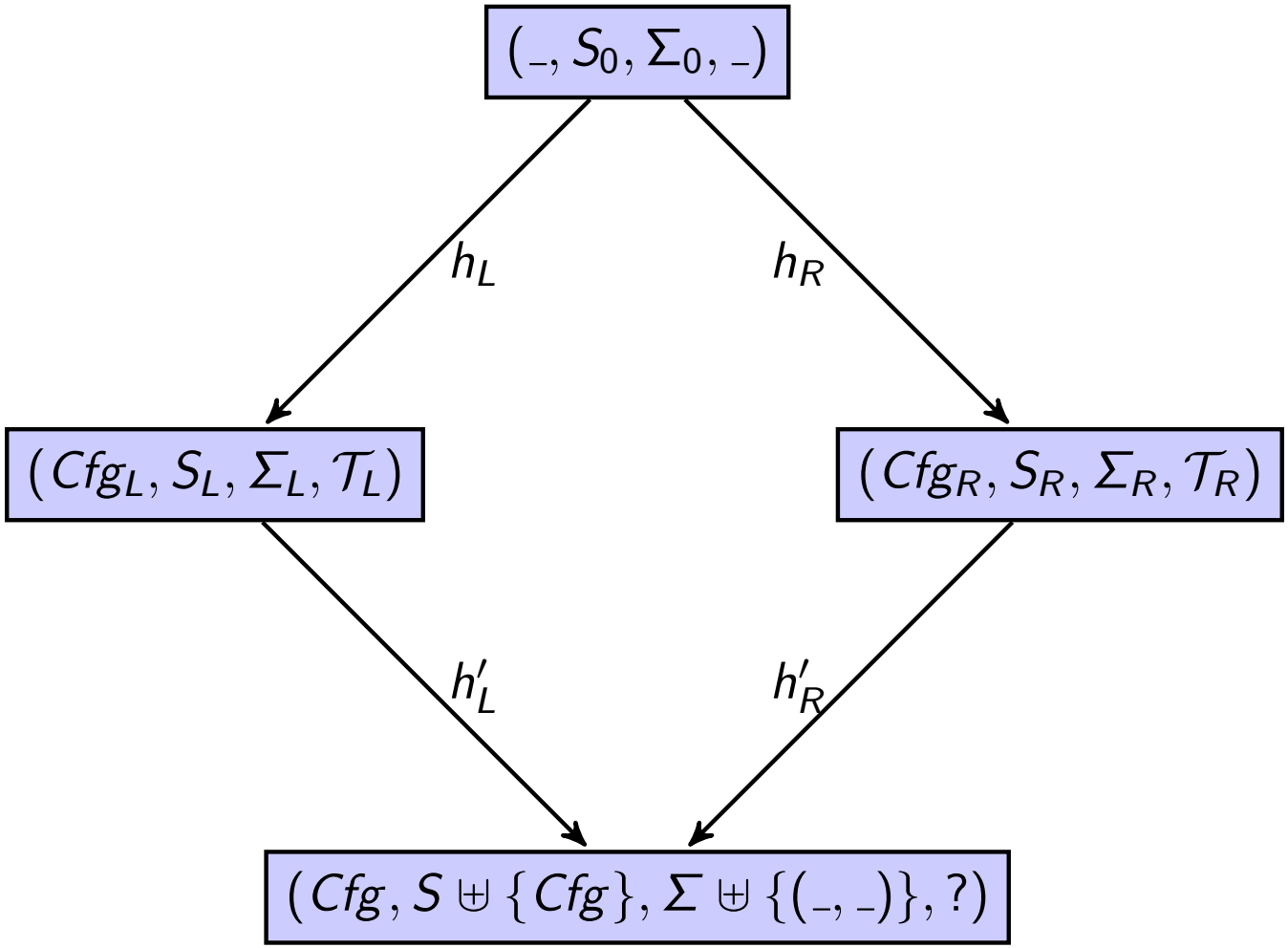


# (1) Using the Push-out Theorem

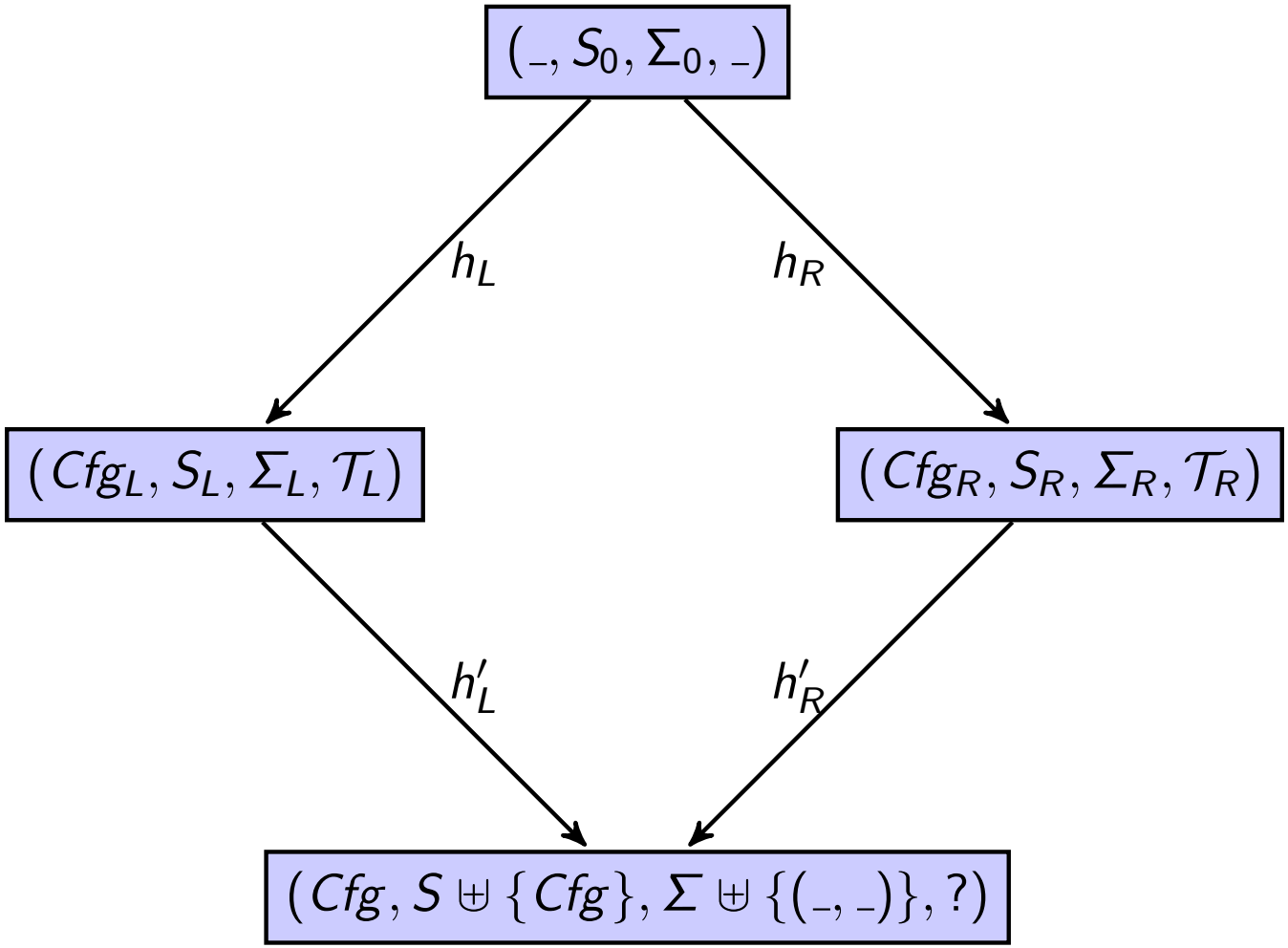



- (commutativity) $h'_L(h_L(x)) = h'_R(h_R(x))$
- (minimality) $(S, \Sigma)$ is smallest with this property

