# Context-Updates Synthesis and Refinement in Chisel

*Irina Mariuca Asavoae, Mihail Asavoae, Adrian Riesco*

Namur, Belgium
LOPSTR 2017

# Mindset of Chisel

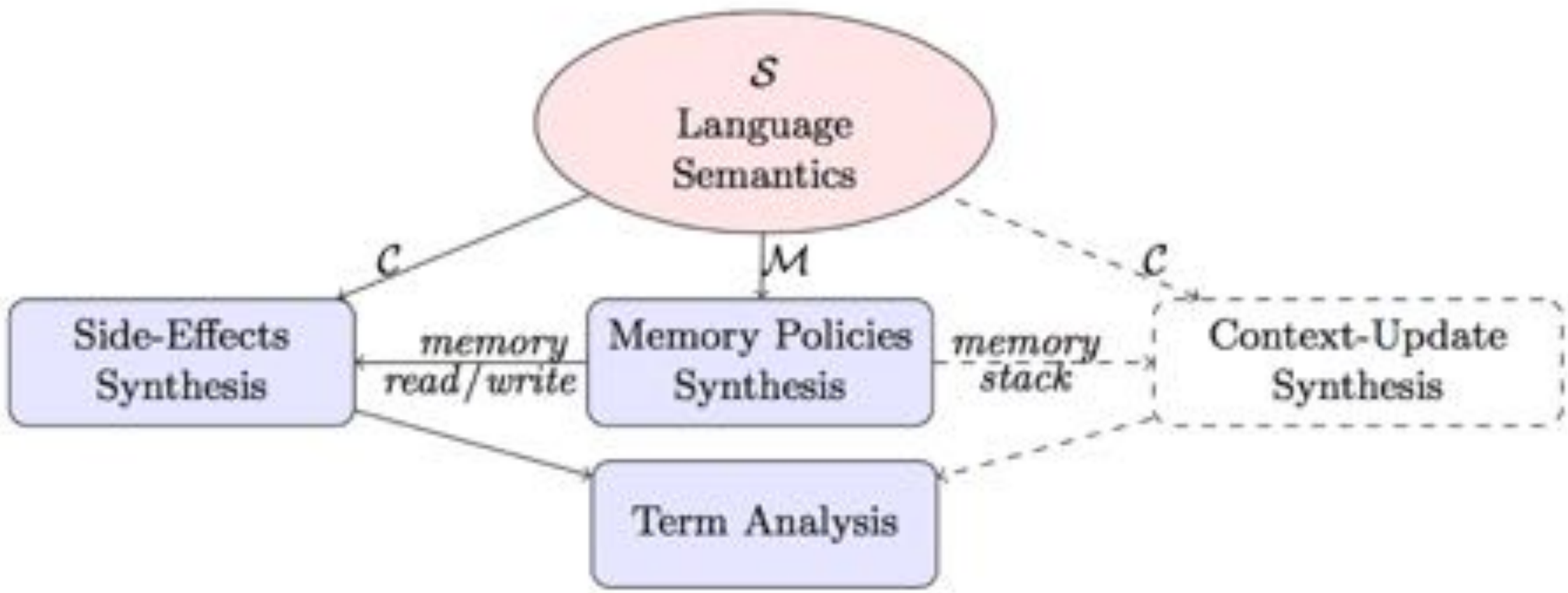
*GENERIC* ***PROGRAM SLICING****,*
i.e., programming language independent

*GENERATED FROM SEMANTICS SPECIFICATIONS,*
e.g., *Maude*'s rewriting-based algebraic programming language specifications

*appeared on Facebook wall under the title*
*"the ultimate hipster fashion"*

*In futuristic work we could use, for example,*
𝕂's formal semantics executable programming language specifications

# Gentle Intro in Semantics Based Program Slicing



# Gentle Intro in Semantics Based Program Slicing

* The scene:

# Gentle Intro in Semantics Based Program Slicing

* The victim character:
  - $\mathcal{S}em$ : the <u>semantics of the programming language</u> given as a rewriting-based algebraic specification (*in Maude*).

# Algebraic Specifications of Program Semantics

- *Sem* - snapshot of a few rules defining constructs as:

`While _ Do _` , `Local _` and `Call _`
(loop) (local variables) (function call)

```
crl [WR1] : < While b Do C, st, rw, fs > => < skip, st', rw, fs > if < b, st > => < F, st' > .

crl [WR2] : < While b Do C, st, rw, fs > => < skip, st'', rw', fs >
 if < b, st > => < T, st' > /\ < C ;(While b Do C), st', rw, fs > => < skip, st'', rw', fs > .

crl [Loc] : < Local vl, st, rw, fs > => < skip, st', rw, fs >  if st' := alloc(vl, st) .

crl [CFn] : < Call fn ( actPrms ), st, rw, fs > => < skip, st', rw', fs >
 if  fn ( Prms ) { C } fs' := fs /\  lenv := asgP(actPrms, Prms, st | mt) /\
     < C, st | lenv, rw, fs > => < skip, st' | lenv', rw', fs > .
```

<pgm:P, mem:M>

# Algebraic Specifications of Program Semantics

- *Sem* - snapshot of a few rules defining constructs in assembly:

| `addiu _ , _ , _ ;` | `jr _ ;` | `load _ , _ ;` | `jr _` |
|---|---|---|---|
| (addition) | (jump to location) | (memory load) | (function call) |

```
crl [lw] : [IIL | RS | M | PC]
 =>        [IIL | RS' | M | PC + 1]
 if (lw RI, N(RI')) := getIns(IIL, PC) /\
    I := RS [RI'] /\
    I' := M [I + N] /\
    RS' := update(RI, I', RS) .
```

```
r] : [IIL | RS | M | PC]
     [IIL | RS | M | PC']
r RI) := getIns(IIL, PC) /\
:' := RS [RI] .
```

```
if (addiu RI, RI', I) := getIns(IIL
   I' := RS [RI'] /\
   RS' := update(RI, I + I', RS) .
```

```
crl [jal] : [IIL | RS | M | PC]
 =>         [IIL | RS | M | PC + 1]
 if (jal PC') := getIns(IIL, PC) /\
    [IIL | RS | M | PC'] => [IIL | RS' | M' | PC''] /\
    nop := getIns(IIL, PC'') .
```

# Gentle Introduction in Semantics Based Slicing

- *Sem* - instance given via one *pgm* which contains the procedures/ functions `Main()`, `A(x,y)`, `B(x,y)`, `Add(a,b)`, and `Inc(z)` in both high-level and assembly languages semantics.

```
Main () {
 sum := 0;
 Local i, j;
 i := 1; j := −1;
 While i < 11 Do {
    Call A (sum, i);
    Call B (sum, j);
    Call A (j, i)
}}
A (x, y) {
 If x > 1 Then {
    Call Add (x, y);
    Call Inc (y)
}}
```

```
B (x, y) {
 If x > 0 Then {
    Call B (x + y, y)
}}
Add (a, b) {
 a := a + b
}
Inc (z) {
 Local i, j;
 i := 1; j := i;
 Call Add (z, i);
 Call Inc (j)
}
```

