

Make FunBlocks alive

Marvin FOURASTIE

Master project

Motivations



Educational purpose



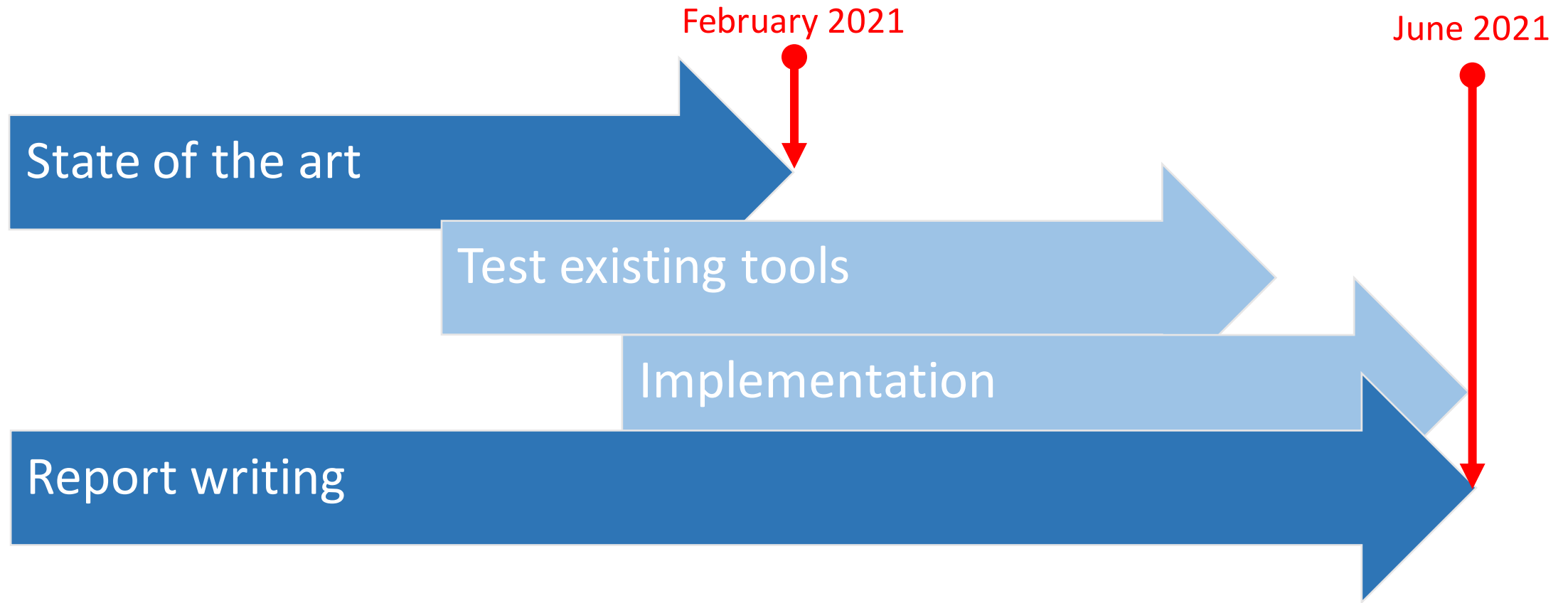
Imperative paradigm alternative



Based on rewrite systems

FunBlocks

Road Map



FunBlocks

```
1  init area(disk(r))  
2  
3  case area(disk($d)) => mul(pi,square($d))
```

PROGRAM STATE

area disk r

RULES

▶ area disk < d > → mul pi square < d >

PROGRAM STATE

mul pi square r

RULES

▶ area disk < d > → mul pi square < d >

FunBlocks

Declarative paradigm



```
area(disk(r))
```

Based on rewrite systems



```
case area(disk($d)) => mul(pi,square($d))
```

Static typing



```
type Tree $t :: empty | leaf $t | node (Tree $t) (Tree $t)
```

Goals

Provide users with **valuable insights** about their program

→ Verification of rewrite systems

Rewrite systems

Stack operators

$\text{Zero} = \{0\}$

$\text{Nat} = \text{Zero} \cup \text{succ}(\text{Nat})$

$\text{Empty} = \Lambda$

$\text{Stack} = \text{Empty} \cup \text{push}(\text{Nat}, \text{Stack})$

$\text{top} : \text{Stack} \rightarrow \text{Nat}$

$\text{pop} : \text{Stack} \rightarrow \text{Stack}$

$\text{alternate} : \text{Stack} \times \text{Stack} \rightarrow \text{Stack}$

Rewrite systems

Canonical rewrite system

$\text{top}(\text{push}(x, y)) = x$

$\text{pop}(\text{push}(x, y)) = y$

$\text{alternate}(\Lambda, z) = z$

$\text{alternate}(\text{push}(x, y), z) = \text{push}(x, \text{alternate}(z, y))$



$\text{top}(\text{push}(x, y)) \rightarrow x$

$\text{pop}(\text{push}(x, y)) \rightarrow y$

$\text{alternate}(\Lambda, z) \rightarrow z$

$\text{alternate}(\text{push}(x, y), z) \rightarrow \text{push}(x, \text{alternate}(z, y))$

Rewrite systems

Termination

Confluence

Soundness

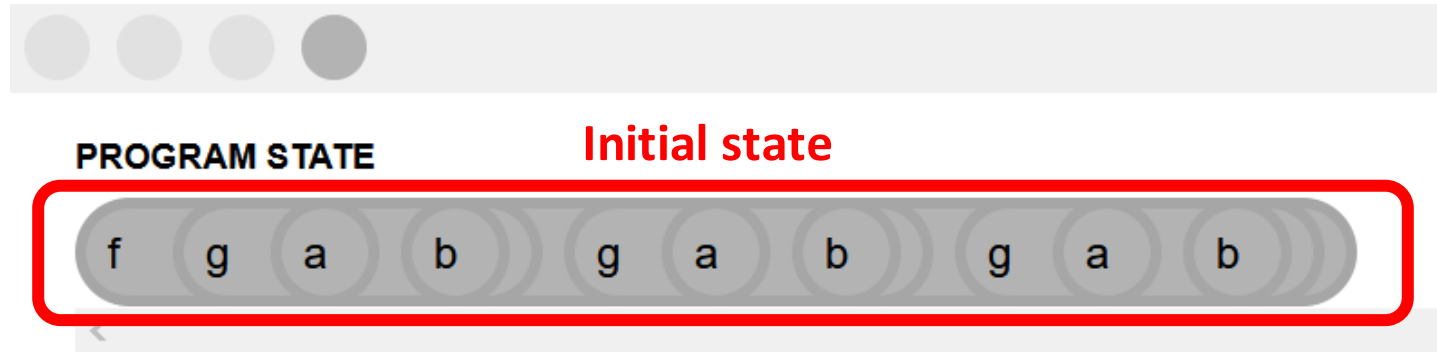
Completeness

Correctness

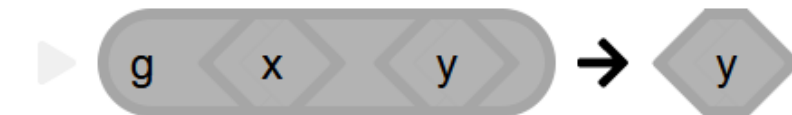
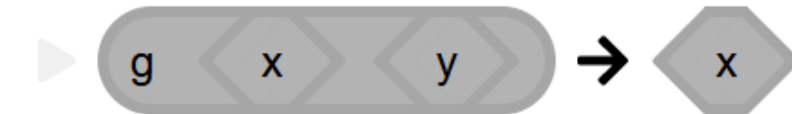
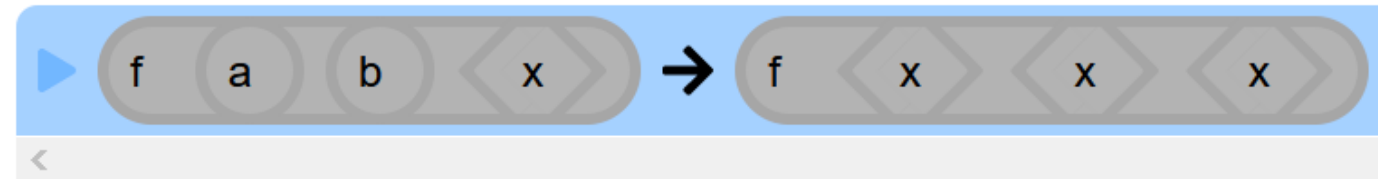


Undecidable in general

Termination



RULES



Reduction order

Monotone



$s_i > t \rightarrow f(s_1, \dots, s_i, \dots, s_n) > f(s_1, \dots, t, \dots, s_n)$
For f of arity n

Close under substitution



$s > t \rightarrow \sigma s > \sigma t$, for all substitution σ

Well-founded



no infinite descending chain

$(\mathbb{N}, <)$ is well-founded

$(\mathbb{Z}, <)$ is not well-founded

Termination

A term rewriting system is **terminating**

if and only if

it admits a **compatible reduction order** $<$
(if $l > r$ for every rewrite rule $l \rightarrow r$)



Verification of termination

Polynomial interpretation

$$f(a, x) \rightarrow x$$

$$f(g(x), y) \rightarrow g(f(x, y))$$

weight
→

$$w(a) = 1$$

$$w(g(t)) = 1 + w(t)$$

$$w(f(t_1, t_2)) = 2w(t_1) + w(t_2)$$

Polynomial interpretation

$$f(a, x) \rightarrow x$$

$$w(f(a, x)) = 2 + w(x)$$

$$w(x) = w(x)$$

$$w(f(a, x)) > w(x)$$

$$f(g(x), y) \rightarrow g(f(x, y))$$

$$w(f(g(x), y)) = 2 + 2w(x) + w(y)$$

$$w(g(f(x, y))) = 1 + 2w(x) + w(y)$$

$$w(f(g(x), y)) > w(g(f(x, y)))$$

Reduction order \rightarrow Termination

Algorithms

Recursive Path Ordering



Order based on the mutisets

Knuth-Bendix Ordering



Based on weights assigned to operators

Dependency pairs



Prove innermost termination

Termination

APROVE

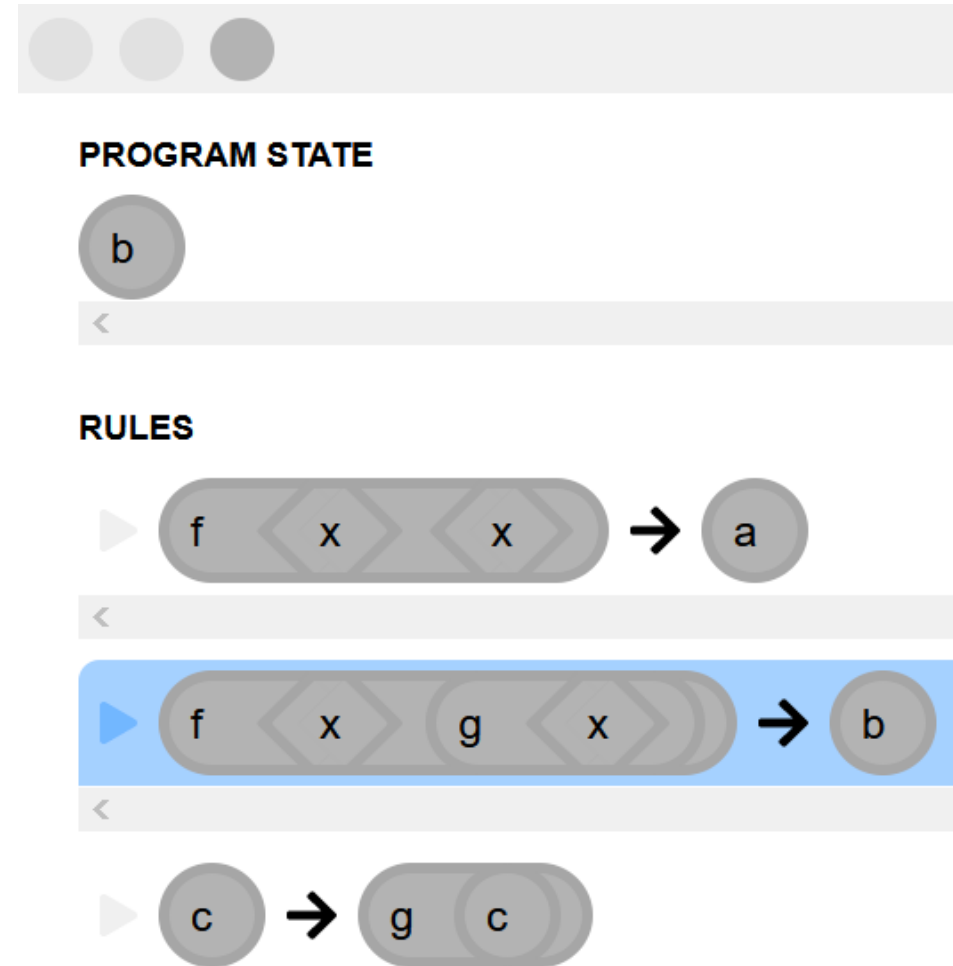
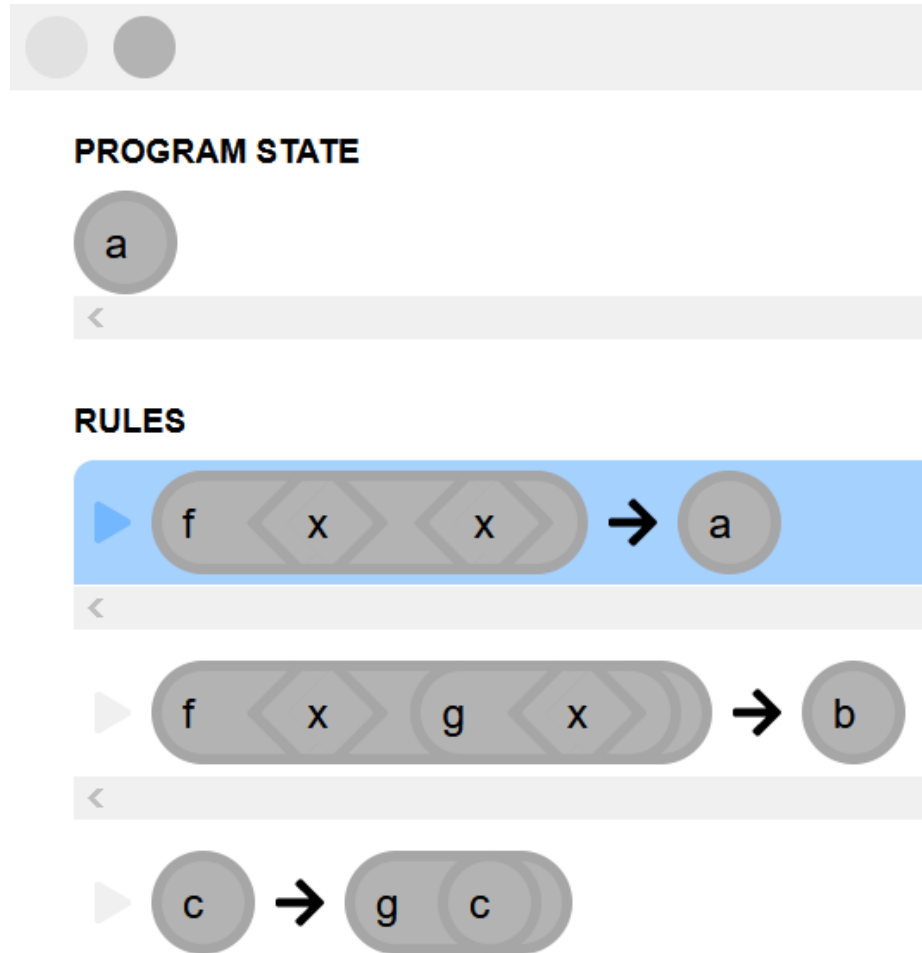
=

Direct proof
(polynomial, LBO, KBO,...)