# (2) Adding Constructor for *Cfg*

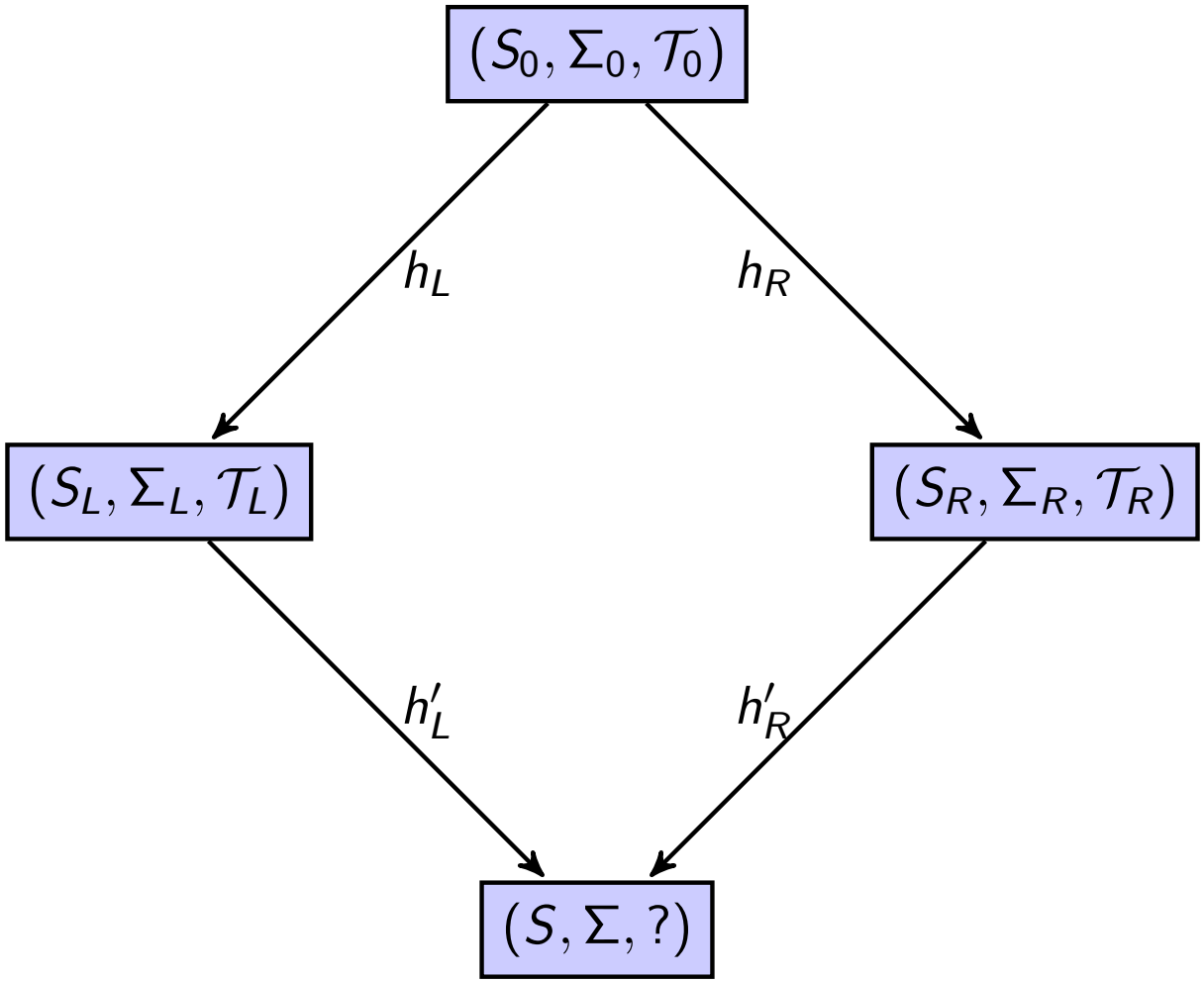


# (2) Adding Constructor for *Cfg*

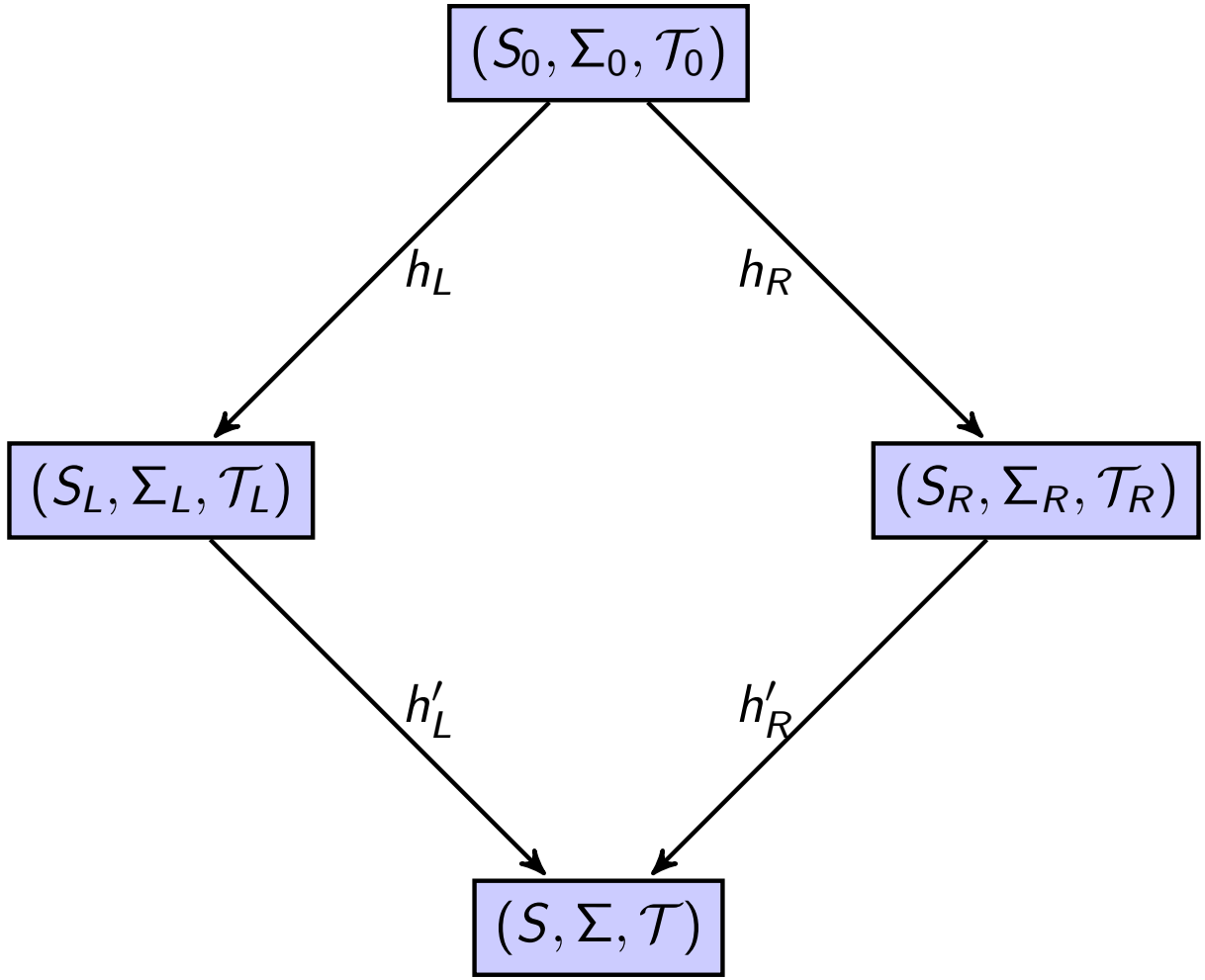