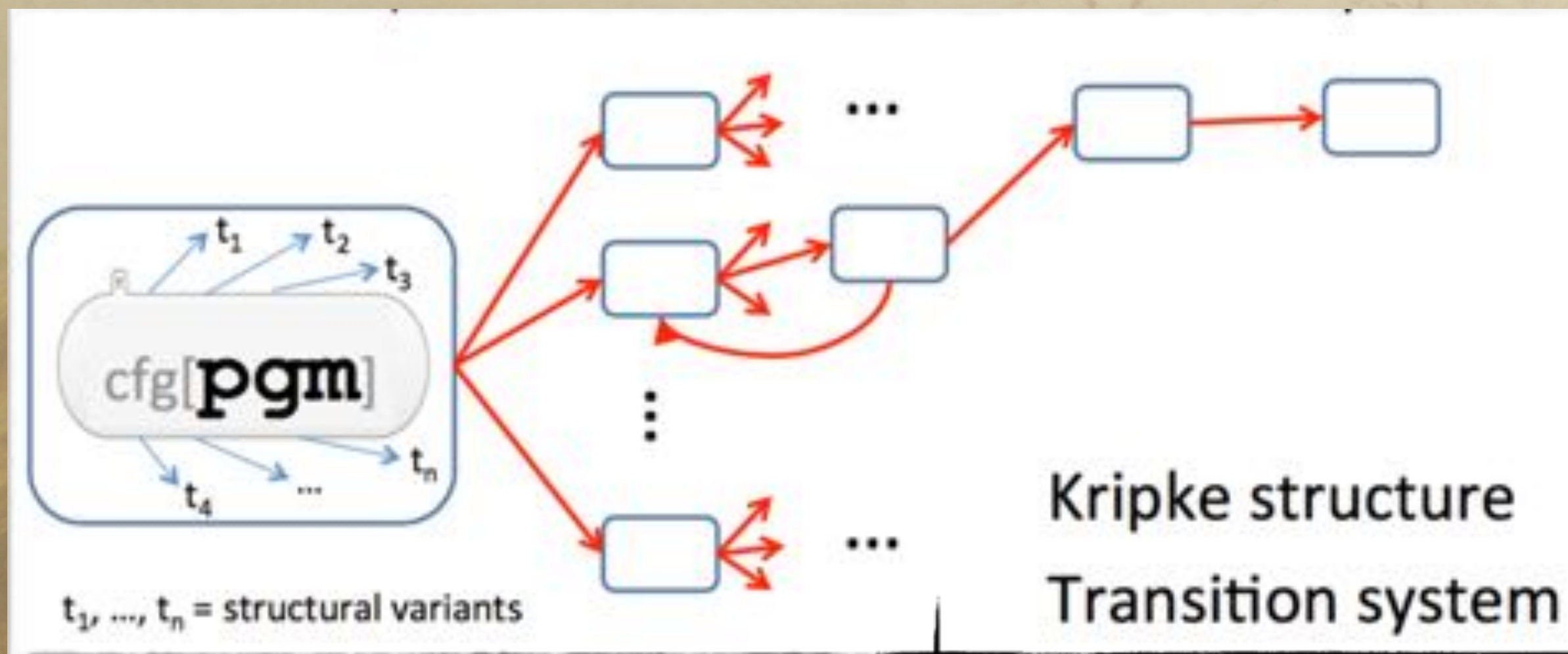
```
main:
 addiu sp,
 sp,-40
 sw ra, 36(sp)
 sw s8, 32(sp)
 move s8, sp
 sw zero,
 16(s8)
 li v0, 1
 sw v0, 20(s8)
 li v0, -1
 sw v0, 24(s8)
 j main_0x5c
 nop
```

```
main_0x2c:
 lw a0, 16(s8)
 lw a1, 20(s8)
 jal A
 nop
 lw a0, 16(s8)
 lw a1, 24(s8)
 jal B
 nop
 lw a0, 24(s8)
 lw a1, 20(s8)
 jal A
 nop
```

```
main_0x5c:
 lw v0, 20(s8)
 slti v0, v0,11
 bnez v0,
 main_0x2c
 nop
 move sp, s8
 lw ra, 36(sp)
 lw s8, 32(sp)
 addiu sp,
 sp,40
 jr ra
 nop
```

# Algebraic Specifications of Program Semantics

- *Sem* - executions of a program *pgm*:



# Algebraic Specifications of Program Semantics

- *Sem* - executions of a program *pgm*:

**Executions of *pgm* are traces of *Sem* in the ground, i.e., *memory*.**

# Generic Semantics Based Verification Mindset

* The static analysis perspective:

# Generic Semantics Based Verification Mindset

* T


-updates)

# Generic Semantics Based Program Slicing

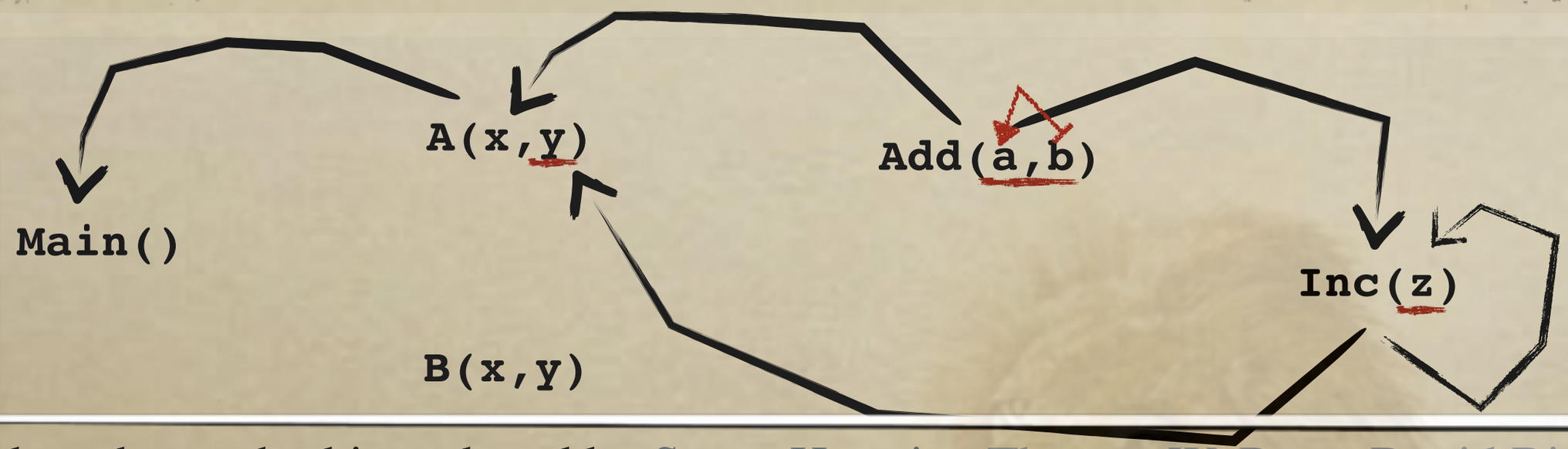
* The lead character:
  - *Sly* : the <u>slicer of *pgm* programs</u> that are executed based on *Sem*.

# The Leader of the Pride

- *Sly* - takes a term ( of *Sem* ) representing the program *pgm*:*P* and, based on the synthesis results obtained by *Syn*(*i*), slices *pgm* producing the *sliced-pgm*, i.e., the <u>*skeleton term* of *pgm*</u> (described in iFM'14):

```
Main () {
  sum := 0;
  Local i, j;
  i := 1; j := -1;
  While i < 11 Do
    Call A (sum, i);
    Call B (sum, j);
    Call A (j, i)
}

A (x, y) {
  If x > 1 Then
    Call Add(x, y);
    Call Inc (y)
}
B (x, y) {
  If x > 0 Then
    Call B(x + y, y)
}

Add (a, b) {
  a := a + b
}
Inc (z) {
  Local i, j;
  i := 1; j := i;
  Call Add (z, i);
  Call Inc (j)
}
```

# The Leader of the Pride




Based on the method introduced by Susan Horwitz, Thomas W. Reps, David Binkley in *Interprocedural Slicing Using Dependence Graphs*. PLDI 1988

```
Main () {
  sum := 0;
  Local i, j;
  i := 1; j := -1;
  While i < 11 Do
    Call A (sum, i);
    Call B (sum, j);
    Call A (j, i)
}
```

```
A (x, y) {
  If x > 1 Then
    Call Add(x, y);
    Call Inc (y)
}
B (x, y) {
  If x > 0 Then
    Call B(x + y, y)
}
```

```
Add (a, b) {
  a := a + b
}
Inc (z) {
  Local i, j;
  i := 1; j := i;
  Call Add (z, i);
  Call Inc (j)
}
```

# Comparing Note

* *Chisel* is generically <u>slicing programs</u> *pgm* based on the programming language semantics;

* *Semantics based compilation* (see ongoing/future work in *K*) is <u>slicing the programming language semantics</u> to make the execution of the program *pgm* more efficient.

# Generic Semantics Based Program Slicing

* The protagonists:
  - $\mathcal{S}yn(\mathfrak{i})$ : the $\mathfrak{i}$-feature synthesiser of the semantics $\mathcal{S}em$ .

# Slicing-Features Synthesis of Sem.

- $Syn(\mathfrak{i})$ – <u>synthesizes the $\mathfrak{i}$-feature</u> of the semantics specification $Sem$ and <u>produces language constructs</u> (terms of sort $\mathfrak{P}$) that are *directly* responsible for producing the $\mathfrak{i}$-feature in the programming language:

# Slicing-Features Synthesis of Sem.

- *Syn(side-effects)* - extracts the <u>*side-effect*</u> constructs,

  i.e., the program constructs (terms of sort $\mathcal{P}$) that are *directly*

  responsible for <u>*changes in the memory*</u> (identified by the sort $\mathcal{M}$):

# Slicing-Features Synthesis of Sem.

- Syn(side-effects) - extracts the <u>*side-effect*</u> constructs,

  i.e., the program constructs (terms of sort $\mathcal{P}$) that are *directly*

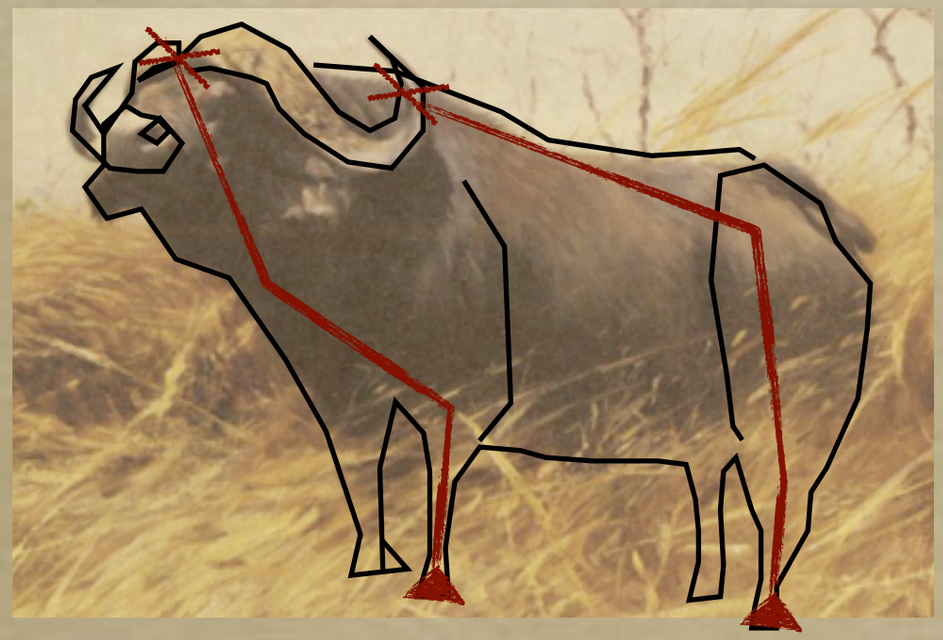
  responsible for <u>*changes in the memory*</u> (identified by the sort $\mathcal{M}$):

  - `_ := _, _ += _`
  - `Call _`
  - `...`
  - `lw _ _, sw _ _, add _ _ _, …`

Described in WADT'12 & LOPSTR'15.