+

Dependency pairs and
size-change principle

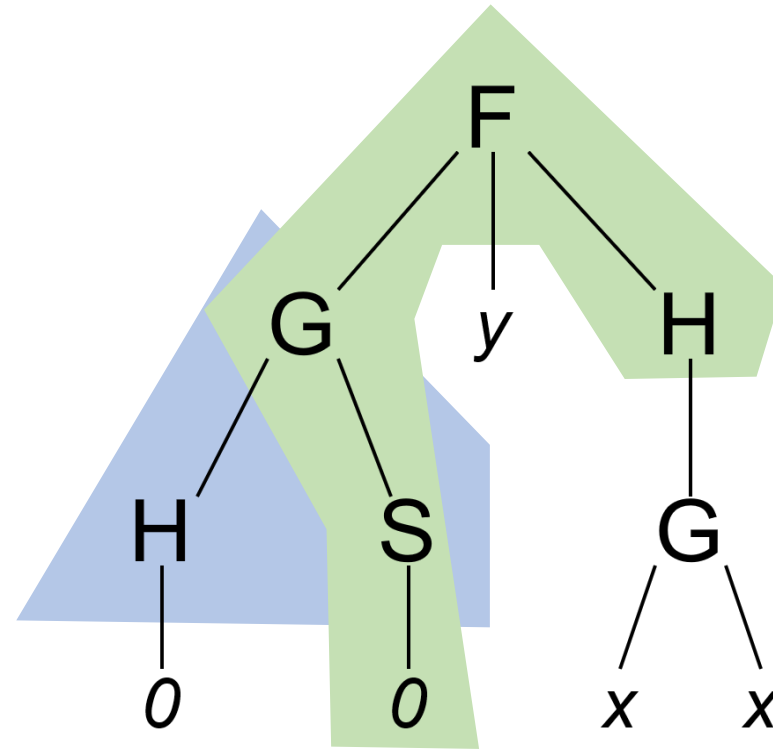
Confluence



Overlap and critical pairs

$$\rho_1 : F(G(x, S(0)), y, H(z)) \rightarrow x$$

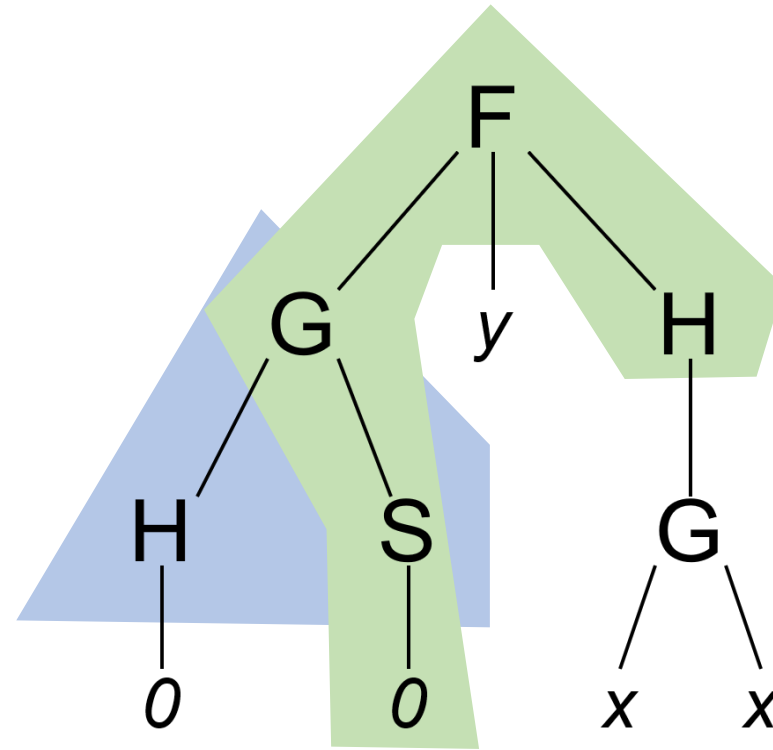
$$\rho_2 : G(H(x), S(y)) \rightarrow y$$



Overlap and critical pairs

Overlapping:

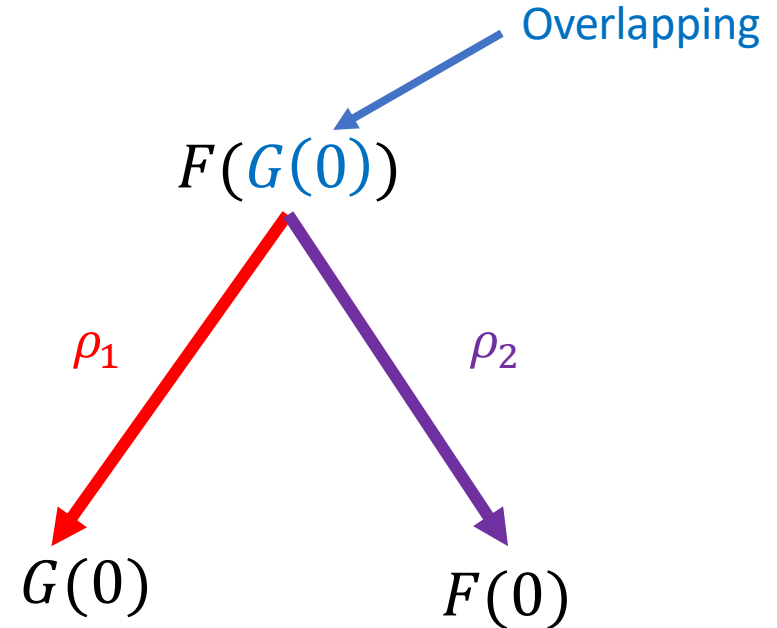
Term: $F\left(G\left(H(0), S(0)\right), y, H(z)\right)$

$$F\left(G(\square, S(0)), \square, H(\square)\right)$$
$$G(H(\square), S(\square))$$


Overlap and critical pairs

$$\rho_1 : F(x) \rightarrow G(0)$$

$$\rho_2 : G(x) \rightarrow 0$$



$\langle G(x), F(x) \rangle$ is called **critical pair**

Critical Pair Lemma

A terminating rewriting system is confluent

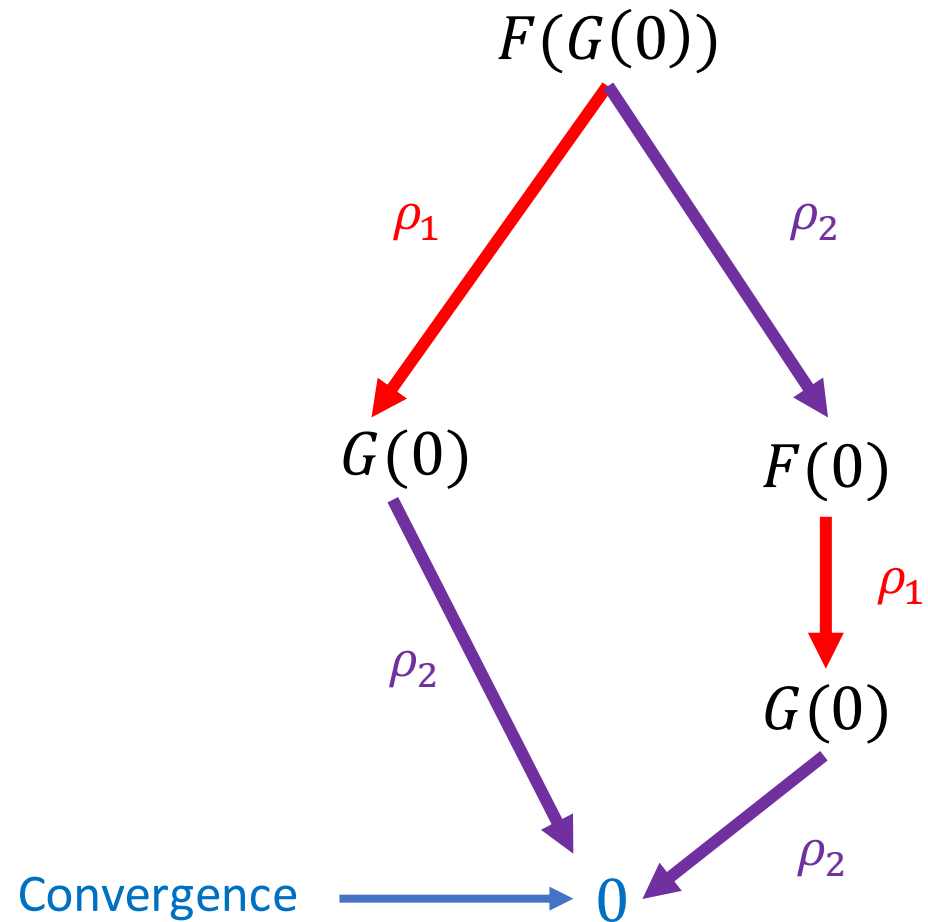
if and only if

all critical pairs are convergent

Critical Pair Lemma

$$\rho_1 : F(x) \rightarrow G(0)$$

$$\rho_2 : G(x) \rightarrow 0$$



Knuth-Bendix completion

Input:

A set of equation

A reduction ordering $<$

$$1 \cdot x = x$$

$$x^{-1} \cdot x = 1$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

Non-confluent

Knuth-Bendix completion

Output:

Terminate successfully



Terminating and confluent rewrite system

Loop indefinitely



Non-terminating rewrite system

Fail



Rule which cannot be ordered
(i.e. commutative operator)

Knuth-Bendix completion

Basic rules:

Orienting



Transform $s = t$ to $s \rightarrow t$

Adding



Add $s = t$ in the set of equation

Simplifying



Simplify $s = t$ in $s' = t'$

Deleting



Delete trivial rules $s = s$

Knuth-Bendix completion

Adding



Add $s = t$ in the set of equation

$$\begin{array}{lcl} (x * y) * z & \rightarrow & x * (y * z) \\ x * x & \rightarrow & x \end{array}$$



$$(x * x) * z$$

$x * z$ $x * (x * z)$

No convergent

$$\begin{array}{lcl} (x * y) * z & \rightarrow & x * (y * z) \\ x * x & \rightarrow & x \\ x * (x * z) & \rightarrow & x * z \end{array}$$

New rule added

Knuth-Bendix completion

Completion process:

1. For each equation $s = t$ reduce s and t to normal form s' and t'
2. Fill the set of rules using basic operators and reduction ordering
3. If the algorithm terminate successfully: terminating and confluent rewrite system

Knuth-Bendix completion

Completion for axioms of groups:

$$\begin{array}{ll} 1 \cdot x & = x \\ x^{-1} \cdot x & = 1 \\ (x \cdot y) \cdot z & = x \cdot (y \cdot z) \end{array}$$



$$\begin{array}{ll} 1 \cdot x & \rightarrow x \\ x^{-1} \cdot x & \rightarrow 1 \\ (x \cdot y) \cdot z & \rightarrow x \cdot (y \cdot z) \\ x^{-1} \cdot (x \cdot y) & \rightarrow y \\ 1^{-1} & \rightarrow 1 \\ x \cdot 1 & \rightarrow x \\ (x^{-1})^{-1} & \rightarrow x \\ x \cdot x^{-1} & \rightarrow 1 \\ x \cdot (x^{-1} \cdot y) & \rightarrow y \\ (x \cdot y)^{-1} & \rightarrow y^{-1} x^{-1} \end{array}$$

Educational tools

$T_T T_2 / \text{RS}^{\text{TOOL}}$

Termination

CSI

Confluence

KBCV

Completion

TRS tool

Parcourir... Aucun fichier sélectionné.

Upload

```
(VAR x y)
(RULES
  f(x,y) -> x
  f(x,y) -> f(x,g(y))
  g(x) -> h(x)
  F(g(x),x) -> F(x,g(x))
  F(h(x),x) -> F(x,h(x))
)
(COMMENT Example 6 of \cite{AT97})
(COMMENT %% TagRevision: 1 %%)
(COMMENT %% Tags: [4ec3f85c01836]non_left_linear{};[4ec3f87f0f1e0]r
```

Go!

50



Rewrites Limit (Use with caution)

TRS tool

$R_0 = f(x, y) \rightarrow x$
R_0 is Left-Linear
R_0 is Right-Linear
R_0 is Linear
R_0 is Collapsing
R_0 is not Duplicating
R_0 is not Conservative
R_0 is Destructive

TRS
The TRS is not Left-Linear
The TRS is not Right-Linear
The TRS is not Linear
The TRS is Collapsing
The TRS is not Duplicating
The TRS is not Conservative
The TRS is Destructive
The TRS is not Orthogonal
The TRS is not Almost Orthogonal
The TRS is not Weakly Orthogonal
The TRS is Locally Confluent
Unknown confluence for The TRS
The TRS is non terminating Infinite Loop: $f(x, g(y)) \rightarrow \underline{f(x, g(g(y)))}$

TTT2

Tyrolean Termination Tool 2 (1.20)

1. Input Term Rewrite System

For input use the **standard TRS format**.

select example ... or upload file Parcourir... Aucun fichier sélectionné.

```
(VAR x y)
(RULES
  add(0,y) -> y
  add(s(x),y) -> s(add(x,y))
  mul(0,y) -> 0
  mul(s(x),y) -> add(y,mul(x,y))
)
```

2. Select Strategy

☒ FAST ☐ FBI ☐ HYDRA ☐ LPO ☐ KBO ☐ POLY ☐ MAT(2) ☐ MAT(3) ☐ COMP ☐ COMPLEXITY
☐ EXPERT

3. Encode State into URL (optional)

encode URL clear URL

4. Start TTT2

check ☐ use HTML output if available (*experimental feature*)

CSI

Enter a **TRS** or **HRS** or upload a file

[browse...](#)

Examples:

[trs1](#)

[trs2](#)

[hrs1](#)

[hrs2](#)



[CSI 0.1](#)

[CSI 0.6](#)

[CSI 1.1](#)

[CSI 1.1✓](#)

[UNR](#)

[UNC](#)

[CR](#)

[CSI 1.2.4](#)

[CSI 1.2.4✓](#)

[CSI^ho](#)

[encode URL](#)

[clear URL](#)

[reset](#)

[submit](#)

KBCV

Orient →

Orient ←

Simplify

Delete

Compose

Collapse

Deduce

Completion

Undo

Redo

Equations

from 1 to 500

1. $f(f(x, y), z) \approx f(x, f(y, z))$

2. $f(x, c) \approx x$

3. $f(x, g(x)) \approx c$

Rules

from 1 to 500

LPO Precedence

Undo / Redo Stack

1. start : Welcome to the 'Knuth-Bendix Completion Visualizer'!