$$(\_, \_) : h'_L(Cfg_L) \times h'_R(Cfg_R) \to Cfg$$

# (3) How to Construct The Model



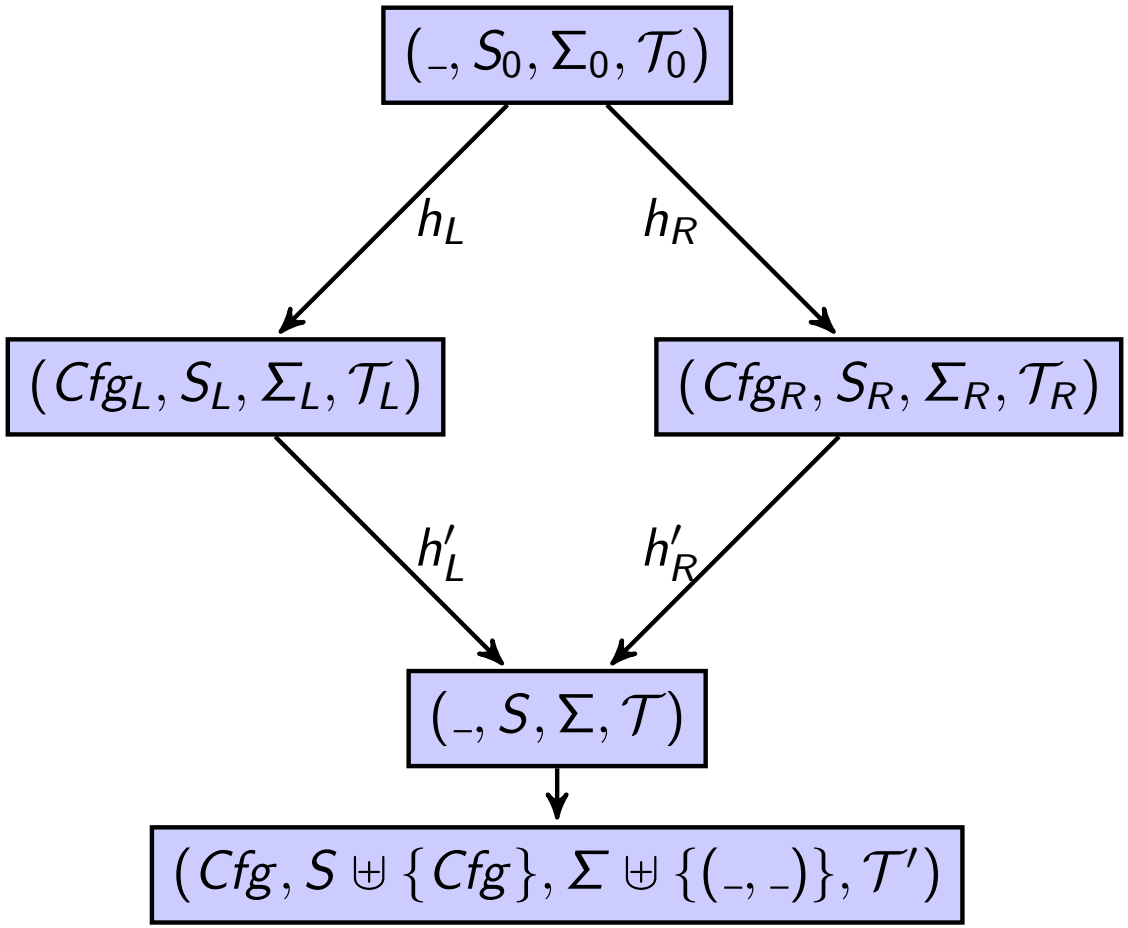