# So... who is Syn(context-updates)?

- *Syn(context-updates)* - extracts the <u>*context-update*</u> constructs, i.e., the program constructs (terms of sort $\mathcal{P}$) that are *directly* responsible for <u>*stacking-up subparts of memory*</u> (stack-ops of $\mathcal{M}$):



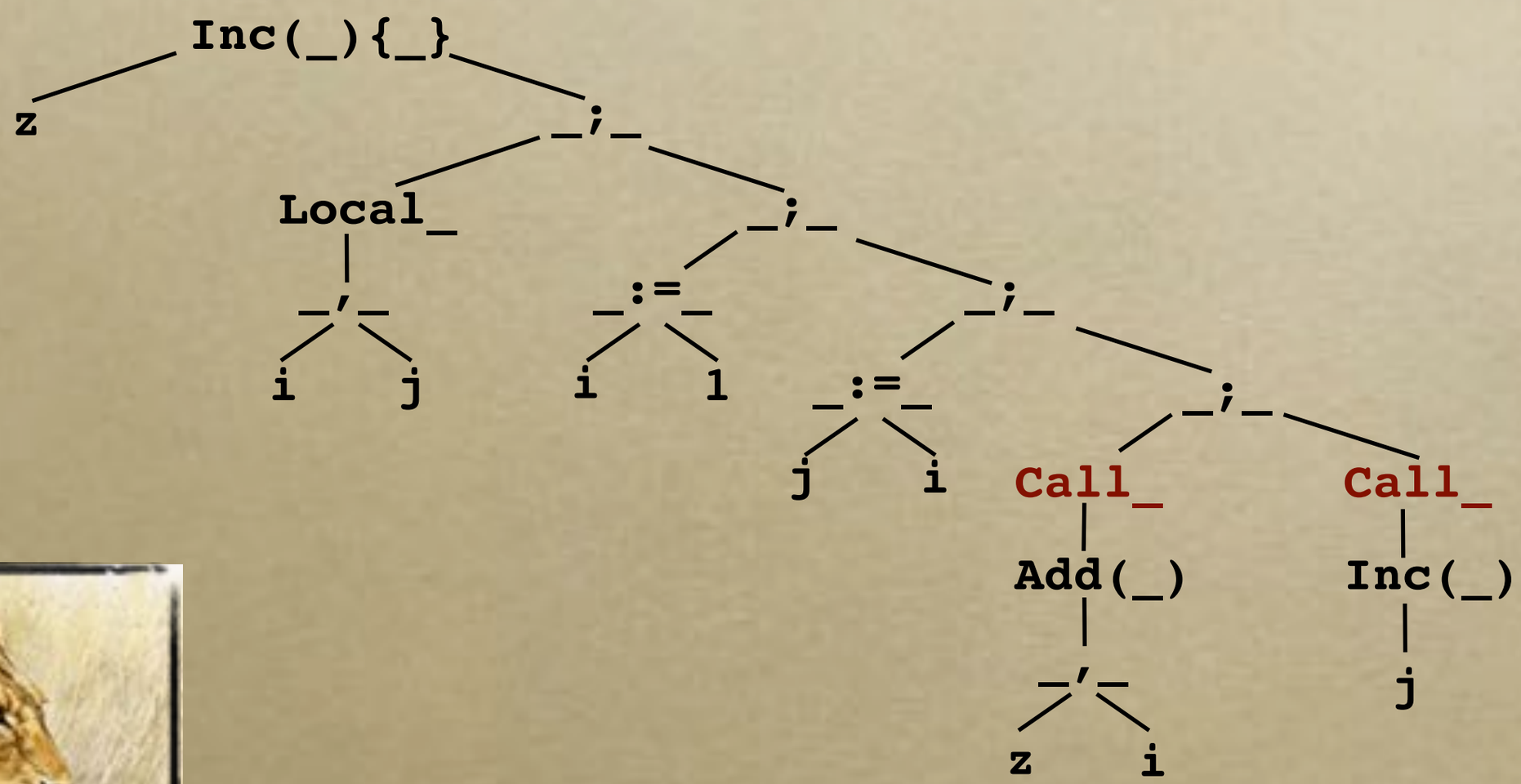
# Slicing-Features Synthesis of Sem.

- *Syn(context-updates)* - extracts the <u>*context-update*</u> constructs, i.e., the program constructs (terms of sort $\mathcal{P}$) that are *directly* responsible for <u>*stacking-up subparts of memory*</u> (stack-ops of $\mathcal{M}$):
  - ~~`Call _`~~
  - ~~`Throw _`~~
  - `...`
  - ~~`j _, jr _, jal _, jalr _`~~

To be described in LOPSTR'17.

# So... who is Syn(context-updates)?

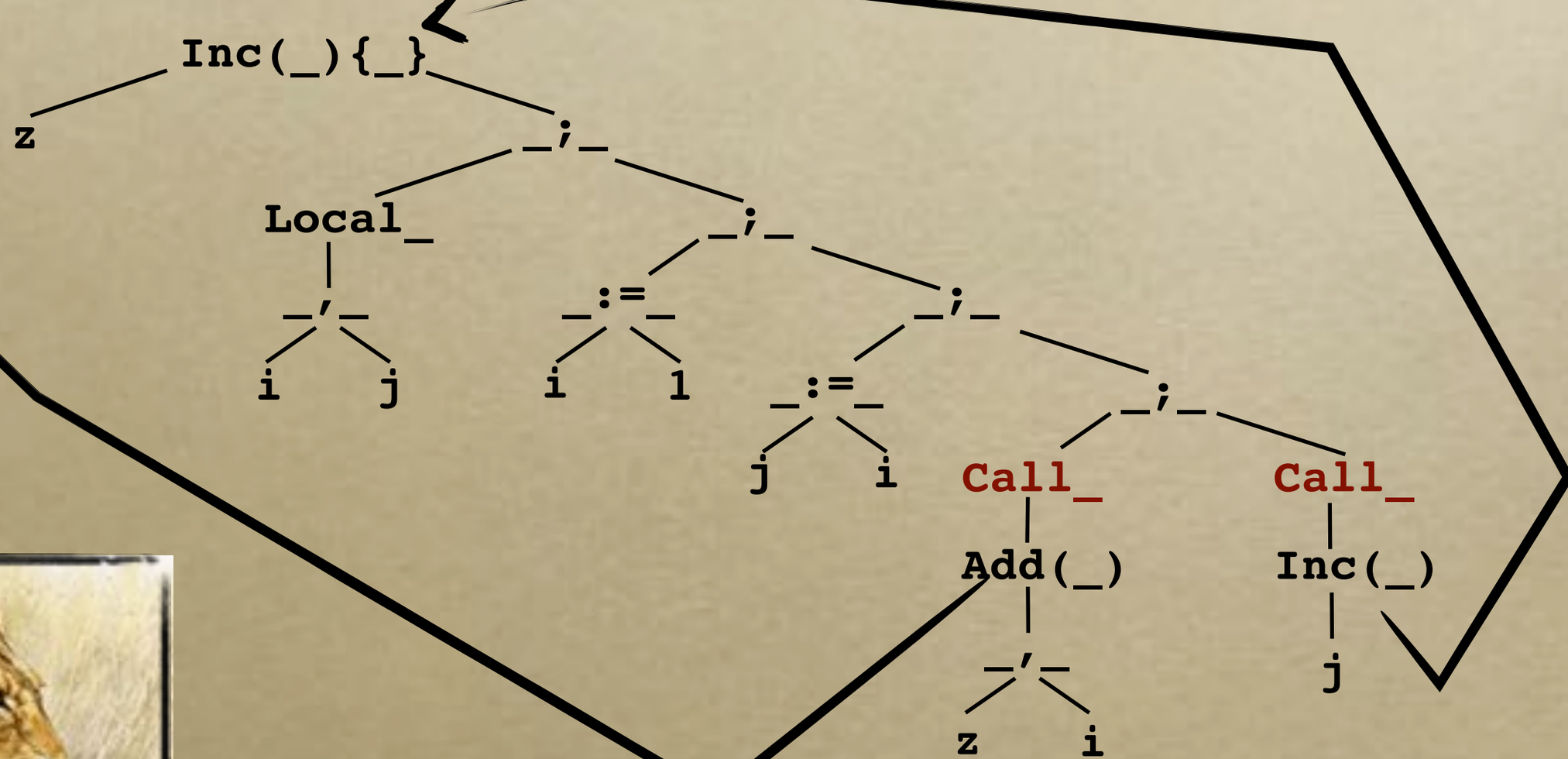
* Syn(context-updates) : the synthesis discovering context-updates