2. add : equations $f(f(x,y),z)=f(x,f(y,z))$, $f(x,c)=x$, $f(x,g(x))=c$ were added

Status Messages

Welcome to the 'Knuth-Bendix Completion Visualizer'!

equations $f(f(x,y),z)=f(x,f(y,z))$, $f(x,c)=x$, $f(x,g(x))=c$ were added

file '.kbcvinit' loaded!

Performance tools

Termination

MU-TERM

NaTT

APROVE

Confluence

ACP

Saigawa

Completion

Maxcomp

Proof verification

CoLoR

CeTA

Hybrid tools

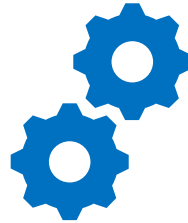


MoudE3

CiME



Rewriting toolkit

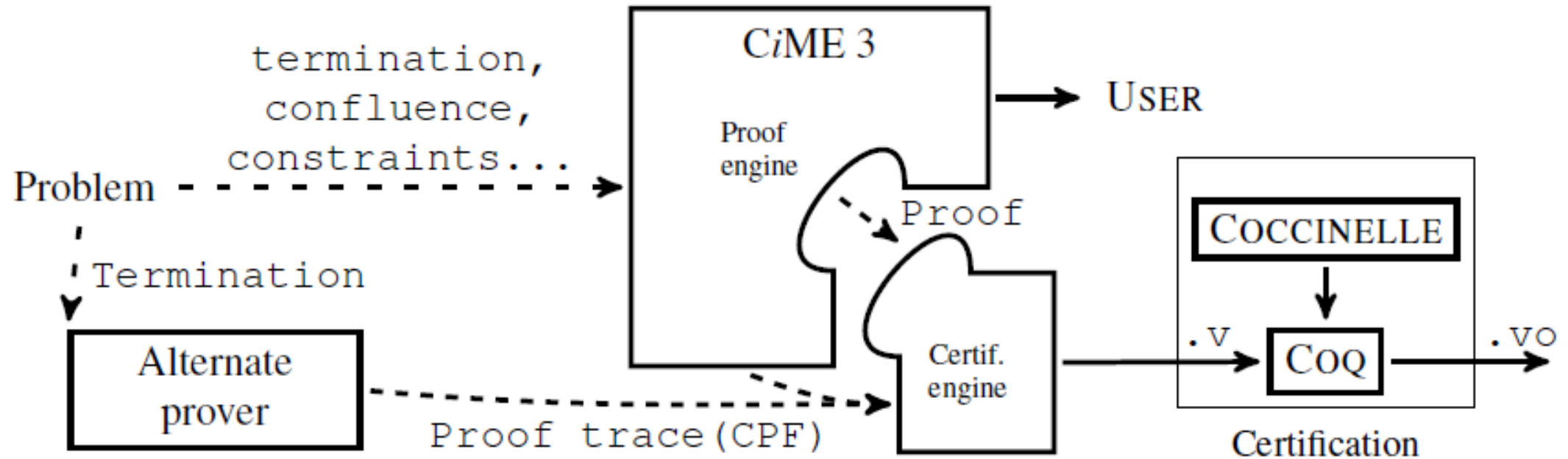


Proof engine



Proof certification

CiME



CiME

Examples of declarations

```
let X = variables "x,y";  
  
let F = signature "plus : binary; 0:constant; S:unary;";  
  
let T = algebra F;  
  
let t1 = term T "S(0)";  
  
let R = trs T "plus(0,x) -> x; plus(S x, y) -> S(plus(x,y));";  
  
let c = order_constraints T "0 < S(0) /\ S(plus(x,y)) < plus(S(x),y)";
```

CiME

Definition of signatures

```
CiME> let F_peano = signature "  
    0 : constant; s : unary; +,* : infix binary;  
    ";
```

```
F_peano : signature = signature "* : 2; s : 1; + : 2; 0 : 0"
```

```
CiME>let X = variables "x,y,z";
```

```
X : variable_set = variables "z,x,y"
```


CiME

Definition of algebra and terms

```
CiME> let A_peano = algebra F_peano ;  
A_peano : F_peano algebra = algebra F_peano
```

```
CiME> let t = term A_peano "s(s(s(0)))*(s(0)+s(s(0)))";  
t : F_peano term = s(s(s(0))) * (s(0)+s(s(0)))
```

CiME

Term rewriting system

```
CiME> let R_peano = trs A_peano "  
  x+0 -> x;  
  x+s(y) -> s(x+y);  
  x*0 -> 0;  
  x*s(y) -> (x*y)+x;  
  ";  
  
R_peano : F_peano trs = trs A_peano "  
  x+0 -> x;  
  x+s(y) -> s(x+y);  
  x *0 -> 0;  
  x *s(y) -> (x *y)+x "
```

```
CiME> termination R_peano;  
  
CiME> coq_certify_proof R_peano;  
  
CiME> convergence R_peano ;  
  
...
```

Maude

MoudE3



Simplicity



Expressiveness



Performance

Maude

```
1  fmod BASIC-NAT is
2      sort Nat .
3
4      op 0 : -> Nat .
5      op s : Nat -> Nat .
6      op _+_ : Nat Nat -> Nat .
7
8      vars N M : Nat .
9
10     eq 0 + N = N .
11     eq s(M) + N = s(M + N) .
12 endfm
```

Maude

```

5      fmod FACTORIAL is
6          protecting NAT .
7          op _! : Nat -> NzNat .
8          var N : Nat .
9          eq 0 ! = 1 .
10         eq (s N) ! = (s N) * N ! .
11     endfm

```

```
> load factorial.maunder
```

```
> red 100 ! .
```

Reduce in FACTORIAL : 100 ! .

```
rewrites: 201 in 0ms cpu (0ms real) (~ rewrites/second)
```

```
result NzNAT:
```

9332621544394415268169923885626670049071596826438162146
8592963895217599993229915608941463976156518286253697920
827223758251185210916864000000000000000000000000

Maude

```
7      mod VENDING-MACHINE is
8        including VENDING-MACHINE-SIGNATURE .
9        var M : Marking .
10       rl [add-q] : M => M q .
11       rl [add-$] : M => M $ .
12       rl [buy-c] : $ => c .
13       rl [buy-a] : $ => a q .
14       rl [change] : q q q q => $ .
15     endm
```

Maude

Inductive Theorem Prover (ITP)

Sufficient Completeness Checker (SCC)

Church-Rosser Checker (CRC)

Coherence Checker (ChC)

Maude Termination Tool (MTT)



Maude Formal Environment (MFE)

Tools overview

	Maude	CiME
Extensibility	+	≈
Still active	≈	—
I/O files	+	+
Syntax	+	+
Documentation	+	—

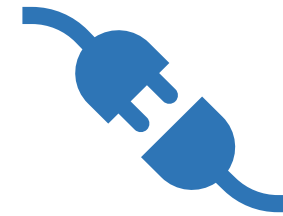
I/O in Maude



Standard input/output



File handling



Sockets

I/O in Maude

```
7      mod VENDING-MACHINE is
8        including VENDING-MACHINE-SIGNATURE .
9        var M : Marking .
10       rl [add-q] : M => M q .
11       rl [add-$] : M => M $ .
12       rl [buy-c] : $ => c .
13       rl [buy-a] : $ => a q .
14       rl [change] : q q q q => $ .
15     endm
```

I/O in Maude

```
5   fmod VENDING-MACHINE-SIGNATURE is
6     sorts Coin Item Marking .
7     subsorts Coin Item < Marking .
8     op __ : Marking Marking -> Marking [assoc comm id: null] .
9     op null : -> Marking .
10    op $ : -> Coin [format (r! o)] .
11    op q : -> Coin [format (r! o)] .
12    op a : -> Item [format (b! o)] .
13    op c : -> Item [format (b! o)] .
14  endfm
```

I/O in Maude

```
5    load vending-machine-signature.maude
6
7    fmod VENDING-MACHINE-GRAMMAR is
8      protecting VENDING-MACHINE-SIGNATURE .
9      protecting NAT .
10     sort Action .
11     op insert $ : -> Action .
12     op insert q : -> Action .
13     op show basket : -> Action .
14     op show credit : -> Action .
15     op buy__(s) : Nat Item -> Action .
16   endfm
```

I/O in Maude

```
5  load vending-machine-grammar.maude
6  load buying-strats.maude
7  load file.maude
8
9  mod VENDING-MACHINE-IO is
10      .
11      .
12      .
95  rl < 0 : X | action : insert $, marking : M, Atts >
96  => < 0 : X | action : idle,
97      marking : downTerm(insertCoin('add-$, upTerm(M)), null), Atts >
98  write(stdout, 0, "one dollar introduced\n") .
```

I/O in Maude

Enter in external environment

```
[Maude> erew vending machine .  
erewrite in VENDING-MACHINE-IO : vending machine .
```

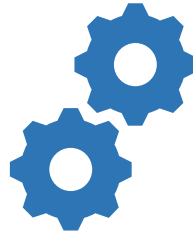
Input	> insert \$
Output	one dollar introduced
	> buy 1 a(s)
	1 apples bought

Parse input?

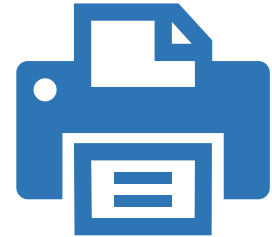
META-LEVEL module



Parse input



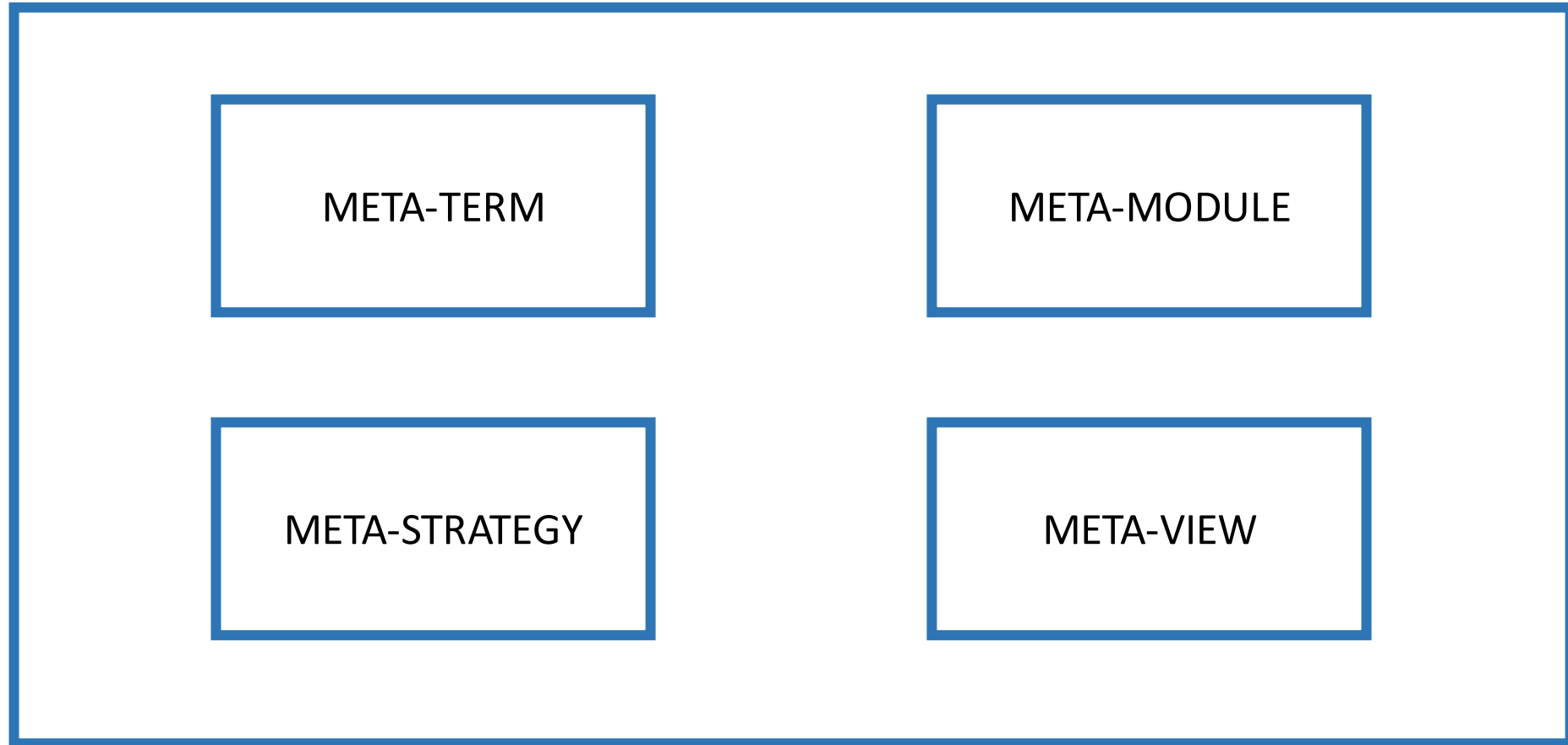
Execute



Print

META-LEVEL module

META-LEVEL



META-LEVEL operations

`metaParse()`



Parse a metarepresentation

`upModule()` / `downModule()`



Move through metarepresentations

Reduce in META-LEVEL



Use operation of the META-LEVEL module

Metalinguage

Funblocks

```
init area(disk(r))
```

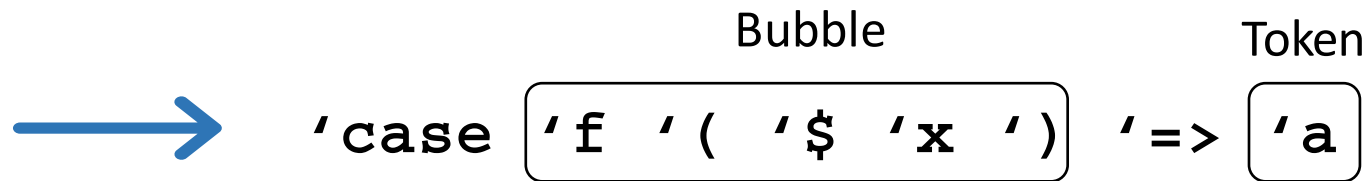
```
case area(disk($d)) => mul(pi, square($d))
```