# (3) Model Amalgamation



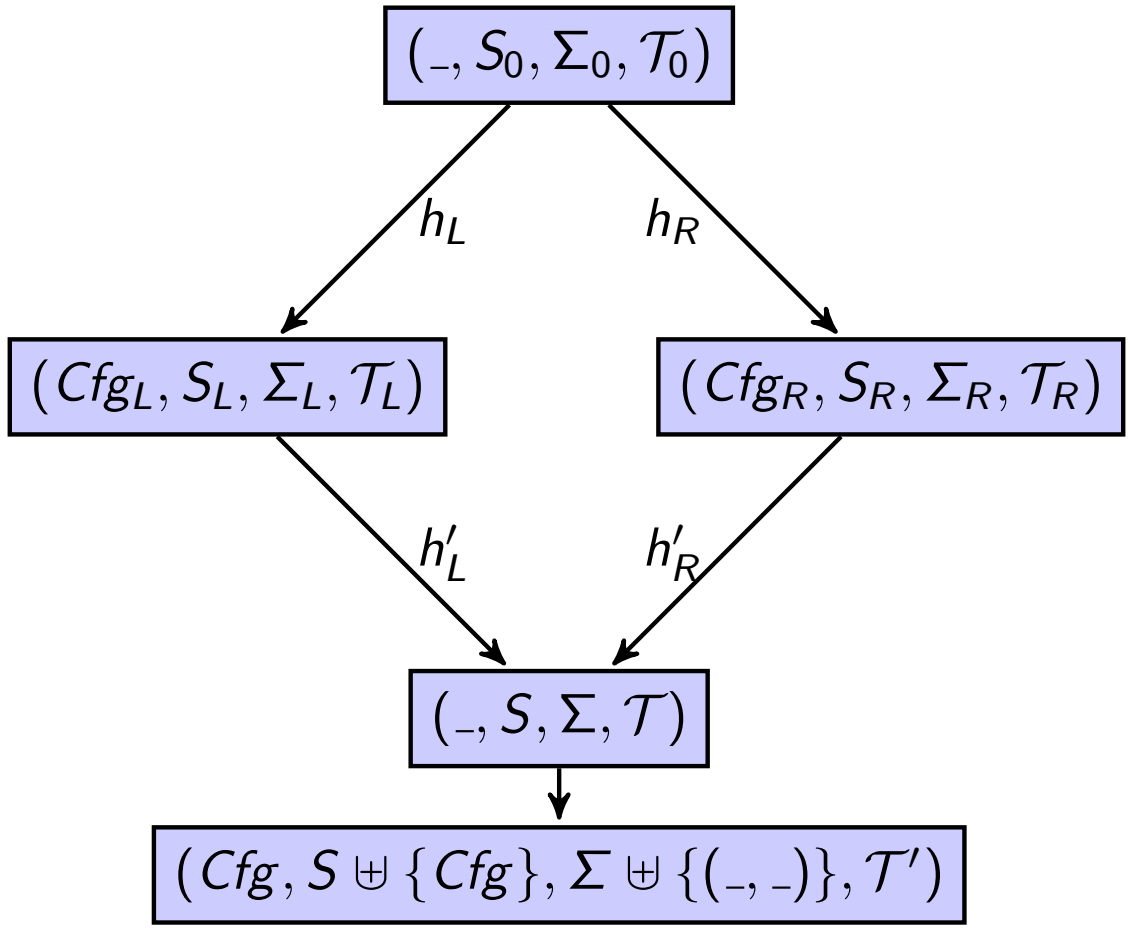

- if $\mathcal{T}_L{\restriction_{h_L}} = \mathcal{T}_0 = \mathcal{T}_R{\restriction_{h_R}}$ then

  there exists a (unique) $\mathcal{T}$ such that $\mathcal{T}{\restriction_{h'_L}} = \mathcal{T}_L$ and $\mathcal{T}{\restriction_{h'_R}} = \mathcal{T}_R$