# So... who is Syn(context-updates)?

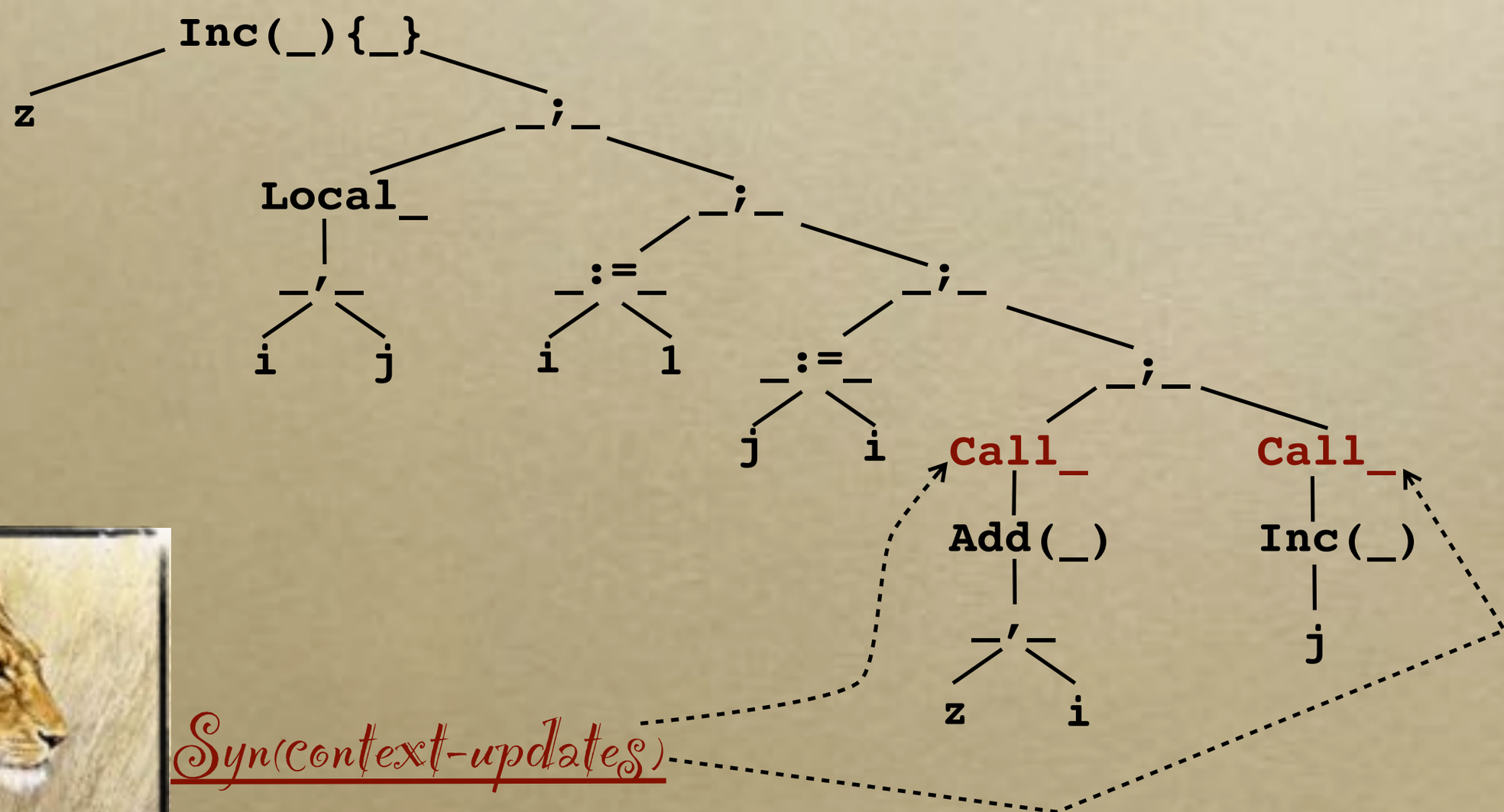
```
>Add(_){_}
```

* Syn(context-updates) : the synthesis discovering context-updates




# So... who is Syn(context-updates)?

* Syn(context-updates) : the synthesis discovering context-updates

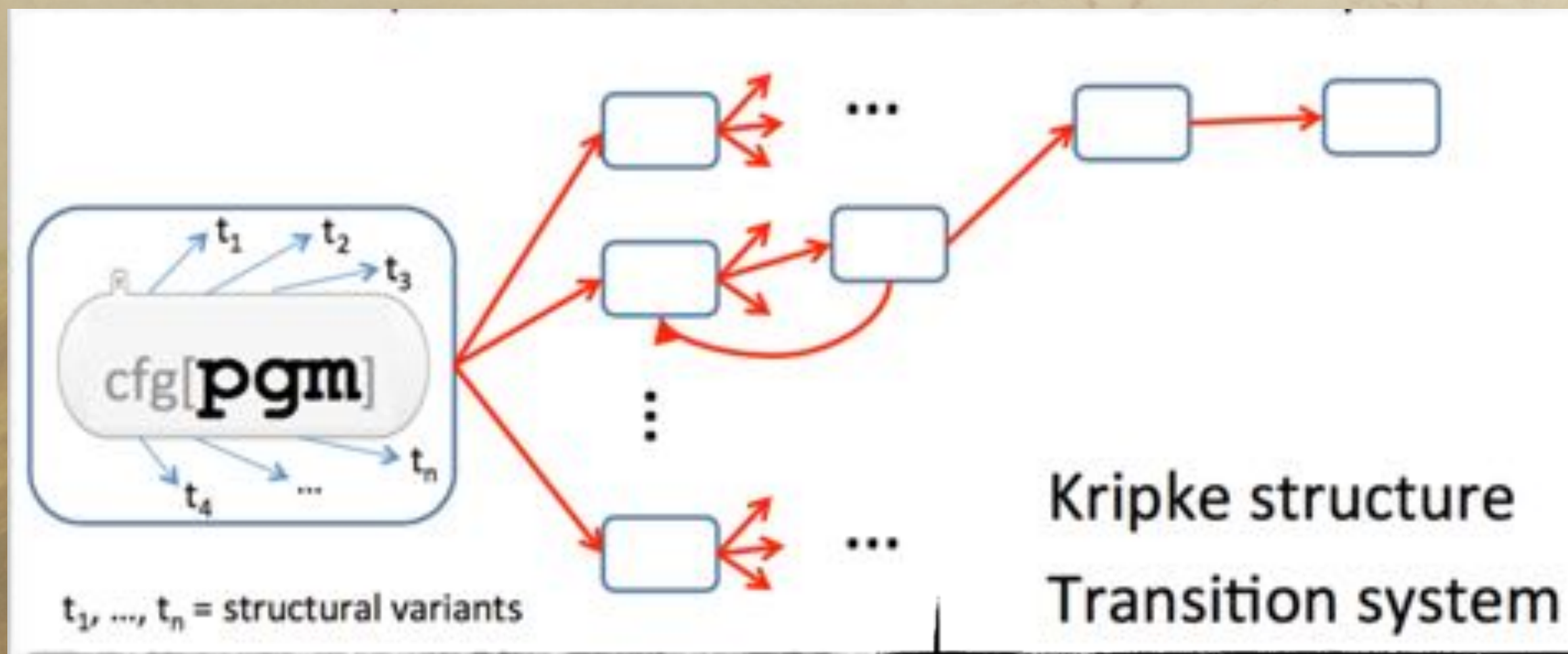



# Memento of Sem

```
crl [WhileR1] :
   < While be Do C, st, rwb, fs > => < skip, st, rwb, fs >
if < be, st > => < F, st > .

crl [WhileR2] :
   < While be Do C, st, rwb, fs > => < skip, st', rwb', fs >
if < be, st > => < T, st > /\
   < C ; (While be Do C), st, rwb, fs > => < skip, st', rwb', fs > .

crl [CallF] :
  < Call fn(actPrms), st, rwb, fs > => < skip, st'', rwb', fs >
if fn(Prms){ C } fs' := fs /\
   st' := asgP(actPrms, Prms, st | mt) /\
   < C, st', rwb, fs > => < skip, st'' | lenv', rwb', fs >  .

crl [LocV] :
  < Local vl, st, rwb, fs > => < skip, st', rwb, fs >
if st' := allocateLocals(vl, st) .
```