Maude

```
sorts Const Var Func State Rule Term Expr .
```

Metalanguage

case f(\$x) => a



Tokens are represented by **quoted identifiers (QID)**

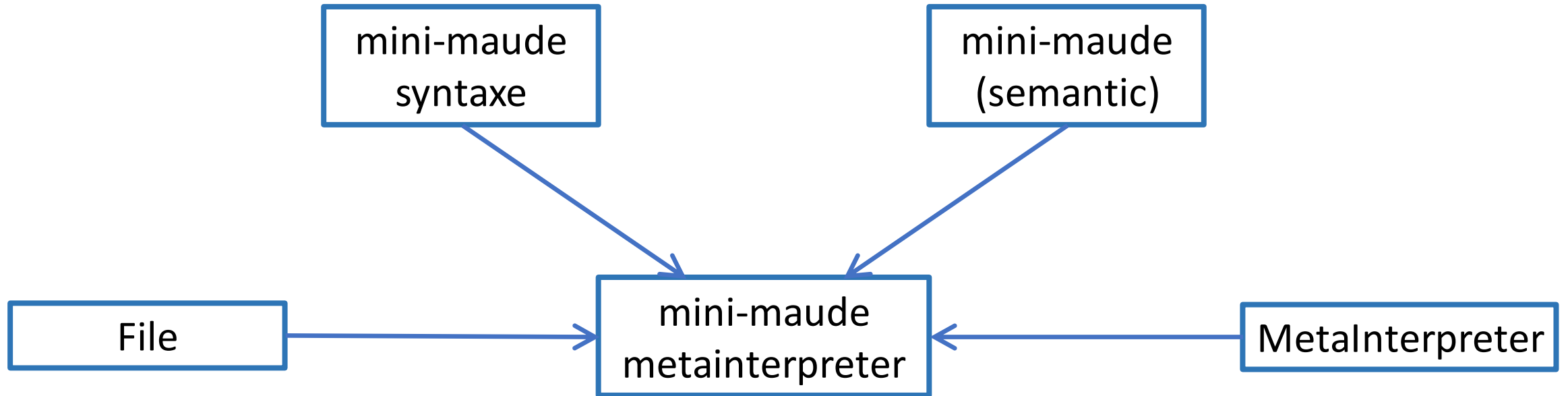
Metalanguage

```
34   op init_ : Bubble -> State .
35   op case_=>_ : Bubble Bubble -> Rule .
36   op _(_) : Token Bubble -> Func .
37   op $_ : Token -> Var .

53   reduce in META-LEVEL :
54     metaParse(upModule('FUNBLOCKS-SYNTAX, false),
55       'case 'f '`( '$ 'x '` ) '=> 'a , anyType) .
```

```
> result: { 'case_=>_[bubble[...]], bubble['a.Qid']], 'Rule }
```

MINI-MAUDE



MINI-MAUDE

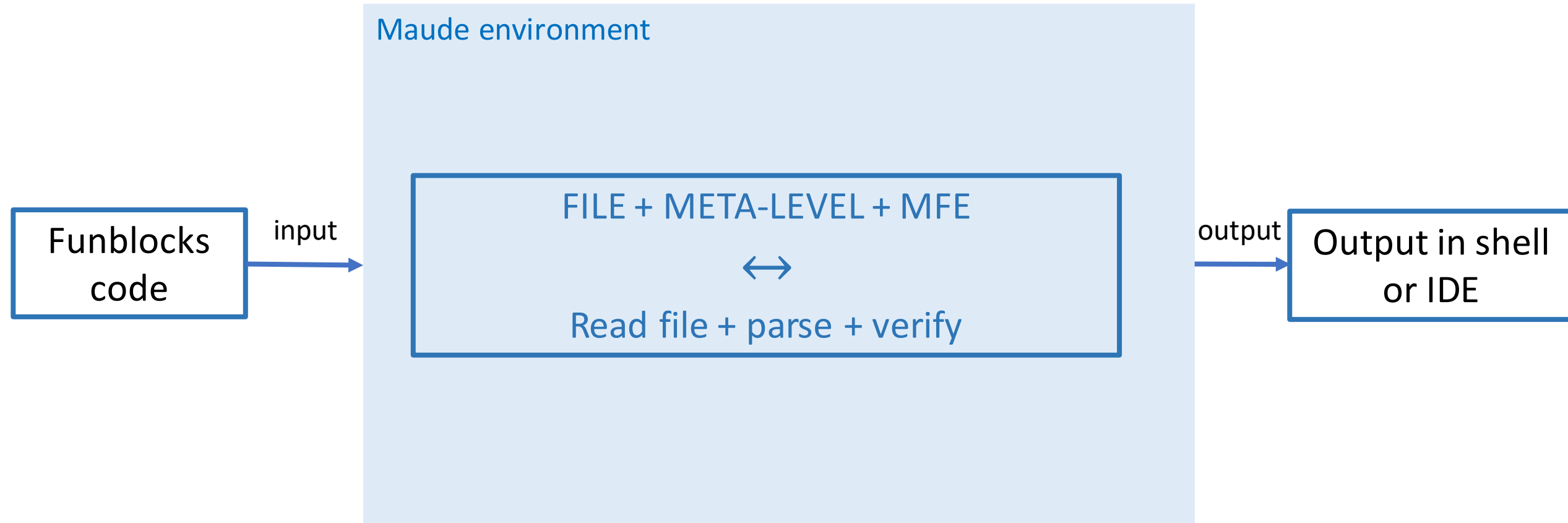
```
Maude> erew minimaude .  
erewrite in MINI-MAUDE-META-INTERPRETER : minimaude .  
MiniMaude Execution Environment
```

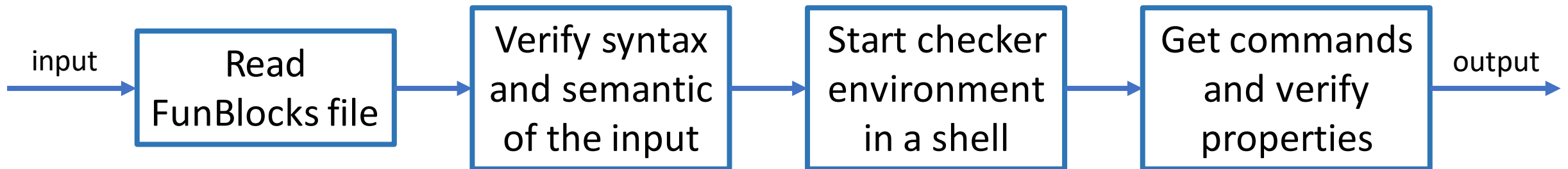
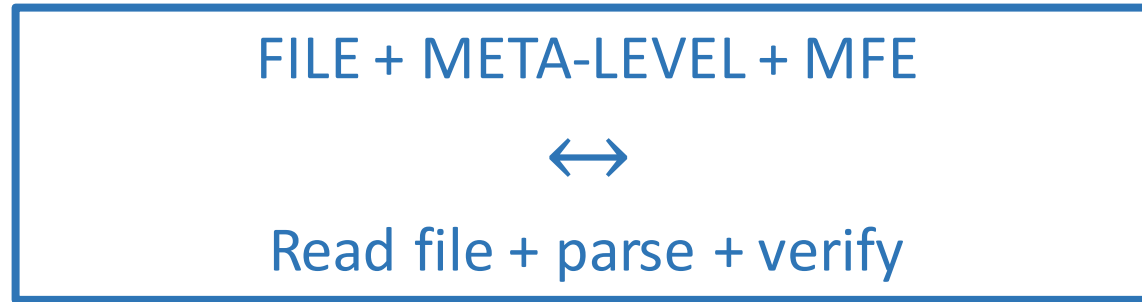
```
minimaude> fmod NAT3 is  
  > sort Nat3 .  
  > op s_ : Nat3 -> Nat3 .  
  > op 0 : -> Nat3 .  
  > eq s s s 0 = 0 .  
  > endfm
```

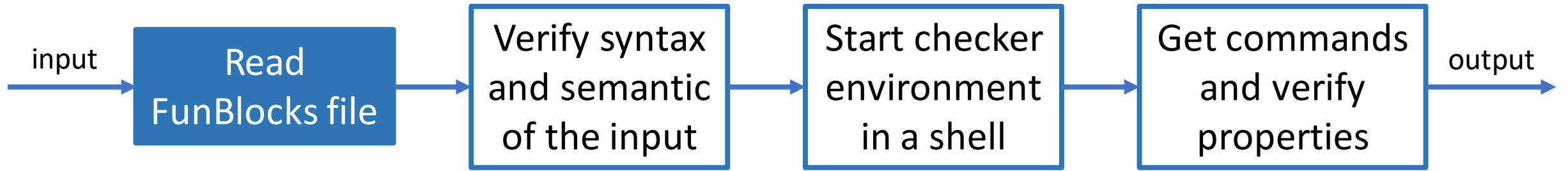
```
Module loaded successfully
```

```
minimaude> reduce s s s s 0 .  
result Nat3: s 0
```

System diagram







FunBlocks file

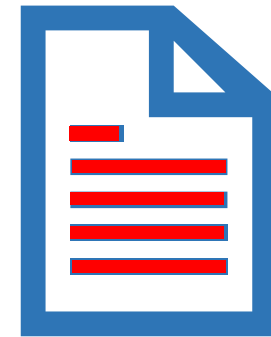


Input

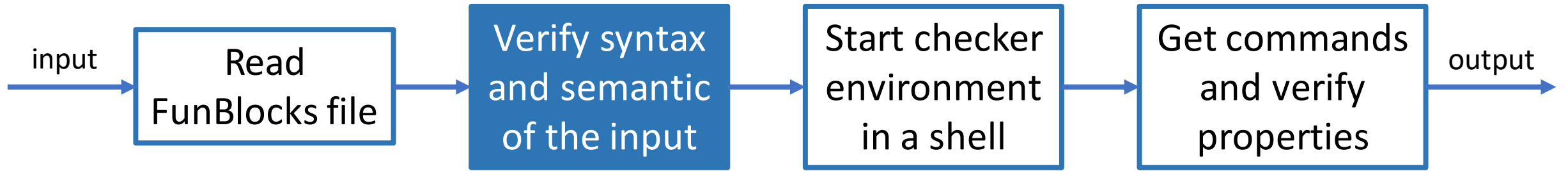
FILE

```
openFile()  
getLine()  
...
```

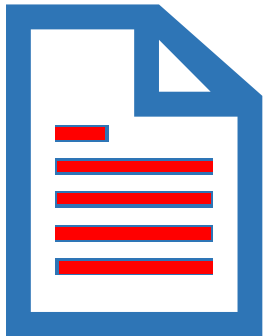
Read file



Output



Read file



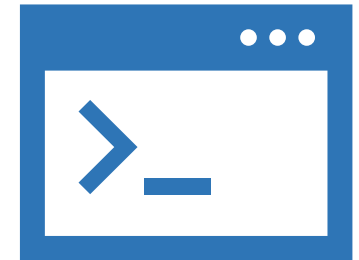
Input

META-LEVEL

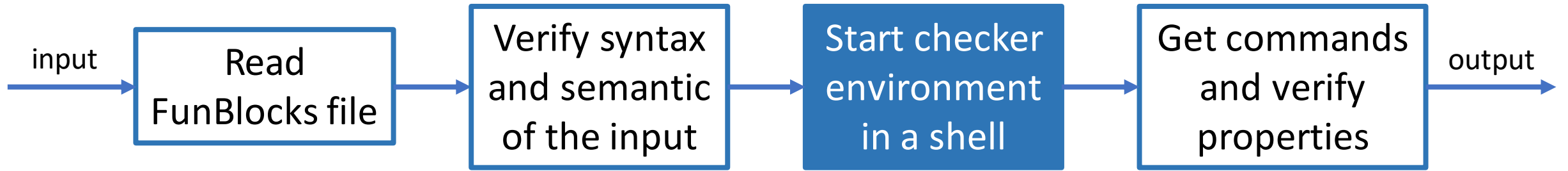
```
metaParse()  
upModule()  
...
```

```
load funblocks-syntax.maude  
load funblocks-semantic.maude
```

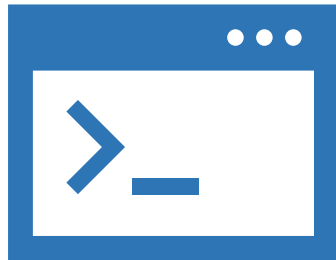
Module inserted



Output

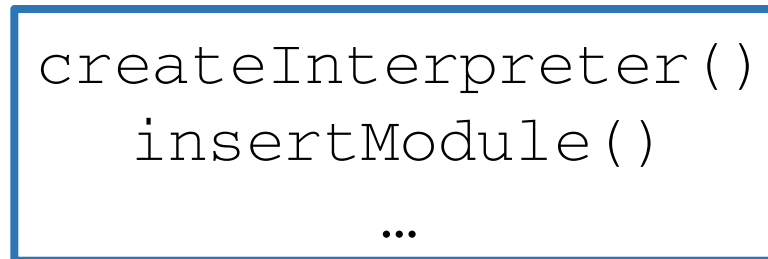


Module inserted

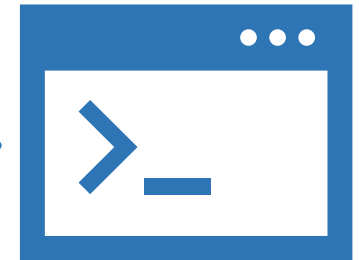


Input

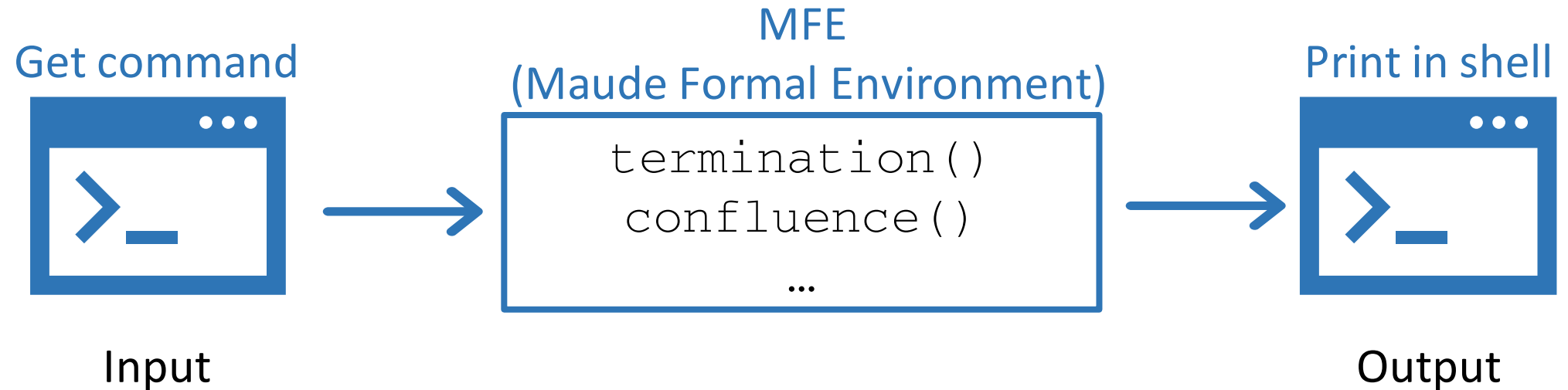
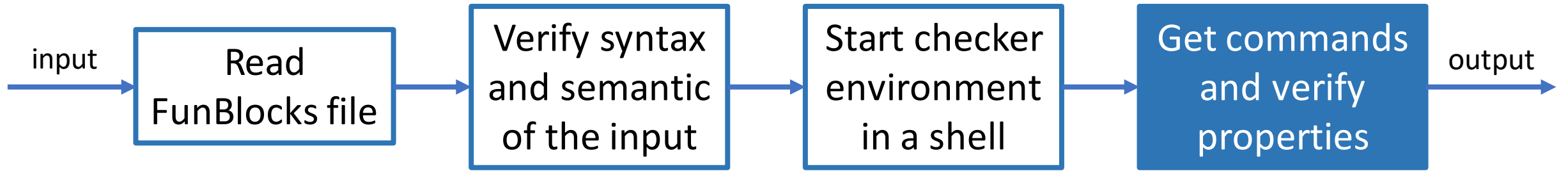
META-INTERPRETER