# (4) The Model for the the *Cfg* Sort



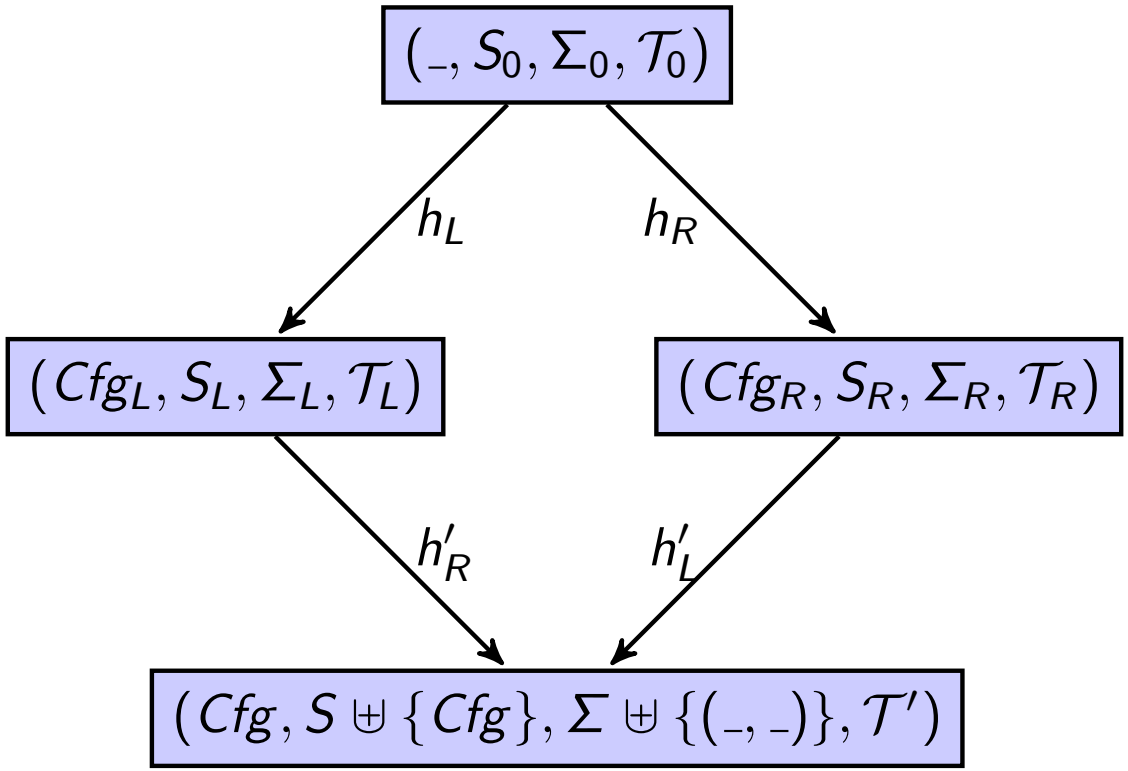
# (4) The Model for the the *Cfg* Sort




$$[\![Cfg]\!]_{\mathcal{T}'} = [\![h'_L(Cfg_L)]\!]_{\mathcal{T}} \times [\![h'_R(Cfg_R)]\!]_{\mathcal{T}}$$

$$[\![(x, y)]\!]_{\mathcal{T}'} = ([\![x]\!]_{\mathcal{T}}, [\![y]\!]_{\mathcal{T}})$$

# Aggregation: Main Property




## Proposition

*For any* $\varphi \in T_{Cfg}(Var)$ *(symbolic aggregated configuration)*
$((\gamma_L, \gamma_R), \rho) \models \varphi$ *iff* $(\gamma_L, \rho) \models proj_L(\varphi)$ *and* $(\gamma_R, \rho) \models proj_R(\varphi)$

# Defining Program Equivalence

Given:

- symbolic aggregated configuration $\varphi$
- set of symbolic aggregated configurations $E$ (e.g., encoding a set of pairs of terminated programs with equal outputs)

we define

$\varphi \Downarrow^{\infty} E$: all concrete pairs of configurations (instances of $\varphi$) either both diverge, or reach an instance of $E$.

# Mutual Equivalence Proof System

$$\textsc{Axiom}\ \frac{\varphi \in E}{\vdash \varphi \Downarrow^{\infty} E}$$

$$\textsc{Step}\ \frac{\varphi \Rightarrow^{*} \varphi' \quad \vdash \varphi' \Downarrow^{\infty} E}{\vdash \varphi \Downarrow^{\infty} E}$$

$$\textsc{Conseq}\ \frac{\models \varphi \to \varphi' \quad \vdash \varphi' \Downarrow^{\infty} E}{\vdash \varphi \Downarrow^{\infty} E}$$

$$\textsc{Case Analysis}\ \frac{\vdash \varphi \Downarrow^{\infty} E \quad \vdash \varphi' \Downarrow^{\infty} E}{\vdash \varphi \vee \varphi' \Downarrow^{\infty} E}$$

$$\textsc{Circularity}\ \frac{\vdash \varphi' \Downarrow^{\infty} E \cup \{\varphi\} \quad \varphi \Rightarrow^{+} \varphi'}{\vdash \varphi \Downarrow^{\infty} E}$$

Figure : $\varphi \Rightarrow^{*} \varphi'$ is syntactic sugar for $A_L \models pr_L(\varphi) \to^{*} pr_L(\varphi')$ and $\mathcal{A}_R \models pr_R(\varphi) \to^{*} pr_R(\varphi')$, and $\varphi \Rightarrow^{+} \varphi'$ is syntactic sugar for $A_L \models pr_L(\varphi) \to^{+} pr_L(\varphi')$ and $A_L \models pr_R(\varphi) \to^{+} pr_R(\varphi')$.