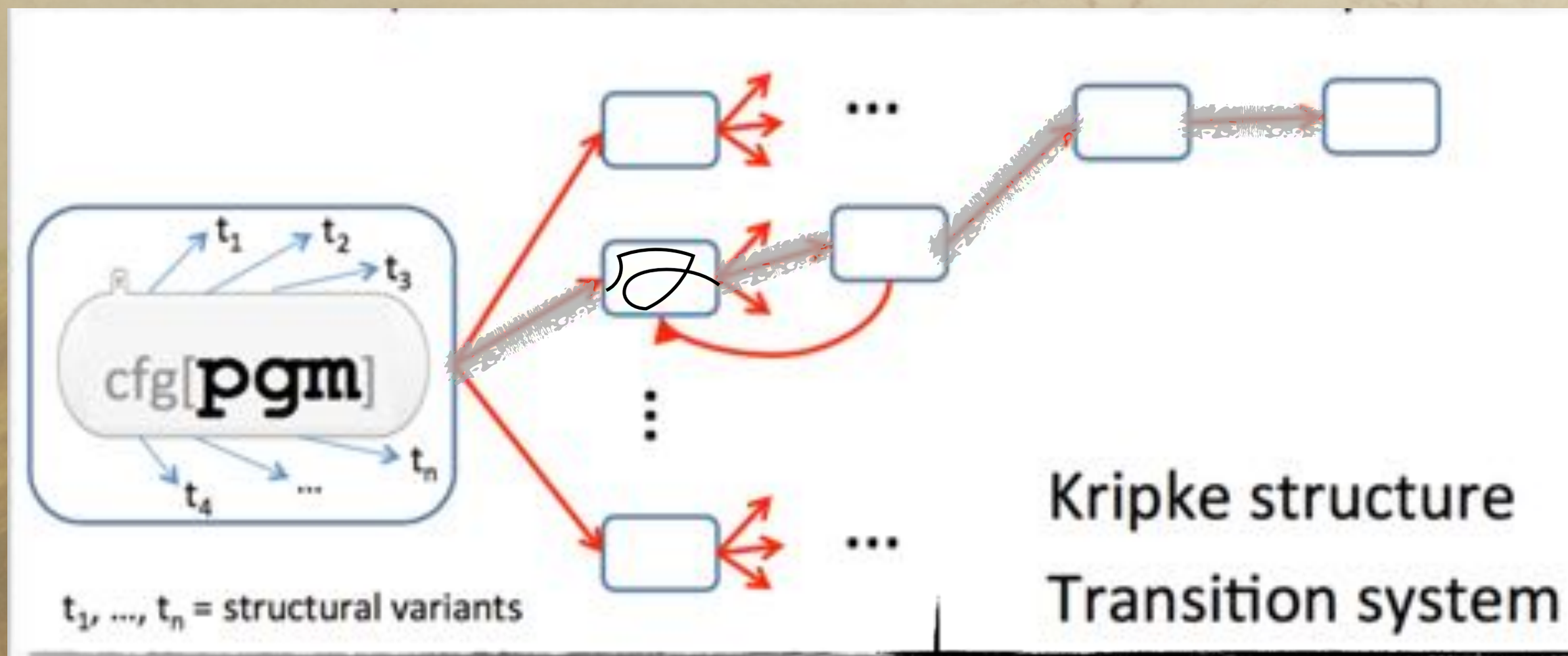
# Memento of Sem's pgm executions

- Sem - symbolic executions of a program pgm:

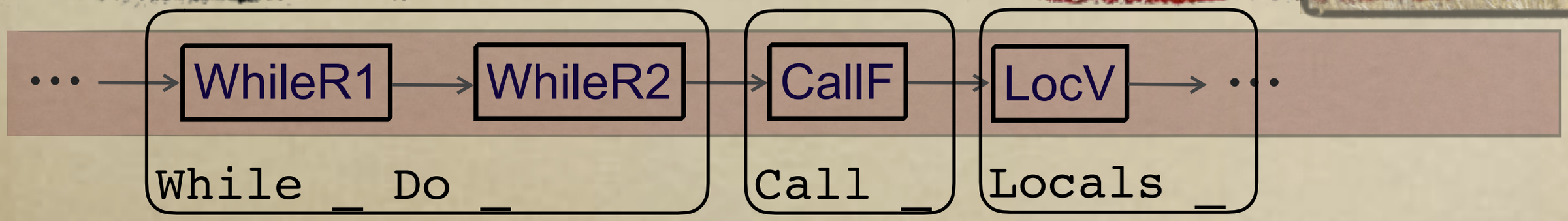


# Memento of Sem's pgm executions

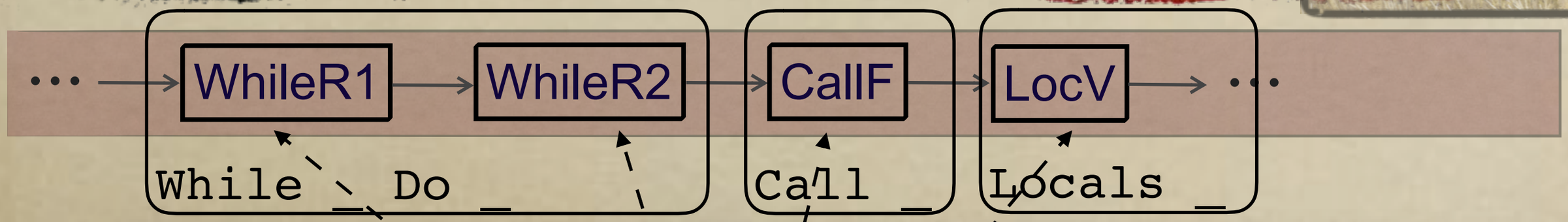
- Sem - symbolic executions of a program pgm:



# Unification HyperTree of Syn(context-updates)



# Unification HyperTree of Syn(context-updates)




*The rules*
*(identified by their unique labels)*
*that unify with the configuration terms*

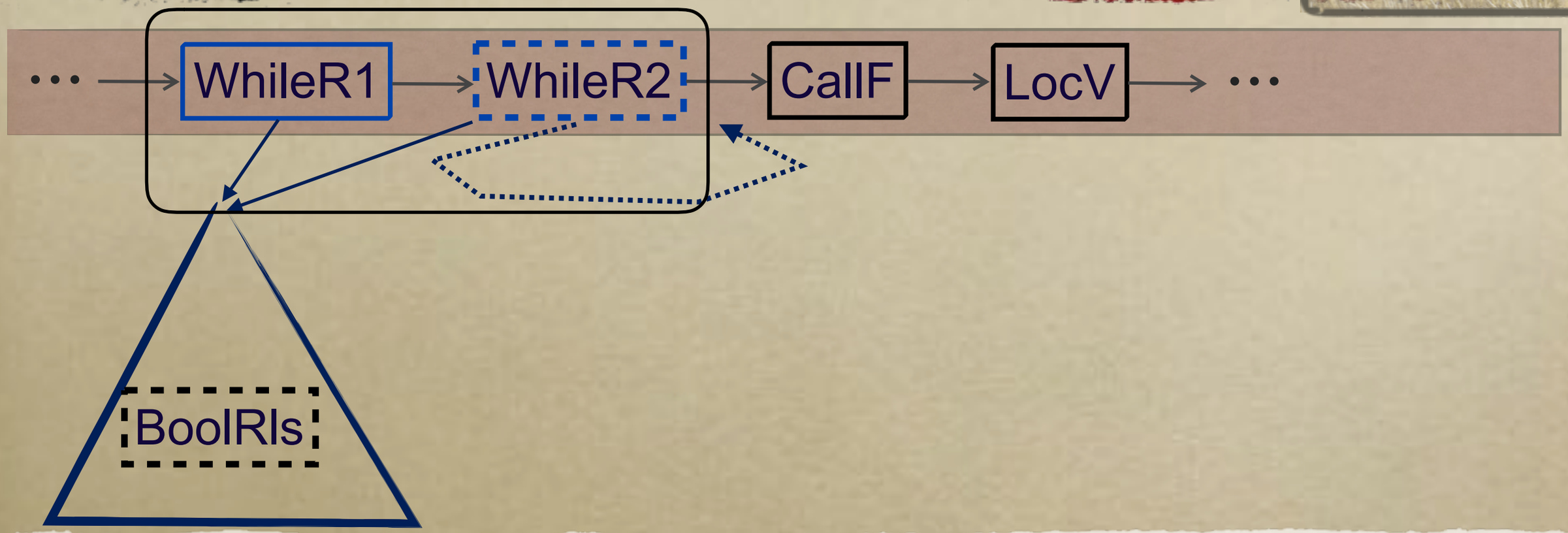
$$\langle pgm{:}\mathbb{P},\ mem{:}\mathbb{M}\rangle : \mathbb{C},$$