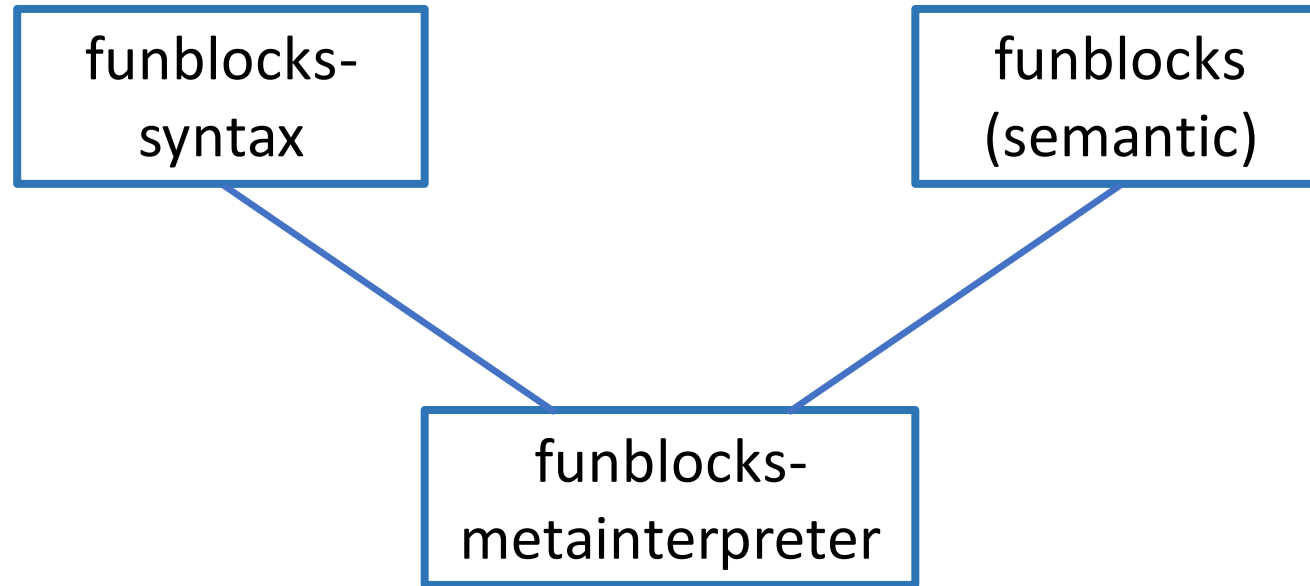
Wait for input



Output



FunBlocks checker system



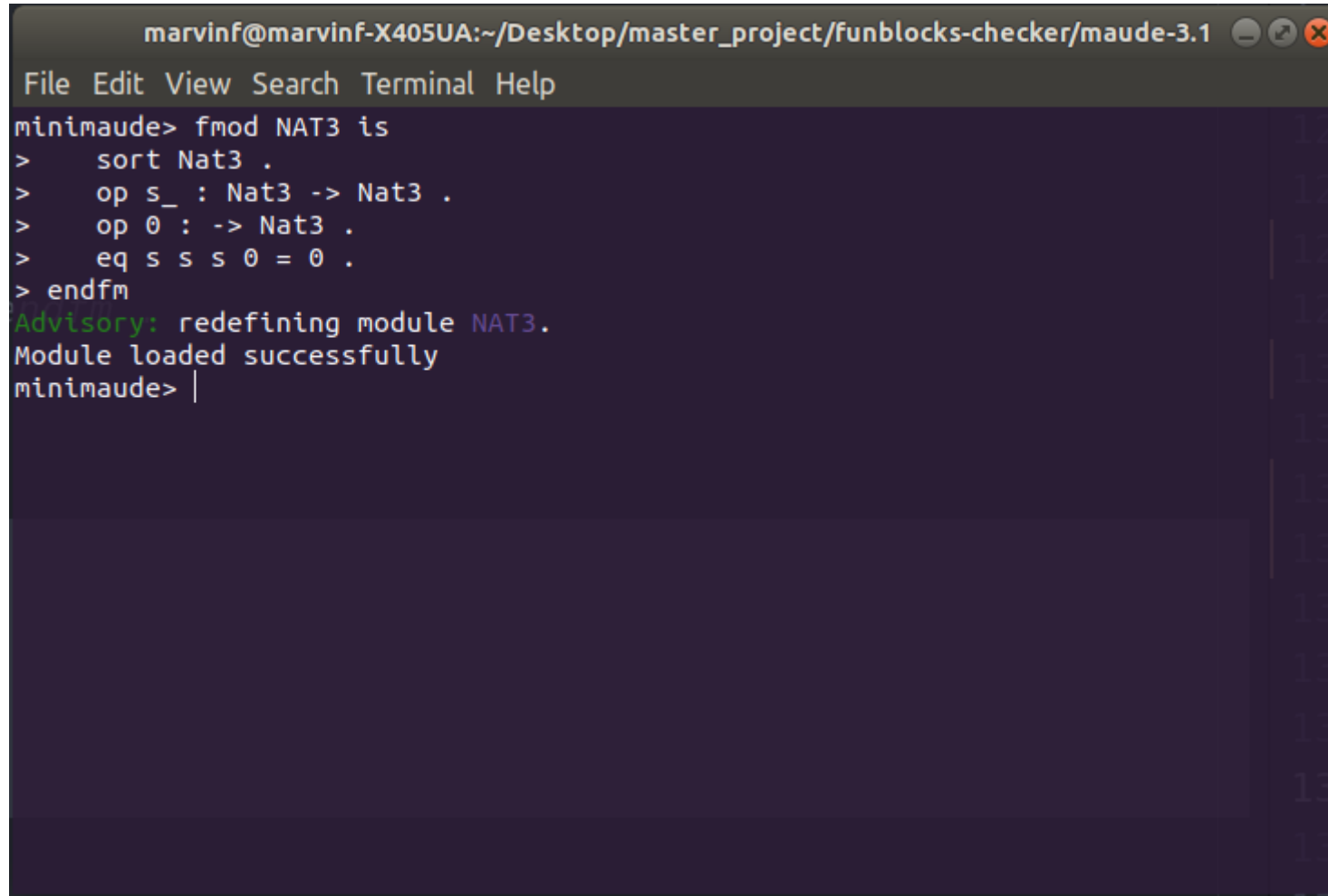
funblocks- metainterpreter

```
3  rl < 0 : FunBlocks | mi: null, st: 0, Atts >
4      wrote(0, 0')
5      createdInterpreter(0, Y, MI)
6  => < 0 : FunBlocks | mi: MI, st: 1, Atts >
7      insertModule(MI, 0, upModule('FUNBLOCKS-SYNTAX, true)) .
8
9  rl < 0 : FunBlocks | mi: MI, st: 1, Atts >
10     insertedModule(0, 0')
11  => < 0 : FunBlocks | mi: MI, st: 2, Atts >
12     getLine(stdin, 0, "funblocks> ") .
```

funblocks- metainterpreter

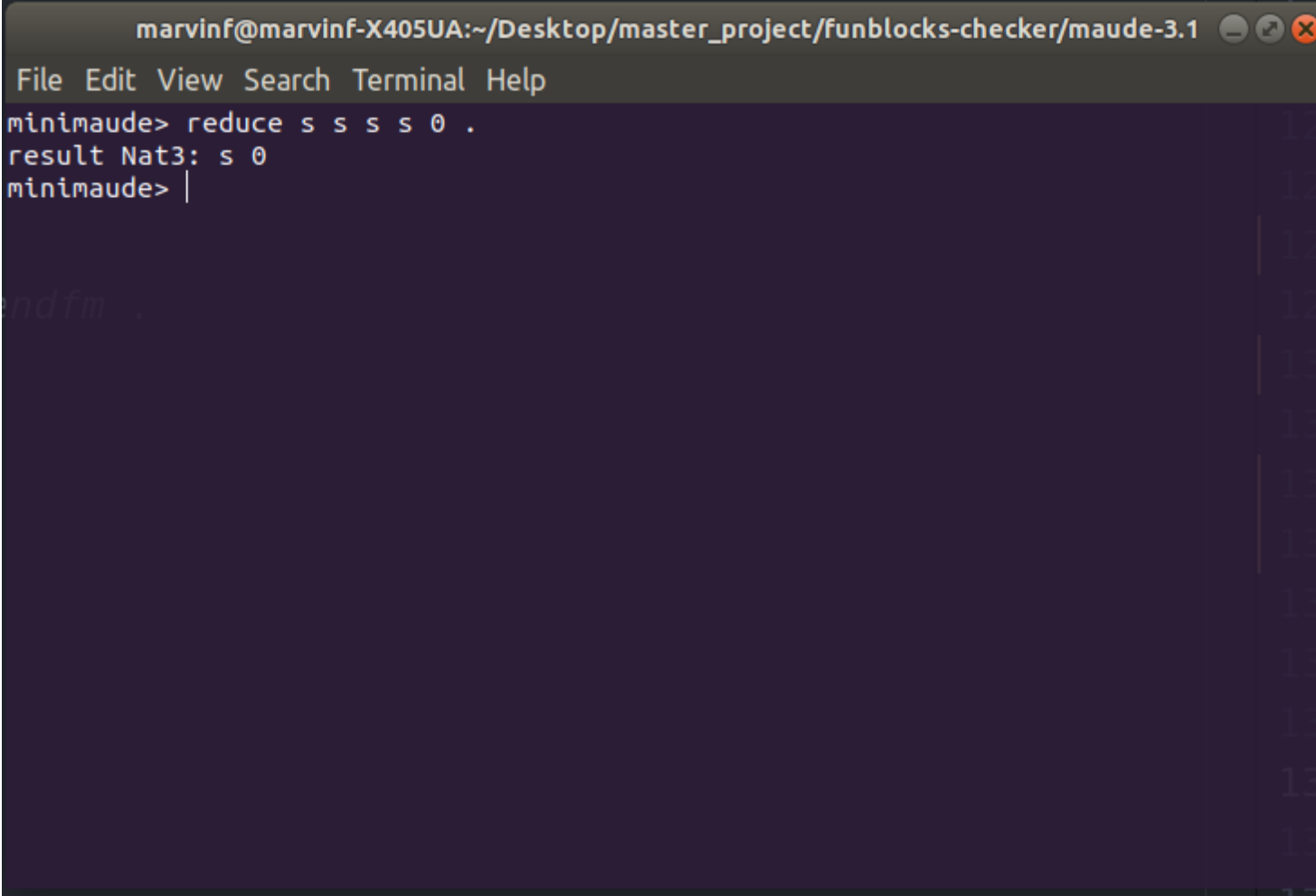
```
3  rl < 0 : FunBlocks | mi: null, st: 0, Atts >
4      wrote(0, 0')
5      createdInterpreter(0, Y, MI)
6  => < 0 : FunBlocks | mi: MI, st: 1, Atts >
7      insertModule(MI, 0, upModule('FUNBLOCKS-SYNTAX, true)) .
8
9  rl < 0 : FunBlocks | mi: MI, st: 1, Atts >
10     insertedModule(0, 0')
11  => < 0 : FunBlocks | mi: MI, st: 2, Atts >
12     getLine(stdin, 0, "funblocks> ") .
```

Mini-Maude: insert module

A screenshot of a terminal window titled "marvinf@marvinf-X405UA:~/Desktop/master_project/funblocks-checker/maude-3.1". The terminal shows the Maude prompt "minimaude>". The user enters "fmod NAT3 is", followed by several lines of module definition: "sort Nat3 .", "op s_ : Nat3 -> Nat3 .", "op 0 : -> Nat3 .", and "eq s s s 0 = 0 .". The user then enters "endfm". The terminal displays the message "Advisory: redefining module NAT3." and "Module loaded successfully". The prompt "minimaude>" is shown again with a cursor. The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help".

```
marvinf@marvinf-X405UA:~/Desktop/master_project/funblocks-checker/maude-3.1
File Edit View Search Terminal Help
minimaude> fmod NAT3 is
>   sort Nat3 .
>   op s_ : Nat3 -> Nat3 .
>   op 0 : -> Nat3 .
>   eq s s s 0 = 0 .
> endfm
Advisory: redefining module NAT3.
Module loaded successfully
minimaude> |
```