# Proving that two Collatz programs are mutually equivalent

$\mathtt{PGM}_L$ := `c := 1; LOOP`$_L$

$\mathtt{LOOP}_L$ :=
```
while (n != 1)
  c := c + 1;
  if (n % 2 != 0)
    then n := 3 * n + 1
    else n := n / 2
```

$\mathtt{PGM}_R$`(n)` := `letrec f n i = LOOP`$_R$ `in f n 0`

$\mathtt{LOOP}_R$ :=
```
if (n != 1)
  then if (n % 2 != 0)
    then f (3 * n + 1) (i + 1)
    else f (n / 2) (i + 1)
  else i
```

# Proving that two Collatz programs are mutually equivalent

$\mathtt{PGM}_L$ := `c := 1; LOOP_L`

$\mathtt{LOOP}_L$ :=

```
while (n != 1)
  c := c + 1;
  if (n % 2 != 0)
    then n := 3 * n + 1
    else n := n / 2
```

$\mathtt{PGM}_R(\mathtt{n})$ := `letrec f n i = LOOP_R in f n 0`

$\mathtt{LOOP}_R$ :=

```
if (n != 1)
  then if (n % 2 != 0)
    then f (3 * n + 1) (i + 1)
    else f (n / 2) (i + 1)
  else i
```

$$\varphi := \exists i, n.(n \neq 1 \wedge \langle \mathtt{LOOP}_L\,,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$$

# Proving that two Collatz programs are mutually equivalent

```
PGM_L   :=  c := 1; LOOP_L
LOOP_L  :=  while (n != 1)
              c := c + 1;
              if (n % 2 != 0)
                then n := 3 * n + 1
                else n := n / 2
```

```
PGM_R(n)  :=  letrec f n i = LOOP_R in f n 0
  LOOP_R  :=  if (n != 1)
                then if (n % 2 != 0)
                  then f (3 * n + 1) (i + 1)
                  else f (n / 2) (i + 1)
                else i
```

$$\varphi := \exists i, n.(n \neq 1 \wedge \langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$$

| | | | | |
|---|---|---|---|---|
| 1. | $\vdash (\langle \mathtt{skip},\ \mathtt{c} \mapsto i, \_ \rangle, \langle i \rangle)$ | $\Downarrow^{\infty}$ | $E$ | Ax |
| 2. | $\vdash (\langle \mathtt{skip},\ \mathtt{c} \mapsto i, \_ \rangle, \langle i \rangle)$ | $\Downarrow^{\infty}$ | $E \cup \{\varphi\}$ | Ax |
| 3. | $\vdash (\langle \mathtt{skip},\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle i \rangle)$ | $\Downarrow^{\infty}$ | $E$ | Con(1) |
| 4. | $\vdash (\langle \mathtt{skip},\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle i \rangle)$ | $\Downarrow^{\infty}$ | $E \cup \{\varphi\}$ | Con(2) |
| 5. | $\vdash (n = 1 \wedge \langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$ | $\Downarrow^{\infty}$ | $E$ | Step(3) |
| 6. | $\vdash (n = 1 \wedge \langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$ | $\Downarrow^{\infty}$ | $E \cup \{\varphi\}$ | Step(4) |
| 7. | $\vdash (n \neq 1 \wedge \langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$ | $\Downarrow^{\infty}$ | $E \cup \{\varphi\}$ | Ax |
| 8. | $\vdash (\langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$ | $\Downarrow^{\infty}$ | $E \cup \{\varphi\}$ | Con(CA(6, 7)) |
| 9. | $\vdash (n \neq 1 \wedge \langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$ | $\Downarrow^{\infty}$ | $E$ | Circ (8) |
| 10. | $\vdash (\langle \mathtt{LOOP}_L,\ \mathtt{n} \mapsto n, \mathtt{c} \mapsto i \rangle, \langle \mathtt{LOOP}_R \rangle)$ | $\Downarrow^{\infty}$ | $E$ | Con(CA(5, 9)) |
| 11. | $\vdash (\langle \mathtt{PGM}_L,\ \mathtt{n} \mapsto n \rangle, \langle \mathtt{PGM}_R(n) \rangle)$ | $\Downarrow^{\infty}$ | $E$ | Step (10) |

# Conclusion & Future Work

- *sound* language-independent proof system for mutual equivalence
- Implementation (w.i.p.) in the $\mathbb{K}$ framework
- Total equivalence
- Compiler correctness.