Make FunBlocks alive

Marvin FOURASTIE

Master project

Motivations

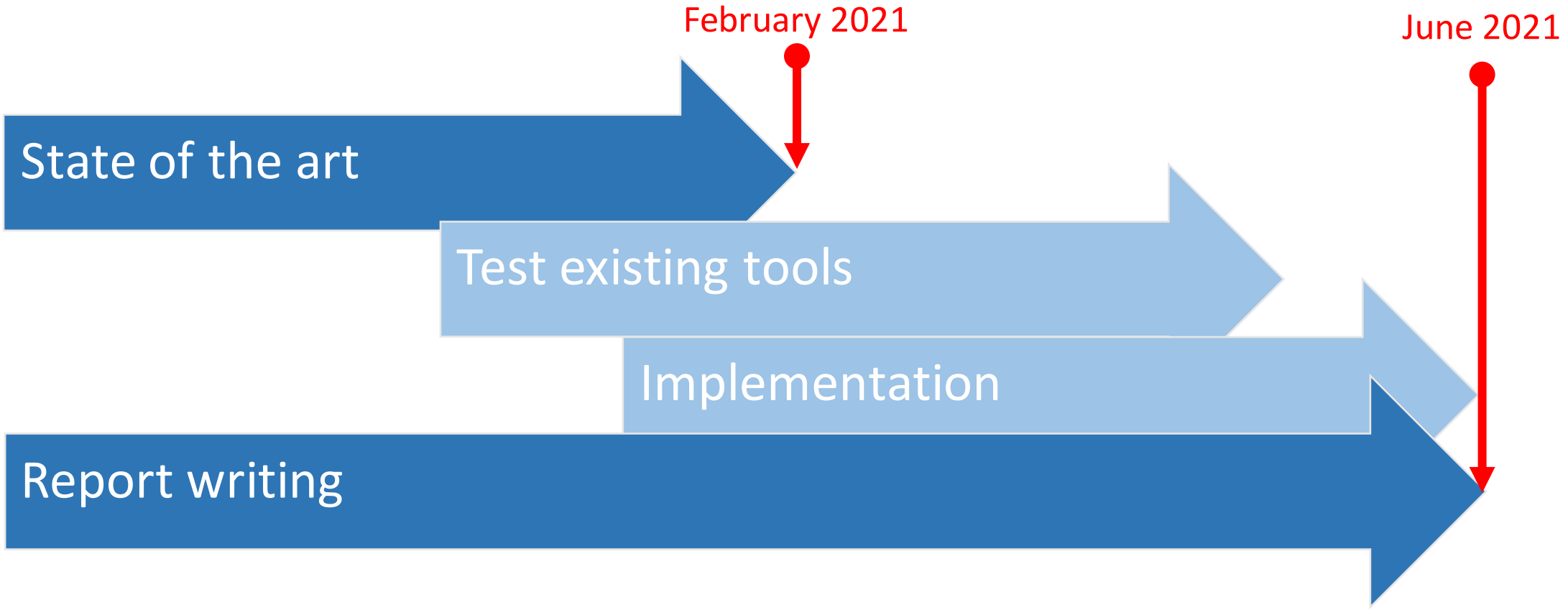
Educational purpose

Imperative paradigm alternative

Based on rewrite systems