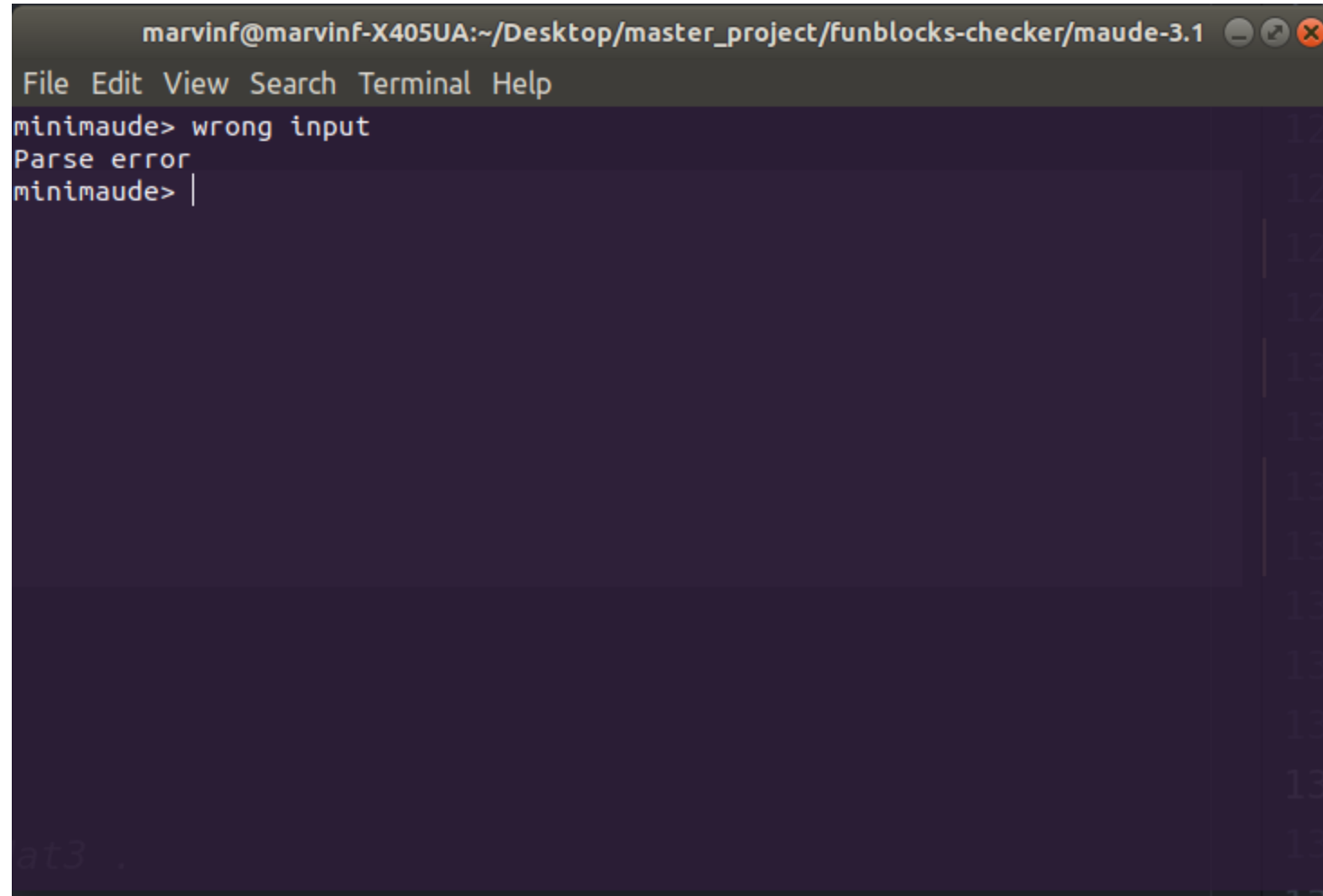

Mini-Maude: use « reduce » command



The screenshot shows a terminal window titled "marvinf@marvinf-X405UA:~/Desktop/master_project/funblocks-checker/maude-3.1". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows the "minimaude" prompt, followed by the command "reduce s s s s 0 .". The output is "result Nat3: s 0". The prompt "minimaude>" is followed by a vertical cursor. On the right side of the terminal, there is a vertical list of line numbers from 12 to 137, with a red vertical bar at the bottom.

```
marvinf@marvinf-X405UA:~/Desktop/master_project/funblocks-checker/maude-3.1
File Edit View Search Terminal Help
minimaude> reduce s s s s 0 .
result Nat3: s 0
minimaude> |
```

Mini-Maude: wrong command



The screenshot shows a terminal window with a dark background. The title bar at the top reads "marvinf@marvinf-X405UA:~/Desktop/master_project/funblocks-checker/maude-3.1" and includes standard window control buttons. Below the title bar is a menu bar with the options "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal displays the following text:

```
minimaude> wrong input
Parse error
minimaude> |
```

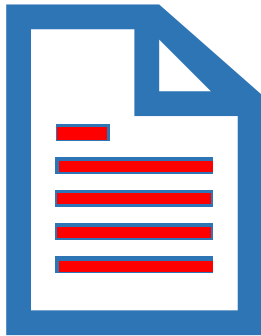
The text "at3 ." is partially visible at the bottom left of the terminal window. On the right side, there is a vertical scrollbar with a list of line numbers, including 12, 13, and 14.

Mini-Maude: quit environment

```
marvinf@marvinf-X405UA:~/Desktop/master_project/funblocks-checker/maude-3.1
File Edit View Search Terminal Help
minimaude> q
goodbye
rewrites: 300 in 20ms cpu (185349ms real) (15000 rewrites/second)
result Portal: <>
=====
reduce in NAT3 : s s s s 0 .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Nat3: s 0
Maude> |
```

Funblocks checker

FunBlocks code



Input



FUN-TO-MAUDE



Module inserted



Output

```
load funblocks-syntax.mau  
load funblocks-semantic.mau
```

Funblocks checker

```
1 case f => g
```

Input

FUN-TO-MAUDE



```
1 mod FUN-RULES is
2   sort Term .
3   op f : -> Term .
4   op g : -> Term .
5   rl f => g .
6 endm
```

Output

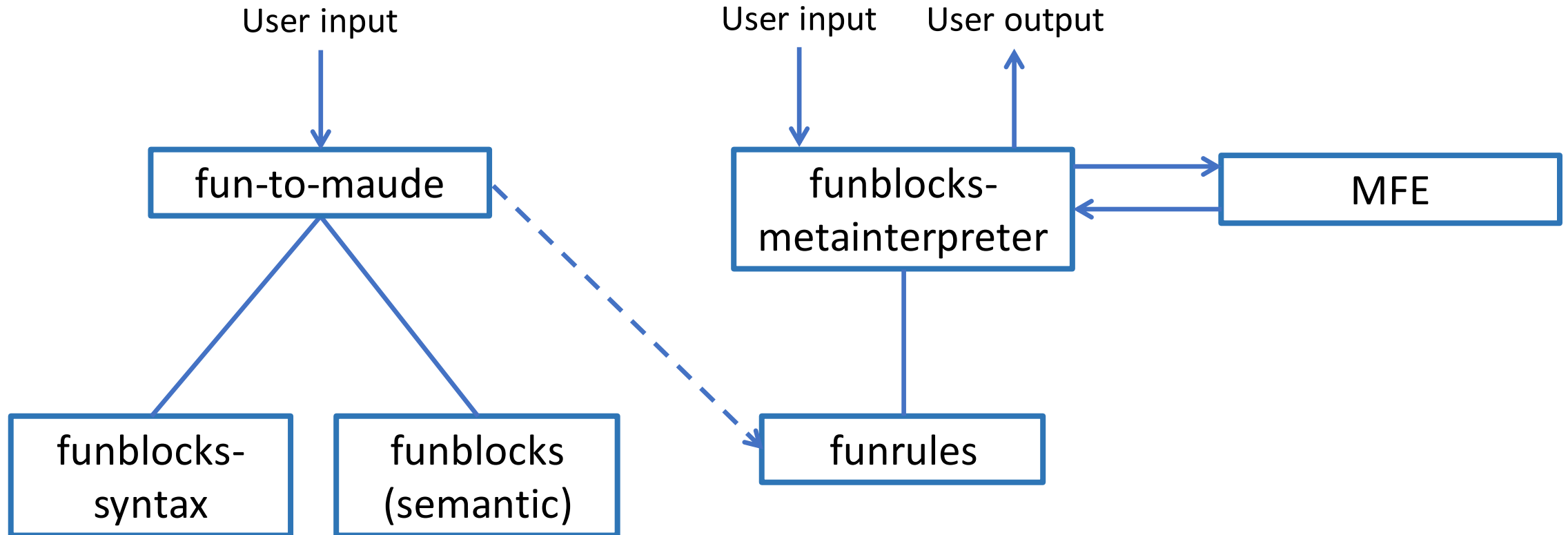
What's next ?

```
1  mod FUN-RULES is
2    sort Term .
3    op f : -> Term .
4    op g : -> Term .
5    rl f => g .
6  endm
```

→ Typing ?

→ Type checking

FunBlocks checker system

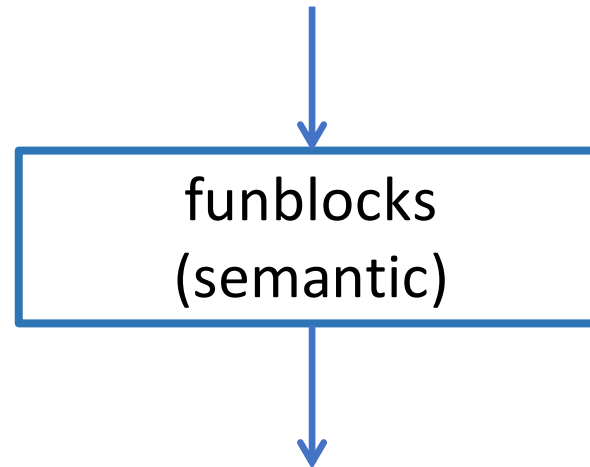


FUNBLOCK-SYNTAX

```
25  sorts Term Var Const Func Decl .
26  subsort Var Func Const < Term .
27
28  op case_=>_ : Term Term -> Decl .
29  op $_ : Token -> Var .
30  op _(_) : Token Token -> Func .
31  op _(_) : Token Term -> Func .
```


FUNBLOCKS

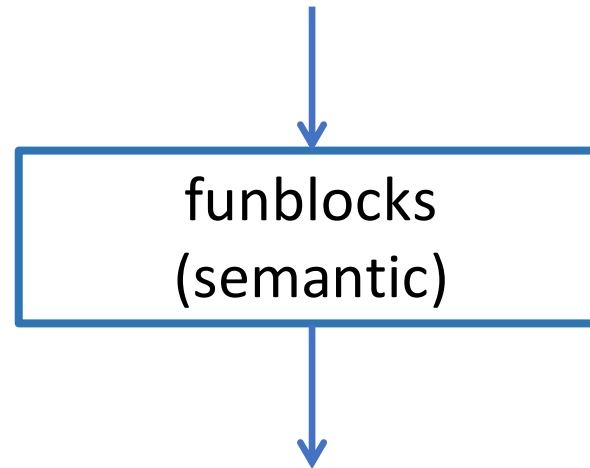
```
'case '$ 'x' => 'id' ( 'a' )
```



```
'var '$ 'x' : 'Untyped' . '\n'<br>'op 'id' : 'Untyped' -> 'Untyped' . '\n'<br>'op 'a' : '-> 'Untyped' . '\n'<br>'rl '$ 'x' => 'id' ( 'a' ) . '\n'
```

FUNBLOCKS

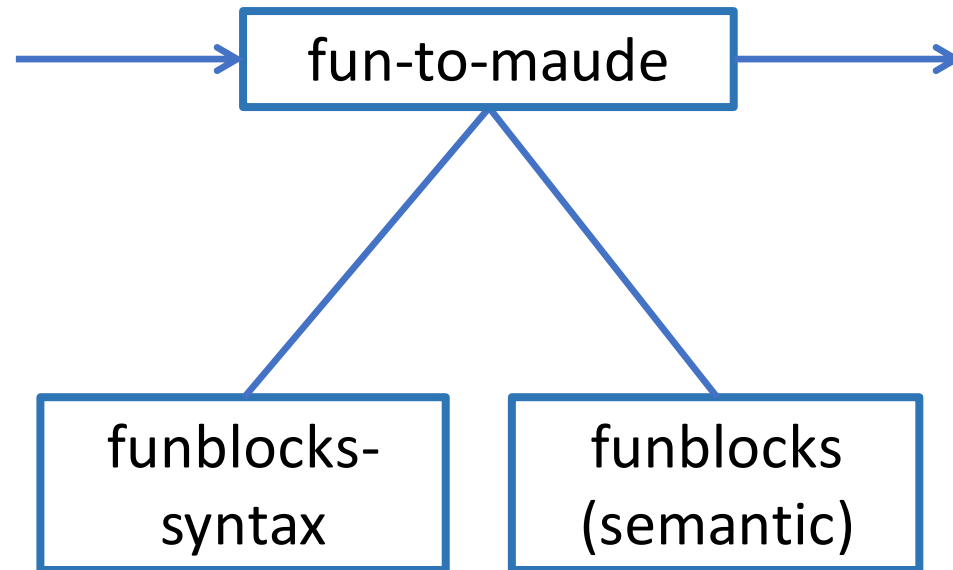
`'case 'compose '('f ', 'g ', '$ 'x ') '=> 'f '('g '('$ 'x ') ')`



`noParse()`

FUN-TO-MAUDE

```
init dist(a,d)  
case $ x => id(a)
```



```
1  mod FUN-RULES is  
2  var '$x' : Untyped .  
3  op 'id' : Untyped -> Untyped .  
4  op 'a' : -> Untyped .  
5  rl '$x' => 'id('a)' .  
6  endm
```

FUNRULES

```
1  mod FUN-RULES is
2  var '$x : Untyped .
3  op 'id : Untyped -> Untyped .
4  op 'a : -> Untyped .
5  rl '$x => 'id('a) .
6  endm
```



```
1  mod FUN-RULES is
2  inc FUNTYPES .
3  sorts Numbers String .
4  var '$x : Untyped .
5  op 'id : Untyped -> Untyped .
6  op 'a : -> Untyped .
7  rl '$x => 'id('a) .
8  endm
```