*i.e., the rules of form*

$$(c)rl\ [label]\ \langle pgm{:}\mathbb{P},\ mem{:}\mathbb{M}\rangle \Rightarrow \langle pgm'{:}\mathbb{P},\ mem'{:}\mathbb{M}\rangle$$

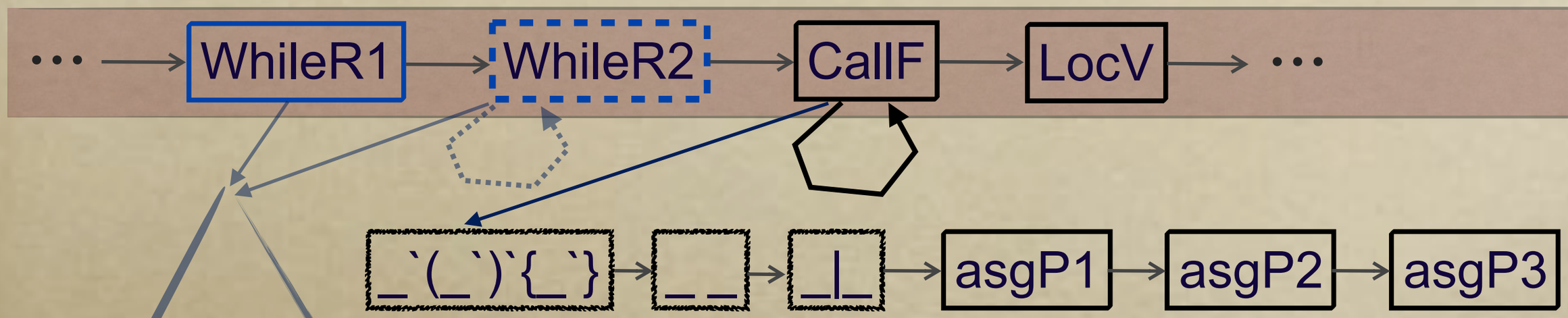
$$(\mathrm{if})\ \ldots$$

# Unification HyperTree of Syn(context-updates)



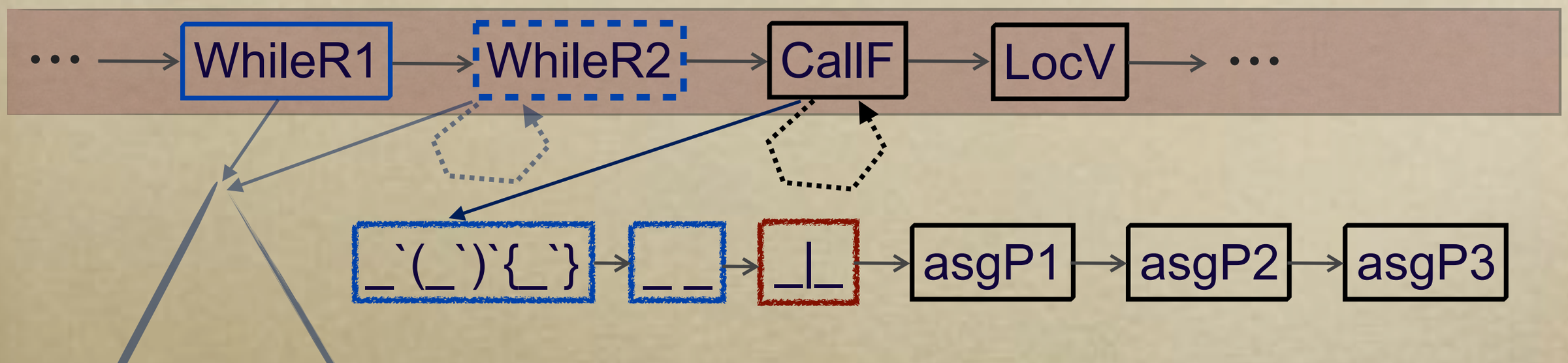

```
crl [WhileR1] :
   < While be Do C, st, rwb, fs > => < skip, st, rwb, fs >
if < be, st > => < F, st > .

crl [WhileR2] :
   < While be Do C, st, rwb, fs > => < skip, st', rwb', fs >
if < be, st > => < T, st > /\
   < C ; (While be Do C), st, rwb, fs > => < skip, st', rwb', fs > .
```

# Unification HyperTree of Syn(context-updates)




```
op asgP : ExpL VarL ESt -> ESt .
eq  [asgP1] : ...
ceq [asgP2] : ...
eq  [asgP3] : ...
op _`(_`)`{_`} : FunId VarL Com -> Fun [ctor] .
op nf : -> FunSet [ctor] .
op __ : FunSet FunSet -> FunSet [ctor comm assoc id: nf] .

op _|_ : ESt ESt -> ESt [ctor assoc] .

crl [CallF] :
  < Call fn(actPrms), st, rwb, fs > => < skip, st'', rwb', fs >
if fn(Prms){ C } fs' := fs /\ st' := asgP(actPrms, Prms, st | mt) /\
   < C, st', rwb, fs > => < skip, st'' | lenv', rwb', fs >  .
```

# Unification HyperTree of Syn(context-updates)

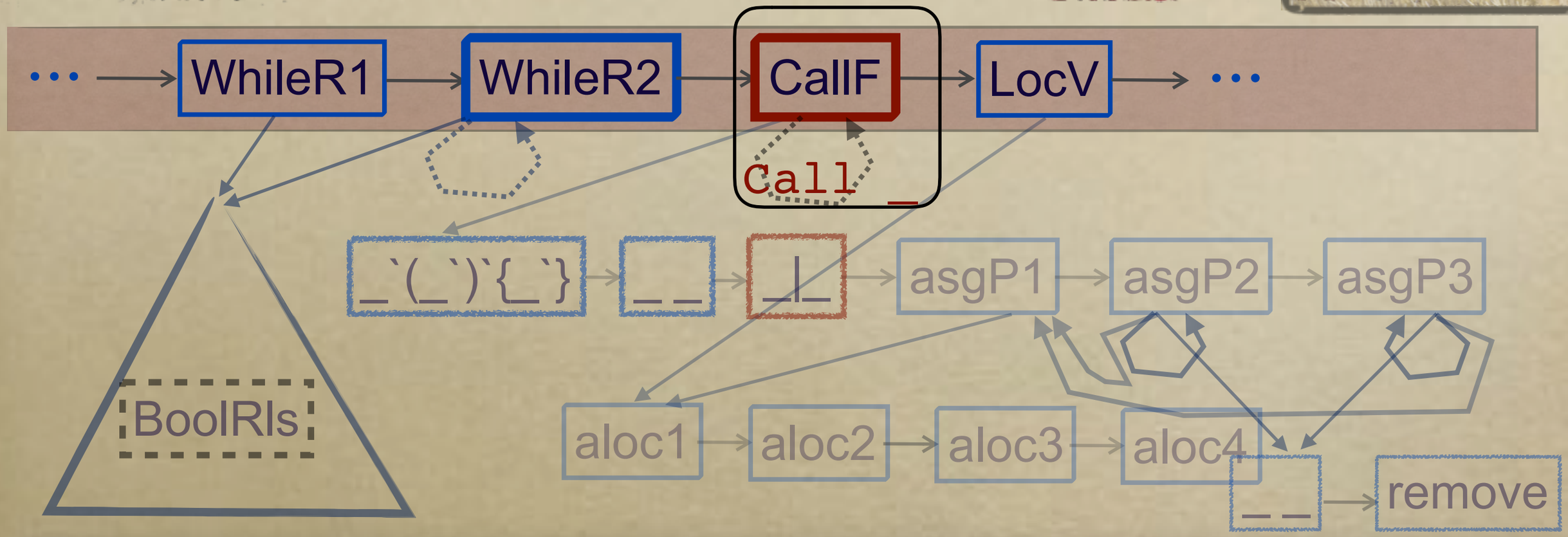



```
op asgP : ExpL VarL ESt -> ESt .
eq  [asgP1] : ...
ceq [asgP2] : ...
eq  [asgP3] : ...
op _`(_`)`{_`} : FunId VarL Com -> Fun [ctor] .
op nf : -> FunSet [ctor] .
op __ : FunSet FunSet -> FunSet [ctor comm assoc id: nf] .

op _|_ : ESt ESt -> ESt [ctor assoc] .
```