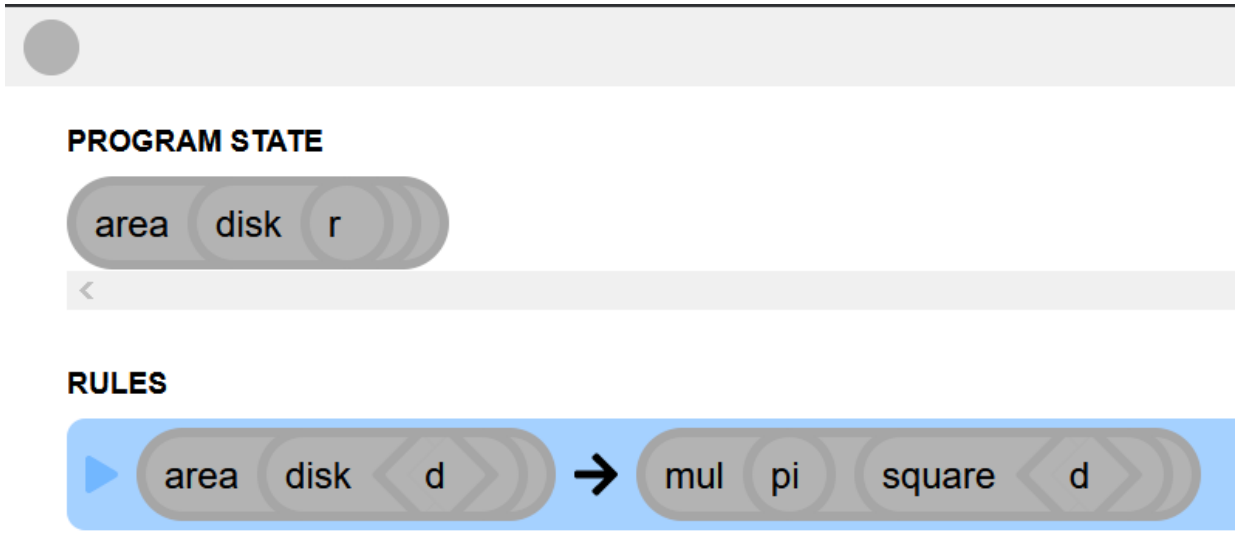
FunBlocks

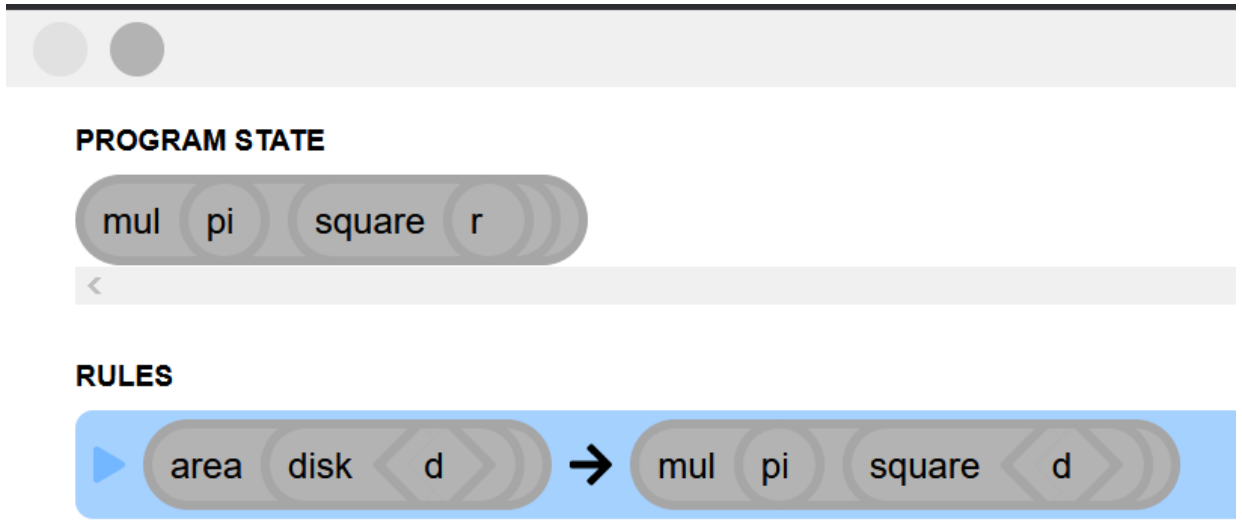
Road Map


February 2021
June 2021
State of the art
Test existing tools
Implementation
Report writing