Pre-processing (script)

case id(\$x) => \$x



case id(\$ x) => \$ x

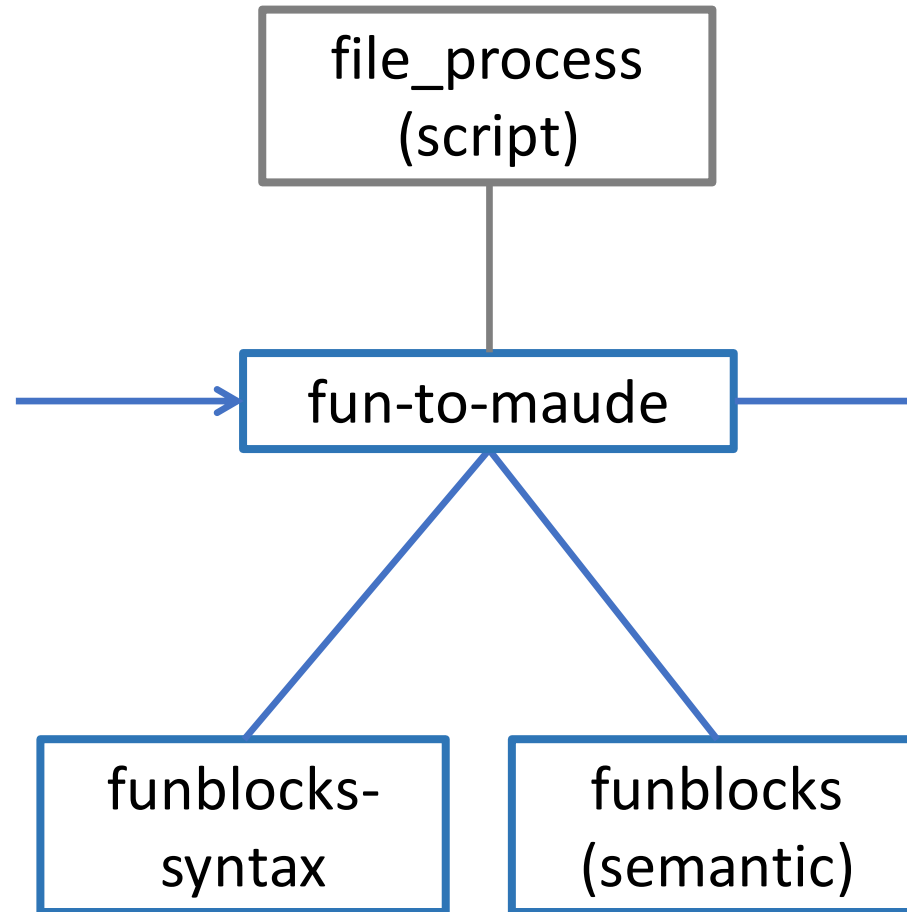
Post-processing (script)

`case id($'x) => $'x`  `case id($x) => $x`

`op a : -> Nat`
`op a : -> Nat`  `op a : -> Nat`

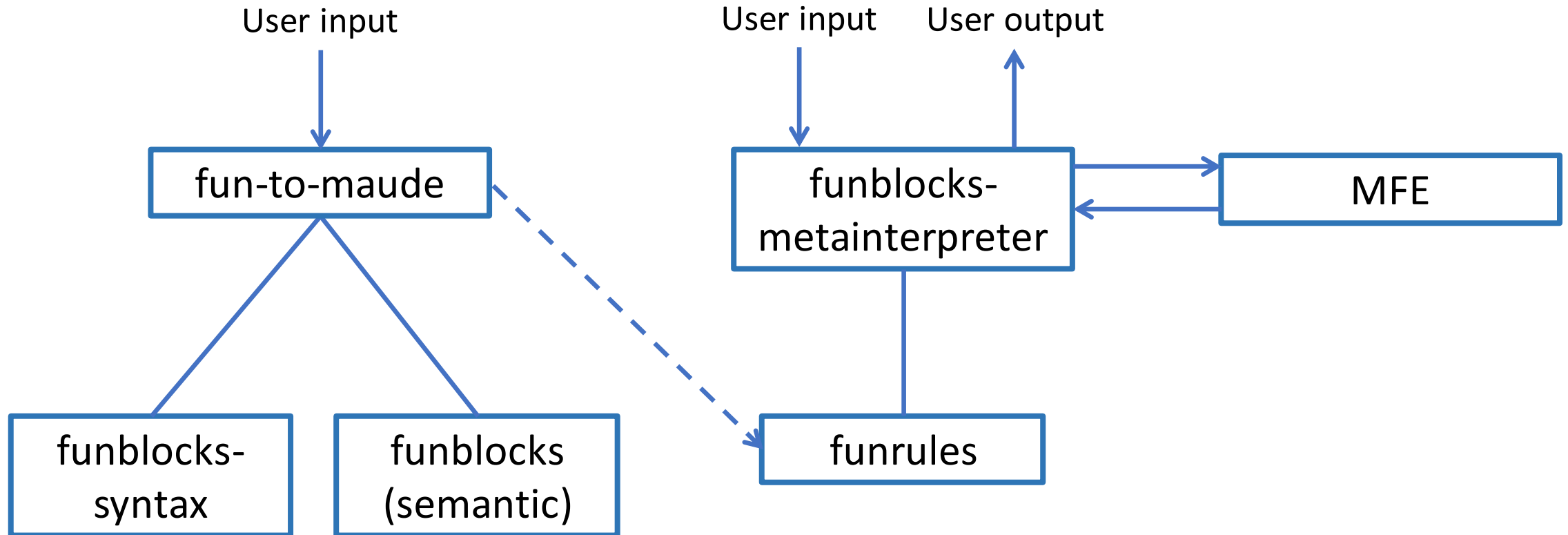
FUN-TO-MAUDE

```
init dist(a,d)  
case $x => id(a)
```



```
1  mod FUN-RULES is  
2    var $x : Untyped .  
3    op id : Untyped -> Untyped .  
4    op a : -> Untyped .  
5    rl $x => id(a) .  
6  endm
```

FunBlocks checker system



Static Typing

```
type List $T :: empty | cons $T (List $T)
rule ajoute $T :: $T -> List $T => List $T
case ajoute($x, cons($y, $tail)) => cons($y, $tail)
```

```
{ ajoute:
  [ TypeVarRef { range: [Object], label: 'T' },
    TypeDeclRef { range: [Object], name: 'List', arguments: [Array] },
    TypeDeclRef { range: [Object], name: 'List', arguments: [Array] } ] }
```

Static Typing

```
case ajoute($x, cons($y, $tail)) => cons($y, $tail)
```

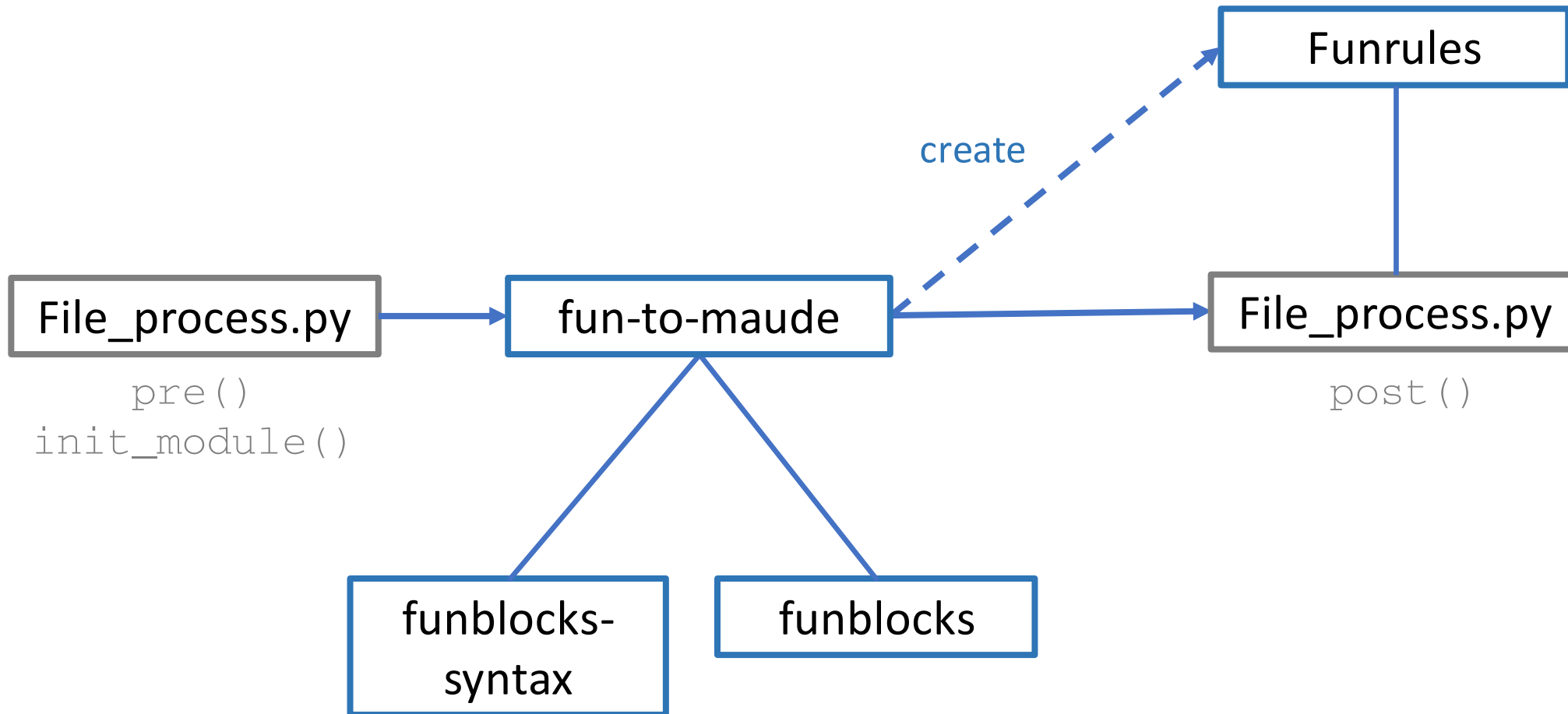
```
{ ajoute: [ TypeVarRef { range: null, label: 'T0' }, <2 empty items> ] }
```

```

5  fmod MY-LIST{T :: TRIV} is
6
7  protecting LIST{T} .
8  protecting NAT .
9  protecting QID .
10
11  subsort Qid < T$Elt < List{T} .
12  vars $x $y : T$Elt .
13  vars $tail : List{T} .
14
15  --- constructed with 'type'
16  op empty : -> List{T} .
17  op ajoute : T$Elt List{T} -> List{T} .
18  op cons : T$Elt List{T} -> List{T} .
19
20  --- constructed with 'case'
21  eq ajoute($x, cons($y, $tail)) = cons($y, ajoute($x, $tail)) .
22  eq ajoute($x, empty) = cons($x, empty) .
23
24  endfm

```

Execution



Maude tools

Inductive Theorem Prover (ITP)

Sufficient Completeness Checker (SCC)

Church-Rosser Checker (CRC)

Coherence Checker (ChC)

Maude Termination Tool (MTT)



Maude Formal Environment (MFE)

Maude Formal Environment

Maude 3.1 (last version)

Inductive Theorem Prover (ITP)

Church-Rosser Checker (CRC)

Coherence Checker (ChC)

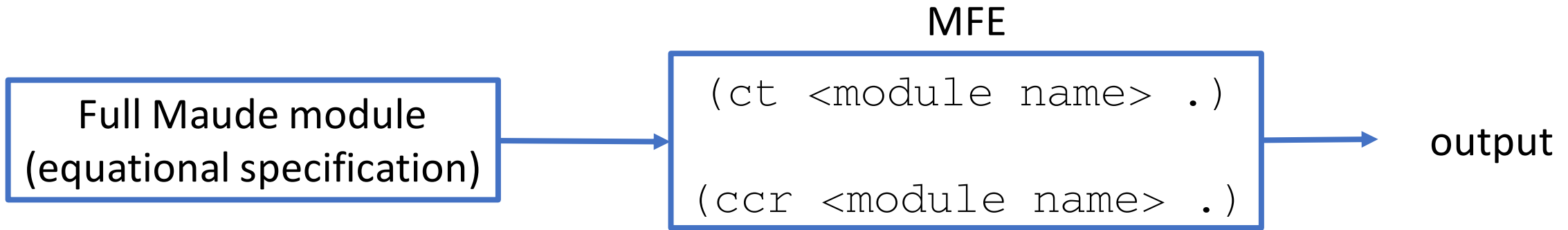
Maude++ 2.7

Sufficient Completeness Checker (SCC)

Maude Termination Tool (MTT)

↕
AProVE

Maude Formal Environment



Works only with equational specifications

Church-Rosser Checker

```
1  mod TEST2 is
2    sort Generic .
3    op id : Nat -> Nat .
4    var x : Nat .
5    op a : -> Nat .
6    eq id(x) = x .
7    eq id(x) = a .
8  endm
```

```
Maude> (ccr TEST2 .)
rewrites: 36429 in 4ms cpu (3ms real) (9107250 rewrites/second)
Church-Rosser check for TEST2
The following critical pairs must be proved joinable:
  cp TEST21
    a
  = x:Nat .
    The module is sort-decreasing.
```


Church-Rosser Checker

```
1  mod TEST3 is
2    sort Generic .
3    op id : Nat -> Nat .
4    var x : Nat .
5    op a : -> Nat .
6    eq id(x) = x .
7  endm
```

```
Maude> (ccr TEST3 .)
rewrites: 36299 in 4ms cpu (3ms real) (9074750 rewrites/second)
Church-Rosser check for TEST3
  All critical pairs have been joined.
  The specification is locally-confluent.
  The module is sort-decreasing.
```

Church-Rosser Checker

```
1  mod TEST3 is
2    sort Generic .
3    op id : Nat -> Nat .
4    var x : Nat .
5    op a : -> Nat .
6    eq id(x) = x .
7  endm
```

```
Maude> (ccr TEST3 .)
rewrites: 36299 in 4ms cpu (3ms real) (9074750 rewrites/second)
Church-Rosser check for TEST3
  All critical pairs have been joined.
  The specification is locally-confluent.
  The module is sort-decreasing.
```

Termination tool

```
14 (fmod TEST2 is
15   sort Nat .
16   vars x y : Nat .
17   op a : -> Nat .
18   op b : -> Nat .
19   op f : Nat Nat Nat -> Nat .
20   op g : Nat Nat -> Nat .
21
22   eq f(a,b,x) = f(x,x,x) .
23   eq g(x,y) = x .
24   eq g(x,y) = y .
25 endfm)
```

```
rewrites: 66 in 4ms cpu (4ms real) (16500 rewrites/second)
The MTT has been set as current tool.

rewrites: 29 in 4ms cpu (4ms real) (7250 rewrites/second)
aprove is now the current external tool.

rewrites: 75 in 0ms cpu (1ms real) (~ rewrites/second)
Success: The module TEST2 is non-terminating.

rewrites: 66 in 4ms cpu (0ms real) (16500 rewrites/second)
The CRC has been set as current tool.

rewrites: 126 in 0ms cpu (3ms real) (~ rewrites/second)
Church-Rosser check for TEST2
The following critical pairs must be proved joinable:
  cp TEST22
    x:Nat
    = y:Nat .
    The module is sort-decreasing.
```

Equation vs rules (in Maude)



Equations

Represent theory

Terminating and confluent



Rules

Represent states changes

No constraints on termination and confluence

Reductions using rewrite logic

Parsing in Maude

reduce

```
red 1 + (1 + 1) .
```

rewrite

```
rew [6] $ $ q q .
```

Precedence and gathering

```
sort Nat .  
ops 1 2 3 : -> Nat .  
ops _+_ *_ : Nat Nat -> Nat .
```



$$\begin{array}{l} 1 + 2 * 3 \left\{ \begin{array}{l} 1 + (2 * 3) \\ (1 + 2) * 3 \end{array} \right. ? \\ \\ 1 + 2 + 3 \left\{ \begin{array}{l} 1 + (2 + 3) \\ (1 + 2) + 3 \end{array} \right. ? \end{array}$$


Precedence and gathering

prec \longrightarrow Indication of « priority » using a natural number

gather \longrightarrow E : precedence value \leq precedence value of the operator
e : precedence value $<$ precedence value of the operator
& : any precedence value

Precedence and gathering

```
op _+_ : Nat Nat -> Nat [prec 33 gather (E e)] .  
op _*_ : Nat Nat -> Nat [prec 31 gather (E e)] .
```


$$\begin{aligned} 1 + 2 * 3 &= 1 + (2 * 3) \\ 1 + 2 + 3 &= (1 + 2) + 3 \end{aligned}$$

Default precedence values

prec



0 for the constants (`zero`)

0 for mixfix operator which not begin **nor** end with `_ ((_))`

41 for mixfix operator which begin **or** and with `_ (to _ : _)`
-> 15 for unary operator (`not _`)

41 : for mixfix operator which begin **and** end with `_ (_ + _)`

Default gathering values

gather \longrightarrow & for default prefix operators
& for operator which not begin nor end with `_ ((_))`
E for adjacent `_` or `_` at the edges `(_?_:_ is (E & E))`



Special cases of **binary** operators which **start and end with `_`** and have **precedence value > 0** :

`(e E)` if it has the `assoc` attribute

If not, we have to look at the subsorts relationship...