*memory stack operator*

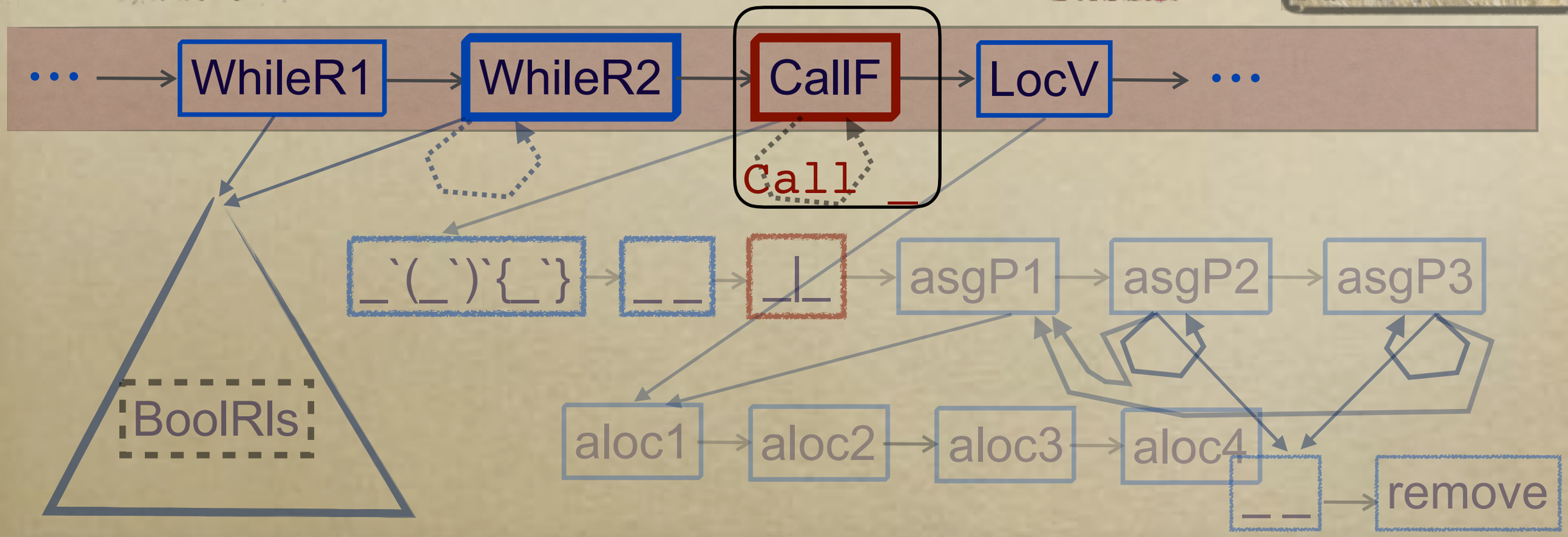
```
crl [CallF] :
  < Call fn(actPrms), st, rwb, fs > => < skip, st'', rwb', fs >
if fn(Prms){ C } fs' := fs /\ st' := asgP(actPrms, Prms, st | mt) /\
   < C, st', rwb, fs > => < skip, st'' | lenv', rwb', fs >  .
```

# Unification HyperTree of Syn(context-updates)




*Red label propagation from the memory stack operator* _|_ *to the* CallF *rule and further to the* `Call _` *language construct (the* $\mathcal{P}$ *subterm of the lefthand side of the rule* CallF*)*

# Unification HyperTree of Syn(context-updates)



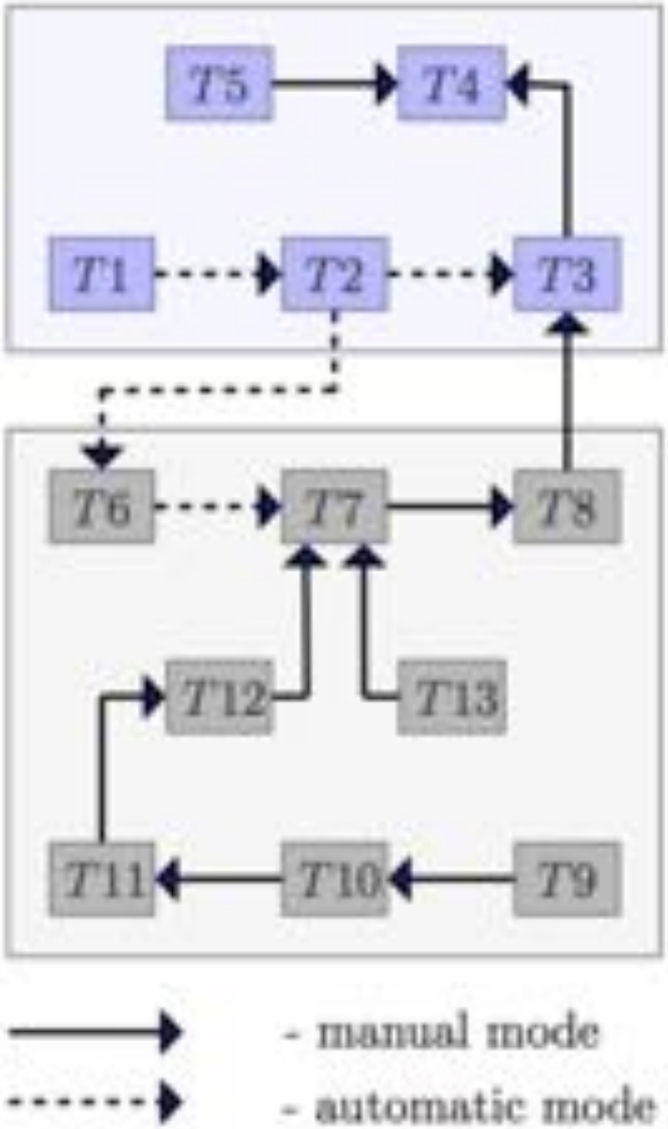

## Syn(context-updates)

*classifies all the red labeled language constructs, e.g.,* `Call _`*, as elements of* $\mathcal{O}$ *~ the over-approximated set of context-updates*

# Benchmarking Slicing and more...

* PapaBench tasks with their dependencies:

| Tasks : fbw and autopilot |
|---|
| T1 - receive_radio_commands |
| T2 - send_data_to_autopilot |
| T3 - receive_data_from_autopilot |
| T4 - transmit_servos |
| T5 - check_failsafe |
| T6 - manage_radio_commands |
| T7 - control_stabilization |
| T8 - send_data_to_fbw |
| T9 - receive_gps_data |
| T10 - control_navigation |
| T11 - control_altitude |
| T12 - control_climb |
| T13 - manage_reporting |




xt-updates

# Testing the Benchmarks for Synthesis Refinement

* Transforming the program term $pgm : \mathbb{P}$ into a list $Lp$:

```
Main () {
  sum := 0;
  Local i, j;
  i := 1; j := -1;
  While i < 11 Do
    Call A (sum, i);
    Call B (sum, j);
    Call A (j, i)
}
```

```
A (x, y) {
  If x > 1 Then
    Call Add(x, y);
    Call Inc (y)
}
B (x, y) {
  If x > 0 Then
    Call B(x + y, y)
}
```

```
Add (a, b) {
  a := a + b
}
Inc (z) {
  Local i, j;
  i := 1; j := i;
  Call Add (z, i);
  Call Inc (j)
}
```