FunBlocks

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





FunBlocks

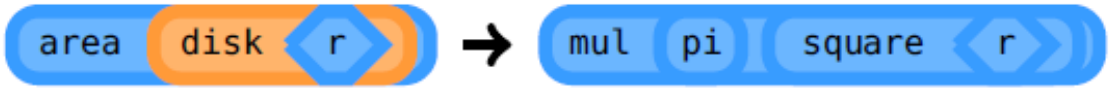
Declarative paradigm

```
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

Zero = {0}

Nat = Zero $\cup$ succ(Nat)

Empty = Λ

Stack = Empty $\cup$ push(Nat, Stack)

top : Stack $\rightarrow$ Nat

pop : Stack $\rightarrow$ Stack

alternate : Stack $\times$ Stack $\rightarrow$ Stack

Rewrite systems

Canonical rewrite system

top(push(x, y)) = x

pop(push(x, y)) = y

alternate(Λ, z) = z

alternate(push(x, y), z) = push(x, alternate(z, y))

top(push(x, y)) → x

pop(push(x, y)) → y

alternate(Λ , z) → z

alternate(push(x, y), z) → push(x, alternate(z, y))

Rewrite systems

Termination

Confluence

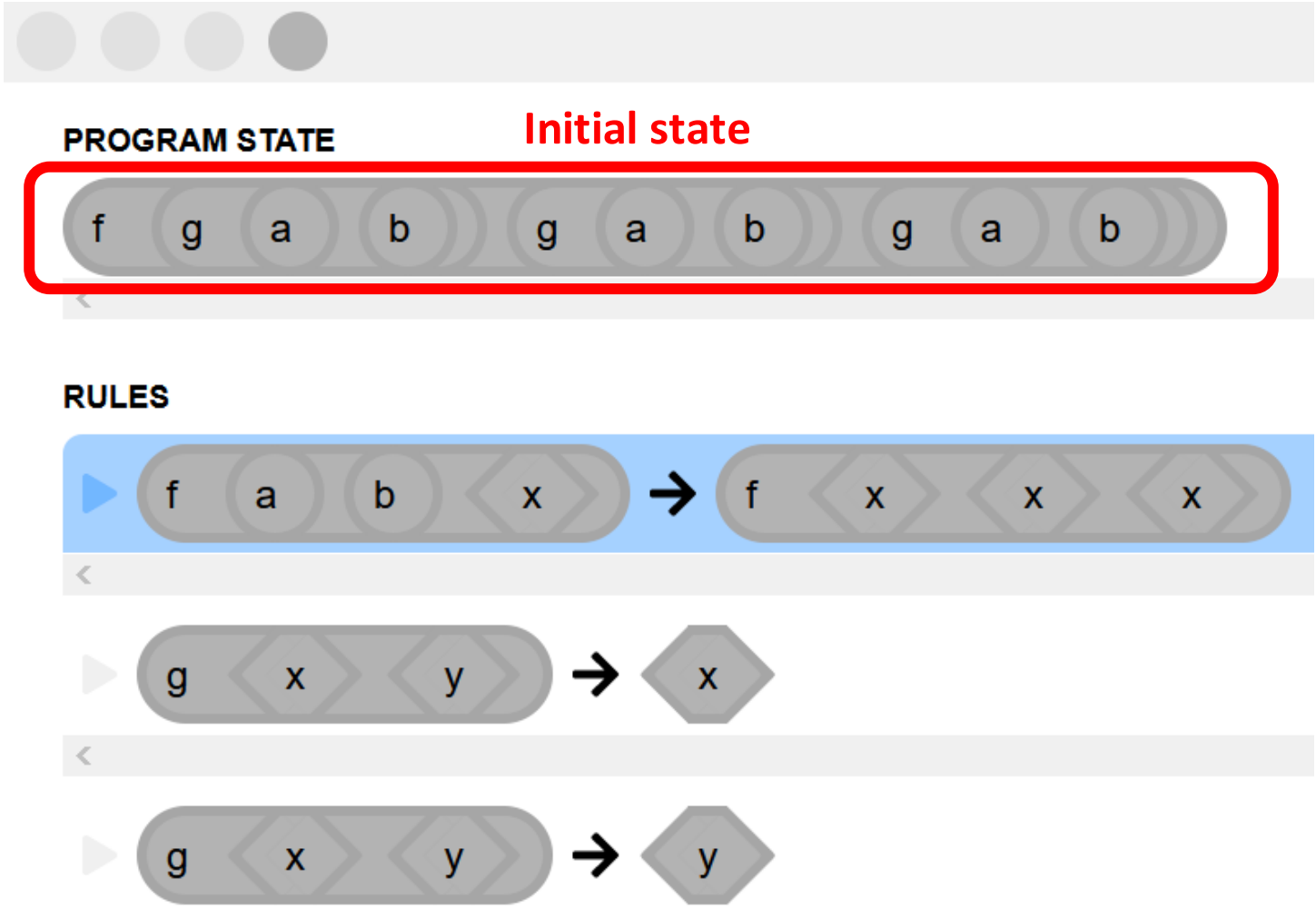
Soundness

Completeness

Correctness

Undecidable in general

Termination



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 $l \to r$)

Verification of termination

Polynomial interpretation

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

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

weight

$$w(a) = 1$$

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

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

Polynomial interpretation

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

$$w\big(f(a,x)\big) = \mathbf{2} + \boldsymbol{w(x)}$$
$$w(x) = \boldsymbol{w(x)}$$