Default gathering values

If we have `Int < IntList`

<code>op _<:_ : Int IntList -> IntList .</code>		<code>(e E)</code>
<code>op _:>_ : IntList Int -> IntList .</code>		<code>(E e)</code>

Pipeline

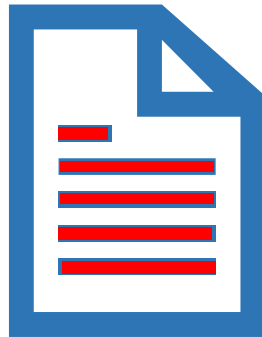
FunBlocks file



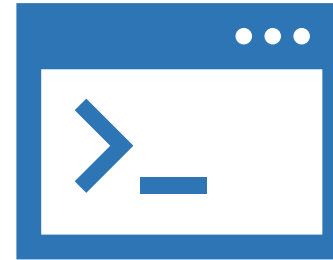
Input



Maude file



Maude Checker tool
(MFE)



User input



Feedback

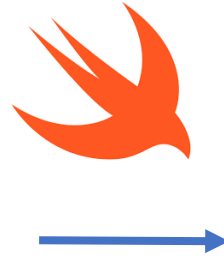


Pipeline

FunBlocks file

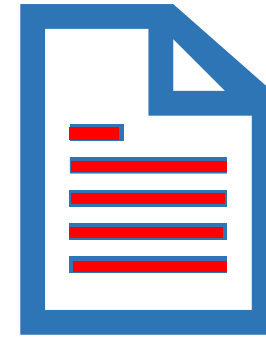


Input



Parser (Swift)

Maude file



```
rule add :: Nat -> Nat => Nat
case add($x, zero) => $x
```



```
op add : Nat Nat -> Nat .
eq add($x, zero) = $x .
```

```

1  type Nat :: a | b | succ Nat
2  rule f :: Nat -> Nat -> Nat => Nat
3  rule g :: Nat -> Nat => Nat
4  case f(a,b,$x) => f($x,$x,$x)
5  case g($x,$y) => $x
6  case g($x,$y) => $y

```

```

1  (fmod Nat is
2    sort Nat .
3  op a : -> Nat .
4  op b : -> Nat .
5  op succ : Nat -> Nat .
6  endfm)
7  (fmod FUNRULES is
8    including Nat .
9    op f : Nat Nat Nat -> Nat .
10   op g : Nat Nat -> Nat .
11   var $x : Nat .
12   var $x : Nat .
13   var $x : Nat .
14   var $x : Nat .
15   eq f(a, b, $x) = f($x, $x, $x) .
16   var $x : Nat .
17   var $y : Nat .
18   eq g($x, $y) = $x .
19   var $x : Nat .
20   var $y : Nat .
21   eq g($x, $y) = $y .
22   endfm)

```

```
Success: The module FUNRULES is non-terminating.

rewrites: 66 in 4ms cpu (0ms real) (16500 rewrites/second)
The CRC has been set as current tool.

rewrites: 128 in 4ms cpu (4ms real) (32000 rewrites/second)
Church-Rosser check for FUNRULES
The following critical pairs must be proved joinable:
  cp FUNRULES2
    $x:Nat
    = $y:Nat .
    The module is sort-decreasing.
```

Term Rewriting System R:

```
[$x, $y]
f-`[Nat`]-`[Nat`]-`[Nat`](a, b,
$x) -> f-`[Nat`]-`[Nat`]-
`[Nat`]($x, $x, $x)
g-`[Nat`]-`[Nat`]($x, $y) -> $x
g-`[Nat`]-`[Nat`]($x, $y) -> $y
```

Termination of R to be shown.

R ->Dependency Pair Analysis

Found an infinite P-chain over R:

Thus, s starts an infinite chain.

Non-Termination of R could be shown.


```

1 type Nat :: zero | succ Nat
2 type Tree $T :: empty | leaf $T | node (Tree $T) (Tree $T)
3 rule depth $T :: Tree $T => Nat
4 rule add :: Nat -> Nat => Nat
5 case add($x, add($y, zero)) => $x
6 case depth(empty) => zero
7 case succ($z) => zero

```

```

1 (fmod Nat is
2   sort Nat .
3   op zero : -> Nat .
4   op succ : Nat -> Nat .
5   endfm)
6 (view $T from TRIV to Nat is sort Elt to Nat
7   (fmod Tree{T :: TRIV} is
8     sort Tree{T} .
9     op empty : -> Tree{T} .
10    op leaf : T$Elt -> Tree{T} .
11    op node : Tree{T} Tree{T} -> Tree{T} .
12    endfm)
13   (fmod FUNRULES is
14     including Nat .
15     including Tree{$T} .
16     op depth : Tree{$T} -> Nat .
17     op add : Nat Nat -> Nat .
18     var $x : Nat .
19     var $y : Nat .
20     eq add($x, add($y, zero)) = $x .
21     eq depth(empty) = zero .
22     var $z : Nat .
23     eq succ($z) = zero .
24     endfm)

```

```
Success: The module FUNRULES is terminating.  
  
rewrites: 66 in 0ms cpu (1ms real) (~ rewrites/second)  
The CRC has been set as current tool.  
  
rewrites: 114 in 4ms cpu (4ms real) (28500 rewrites/second)  
Church-Rosser check for FUNRULES  
    All critical pairs have been joined.  
    The specification is locally-confluent.  
    The module is sort-decreasing.
```

```
1 type Nat :: a | b | c | zero | succ Nat
2 rule f :: Nat -> Nat => Nat
3 rule g :: Nat => Nat
4 case f($x, $x) => a
5 case f($x, g($x)) => b
6 case c => g(c)
```

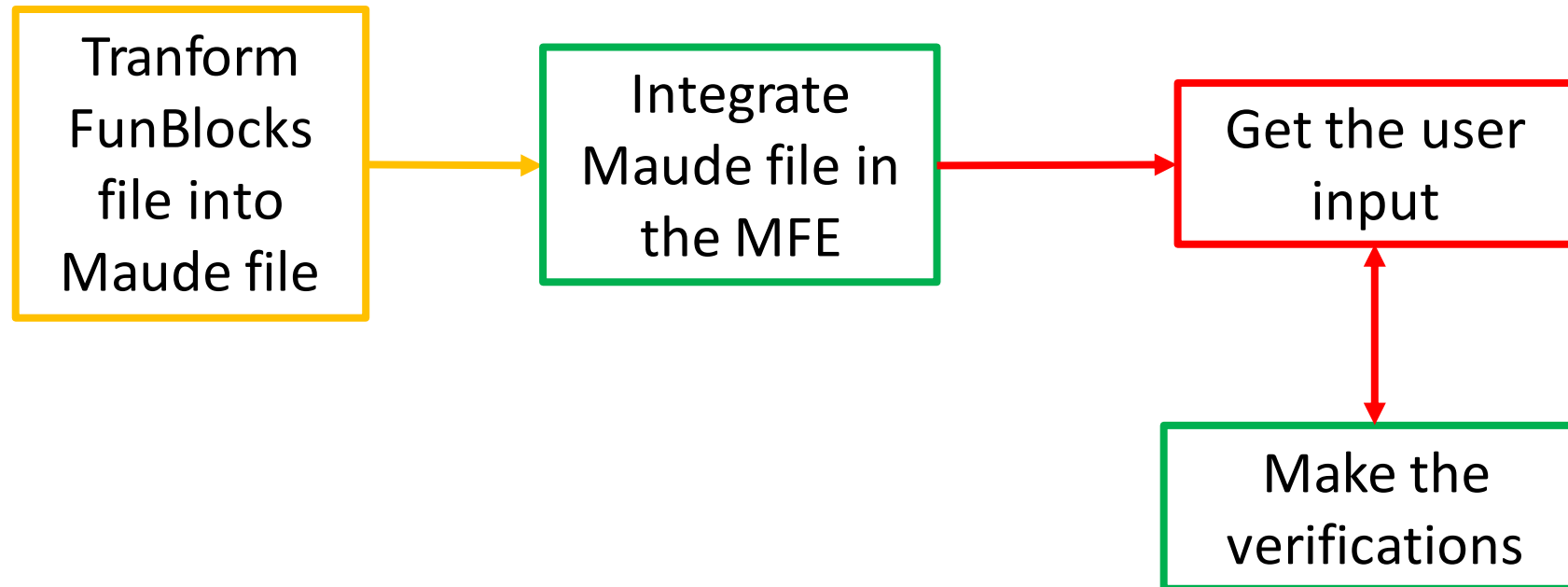
No critical pairs to show. One of the checking commands must be successfully executed previously.

Fatal error: stack overflow.

This can happen because you have an infinite computation, say a runaway recursion, or model checking an infinite model. It can also happen because the stacksize limit in your environment is set too low for the computation you are trying to do. You can find the value of your stacksize with the tcsh command 'limit stacksize' or the bash command 'ulimit -s'.

Depending on your operating system configuration you may be able to increase your stacksize with the tcsh command 'unlimit stacksize' or the bash command 'ulimit -s unlimited'.

« TODO » list



```

1  (fmod Bool is ... endfm)
2  (fmod Rel is ... endfm)
3  (view $T from TRIV to Rel is sort Elt to Rel . endv)
4  (fmod List{T :: TRIV} is
5      ...
6      op cons : List{T} T$Elt -> List{T} .
7  endfm)
8  (fmod FUNRULES is
9      including Bool .
10     including Rel .
11     including List{$T} .
12     op size : List{$T} -> Rel .
13     op isEmpty : List{$T} -> Bool .
14     eq size(empty) = zero .
15     var $x : Rel .
16     eq size(cons(empty, $x)) = succ(zero) .
17  endfm)

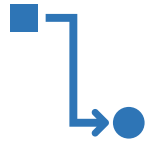
```

```

1  (fmod Bool is ... endfm)
2  (fmod Rel is ... endfm)
3  (view $T1 from TRIV to Bool is sort Elt to Bool . endv)
4  (view $T2 from TRIV to Rel is sort Elt to Rel . endv)
5  (fmod List{T :: TRIV} is
6    ...
7    op cons : List{T} T$Elt -> List{T} .
8  endfm)
9  (fmod FUNRULES is
10   including Bool .
11   including Rel .
12   including List{$T1} .
13   including List{$T2} .
14   op size : List{$T2} -> Rel .
15   op isEmpty : List{$T2} -> Bool .
16   eq size(empty) = zero .
17   var $x : Rel .
18   eq size(cons(empty, $x)) = succ(zero) .
19 endfm)

```

« TODO » list



Type inference of variables



Generic types



Check FunBlocks code before verifications

References

1. Didier Buchs, modelisation verification course material, 2018
2. Dimitri Racordon, Emmanouela Stachtiri, Damien Morard, Didier Buchs, Functional Block Programming and Debugging, 2020
3. Nachum Dershowitz, Jean-Pierre Jouannaud, Rewrite Systems, 1990
4. Nachum Dershowitz, Computing with Rewrite Systems, 1985
5. Terese, Term Rewriting Systems, 2003
6. Thomas Sternagel and Harald Zankl, KBCV-Knuth-Bendix Completion Visualizer, 2012
7. Thomas Artsa, Jürgen Giesl, Termination of term rewriting using dependency pairs, 2000

References

8. Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke, Automated Termination Proofs with AProVE, 2004
9. D. Kapur, P. Narendran, Path ordering for proving termination of term rewriting systems, 1985
10. Jeremy Dick, John Kalmus and Ursula Martin, Automating the Knuth Bendix ordering, 1990
11. E. Contejean, P. Courtieu, J. Forest, O. Pons, X. Urbain, Automated Certified Proofs with CiME3, 2011
12. F. Chalub, C. Braga, Maude MSOS Tool, 2005
13. S. Winkler, A. Middeldorp, Tools in Term Rewriting for Education, 2020
14. A. Salvador, L. Salvador, Term Rewriting Systems .Net Framework, 2013

References (links)

Database of Rewriting Systems

- <http://rewriting.loria.fr/systems.html>
- <http://www.jaist.ac.jp/~hirokawa/tool/>

Knuth-Bendix Completion Visualizer

- <http://cl-informatik.uibk.ac.at/software/kbcv/>

Knuth-Bendix Completion subject-based thesis

- <https://homepage.divms.uiowa.edu/~astump/papers/thesis-wehrman.pdf>

References (links)

Prolog implementation of the Knuth-Bendix completion procedure

- <https://www.metalevel.at/trs/>

Maude tools

- <http://maude.lcc.uma.es/CRChC/>
- <http://www.lcc.uma.es/%7Eduran/MTT/>
- <http://maude.sip.ucm.es/debugging/>

Wikipedia

- https://fr.wikipedia.org/wiki/Compl%C3%A9tion_de_Knuth-Bendix
- https://fr.wikipedia.org/wiki/Paire_critique

References (links)

TRS tool:

- <http://tfmserver.dsic.upv.es:8080/Home.html>

Make FunBlocks alive

Marvin FOURASTIE

Master project