Syn

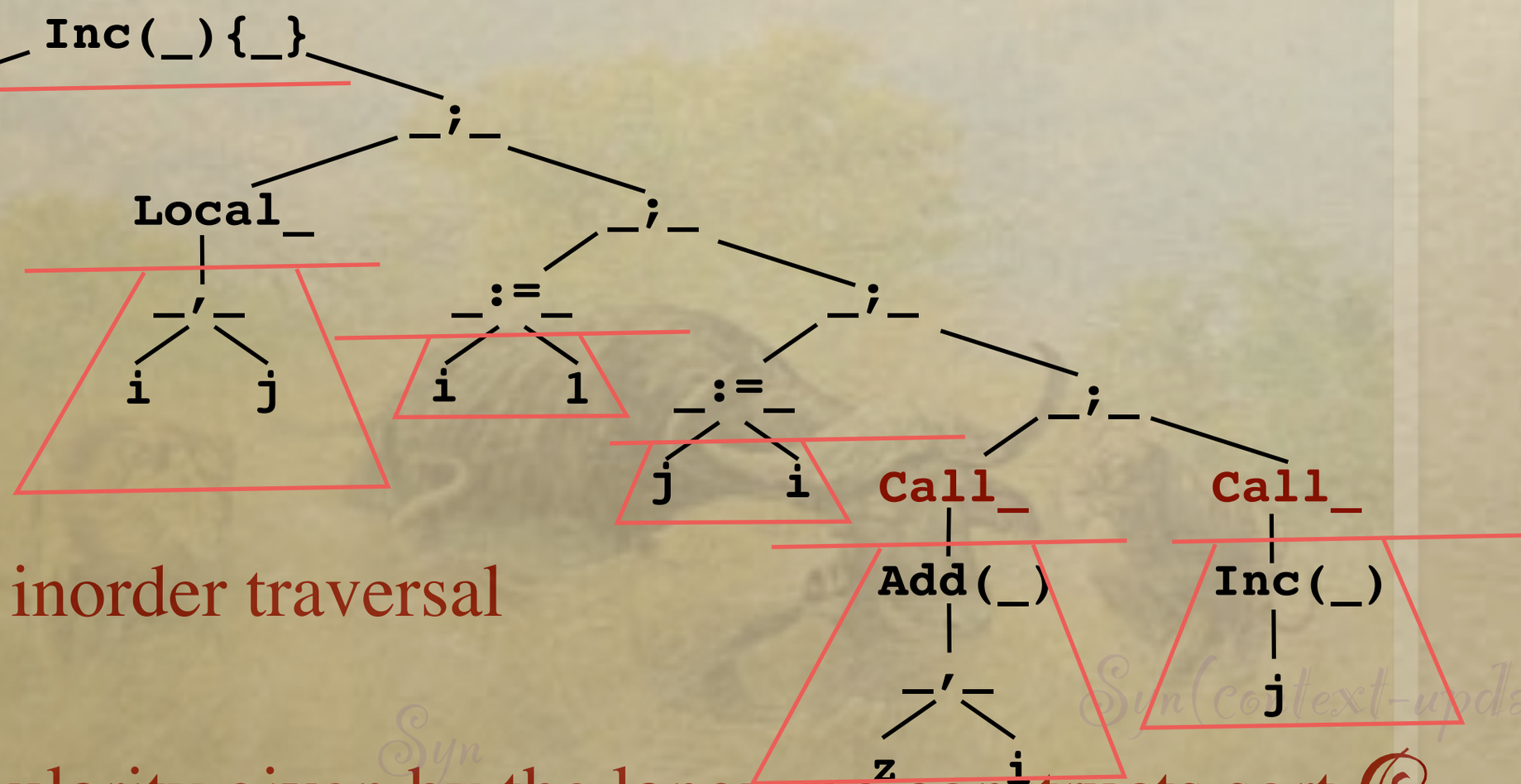
Syn(context-updates

# Testing the Benchmarks for Synthesis Refinement

* Transforming the program term $pgm$:$\mathbb{P}$ into a list $L_p$:

obtained via

GIT($pgm$)




i.e., granular inorder traversal

with the granularity given by the language constructs sort $\mathbb{C}$

# Testing the Benchmarks for Synthesis Refinement

* Fragmenting the list $L_p$ into function segments $[L_p]_{fn}$ :

```
Main () {
  sum := 0;
  Local i, j;
  i := 1; j := -1;
  While i < 11 Do
    Call A (sum, i);
    Call B (sum, j);
    Call A (j, i)
}
```

```
A (x, y) {
  If x > 1 Then
    Call Add(x, y);
    Call Inc (y)
}
B (x, y) {
  If x > 0 Then
    Call B(x + y, y)
}
```

```
Add (a, b) {
  a := a + b
}
Inc (z) {
  Local i, j;
  i := 1; j := i;
  Call Add (z, i);
  Call Inc (j)
}
```

Syn

Syn(context-updates

# Testing the Benchmarks for Synthesis Refinement

* Using Adrian Riesco's Maude Testing Tool:

```
Main () {
```

```
A (x, y) {
```

```
Add (a, b) {
```

**Definition 1.** *The property $\varphi$ w.r.t. $E$ is defined as follows:*

$$\forall \varsigma \in \mathcal{O},\ \forall \varpi \in E, \pi := \mathit{filter}_{\mathcal{C}}(\varpi),\ \forall i \in 1..|\pi| : \pi_i = \varsigma \implies$$
$$(\pi_{i-1}\pi_i \in L_p \implies \varsigma \in \mathcal{O}_r) \wedge$$
$$(\pi_{i-1}\pi_i \notin L_p \wedge (\pi_{i-1}, \pi_i) \in [L_p]_{fn}) \implies \varsigma \in \mathcal{O}_g) \wedge$$
$$(\pi_{i-1}\pi_i \notin L_p \wedge (\pi_{i-1}, \pi_i) \notin [L_p]_{fn}) \implies \varsigma \in \mathcal{O}_f)$$

# Benchmarking Evaluation on Refined Slicing

* PapaBench slicing results:

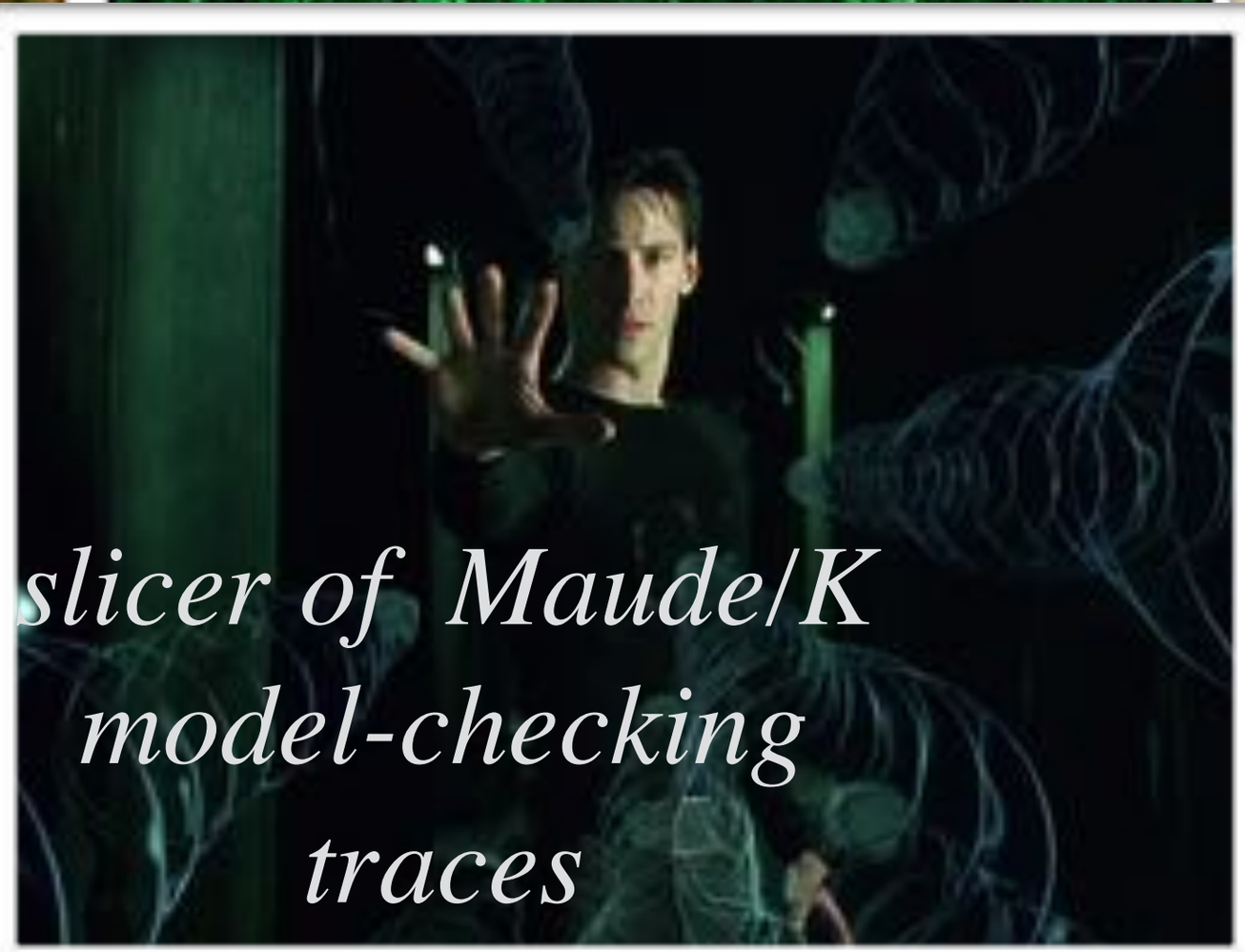
| Name | # Funs | # Calls | LOC (WhileFun) | red (%) (WhileFun) | LOC (MIPS) | red (%) (MIPS) |
|---|---|---|---|---|---|---|
| scheduler_fbw | 14 | 18 | 103 | 72.8 % | 396 | 44.4 % |
| periodic_auto | 21 | 80 | 225 | 73.3 % | 779 | 36.3 % |
| **fly_by_wire** | **41** | **110** | **638** | **91.1 %** | **1913** | **41 %** |
| T1 | 10 | 26 | 119 | 76.5 % | 534 | 36.2 % |
| T2 | 9 | 9 | 59 | 69.5 % | 329 | 44.4 % |
| T3 | 9 | 24 | 82 | 76.5 % | 501 | 43.6 % |
| T4 | 9 | 14 | 50 | 61.5 % | 235 | 34.5 % |
| T5 | 7 | 22 | 66 | 67 % | 453 | 51 % |
| **autopilot** | **95** | **214** | **1384** | **92 %** | **5639** | **41.5 %** |
| T6 | 36 | 71 | 306 | 77.2 % | 1329 | 54 % |
| T7 | 9 | 13 | 57 | 70 % | 426 | 42 % |
| T8 | 7 | 15 | 54 | 69.2 % | 219 | 38 % |
| T9 | 15 | 30 | 87 | 75 % | 617 | 36.5 % |
| T10 | 18 | 27 | 102 | 71.1 % | 1002 | 42.2 % |
| T11 | 3 | 2 | 15 | 63.4 % | 90 | 70.6 % |
| T12 | 4 | 3 | 49 | 66.2 % | 363 | 50 % |
| T13 | 37 | 93 | 240 | 79.7 % | 1535 | 42 % |

xt-updates

# Contributions, Future and Related Work


# Contributions, Future and Related Work



# Thanks! & Question?