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

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

$$w(f(g(x),y)) = \mathbf{2} + \mathbf{2}\boldsymbol{w(x)} + \boldsymbol{w(y)}$$
$$w(g(f(x,y)) = \mathbf{1} + \mathbf{2}\boldsymbol{w(x)} + \boldsymbol{w(y)}$$

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

Reduction order → 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

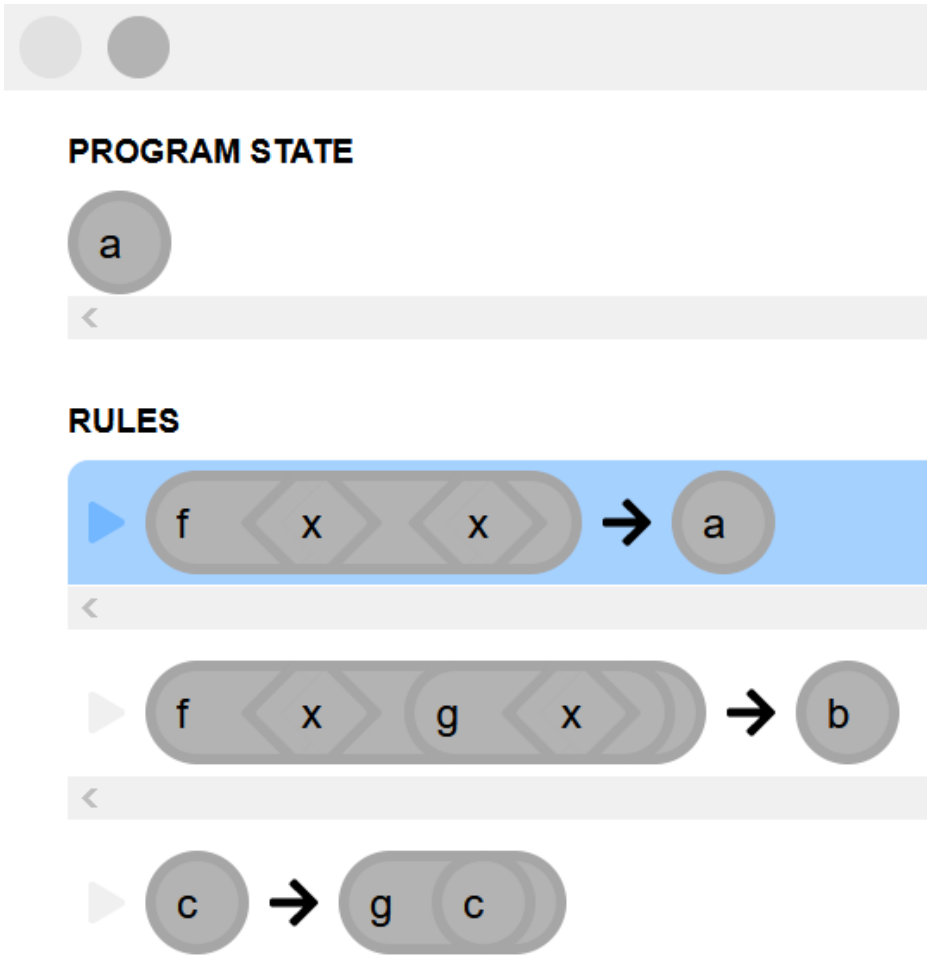
=

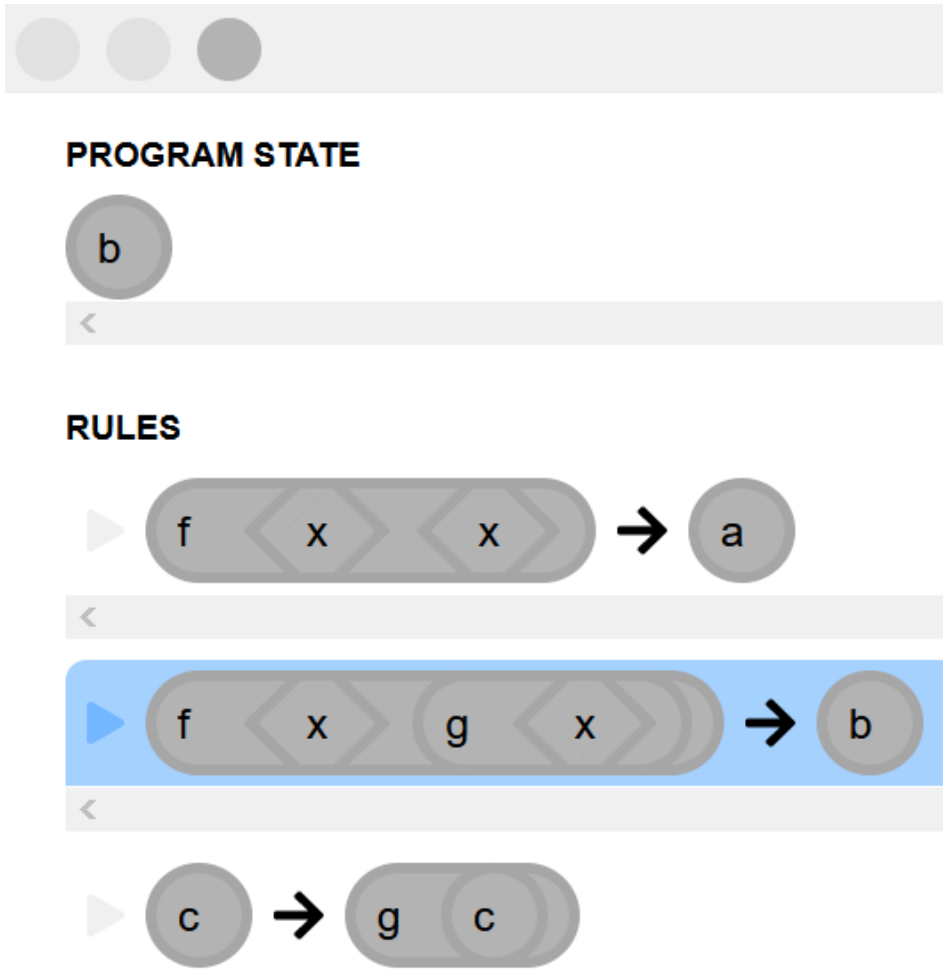
Direct proof
(polynomial, LBO, KBO,…)

+

Dependency pairs and
size-change principle

Confluence

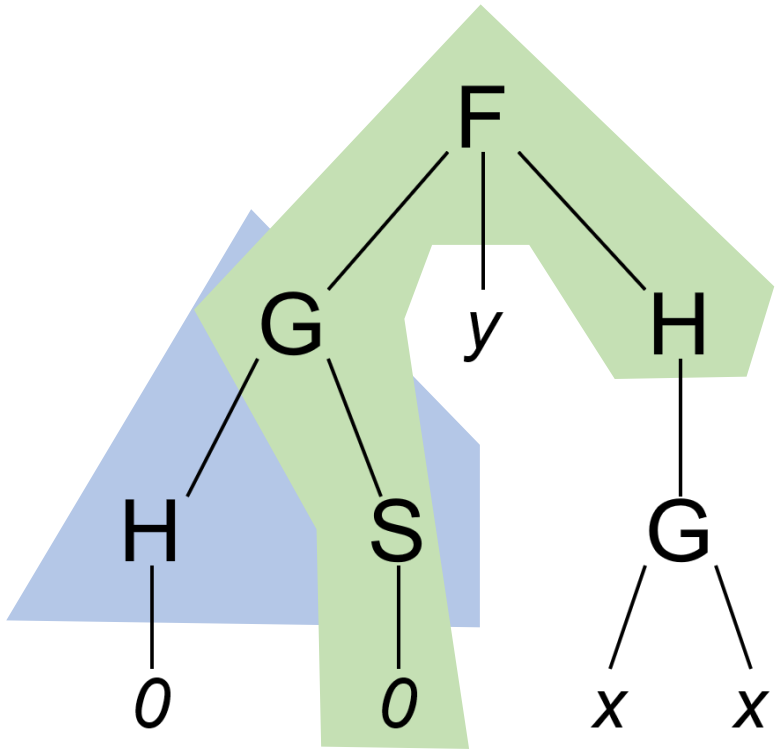

PROGRAM STATE
a
RULES
f x x → a
f x g x → b
c → g c


PROGRAM STATE
b
RULES
f x x → a
f x g x → b
c → g c

Overlap and critical pairs

$\rho_1 : F\left(G\left(x, S(0)\right), y, H(z)\right) \to x$

$\rho_2 : G(H(x), S(y)) \to \mathrm{y}$



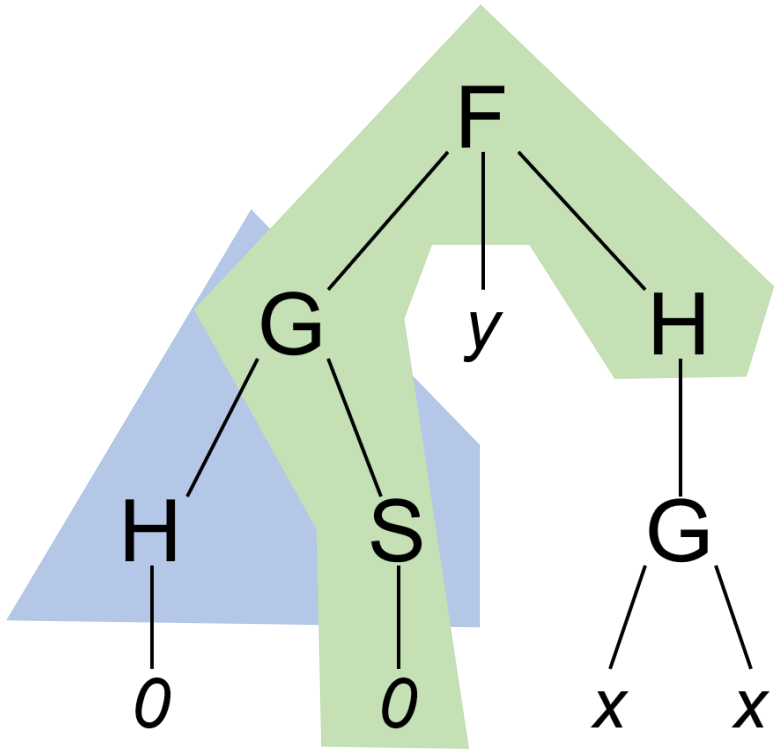
Overlap and critical pairs

Overlapping:

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

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

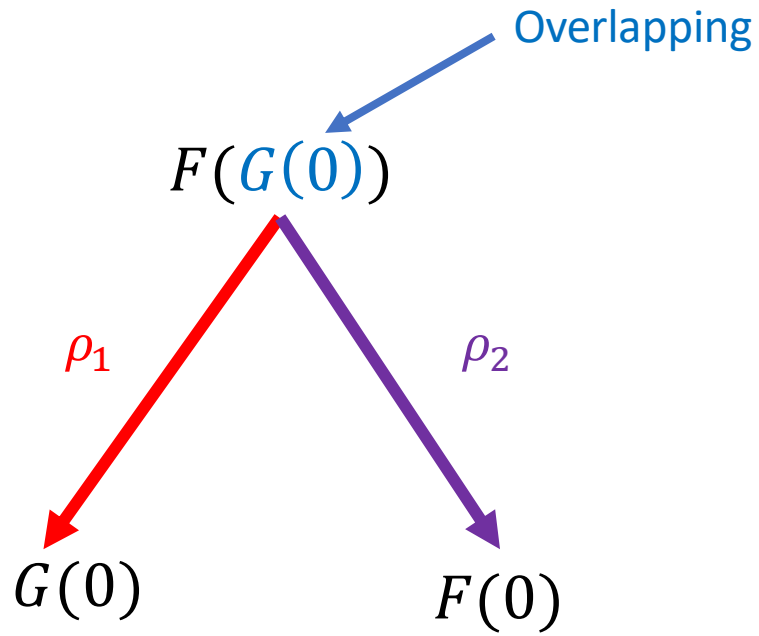
$G(H(\square), S(\square))$



Overlap and critical pairs

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

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



$< G(x), F(x) >$ is called critical pair

Critical Pair Lemma

A terminating rewriting system is confluent

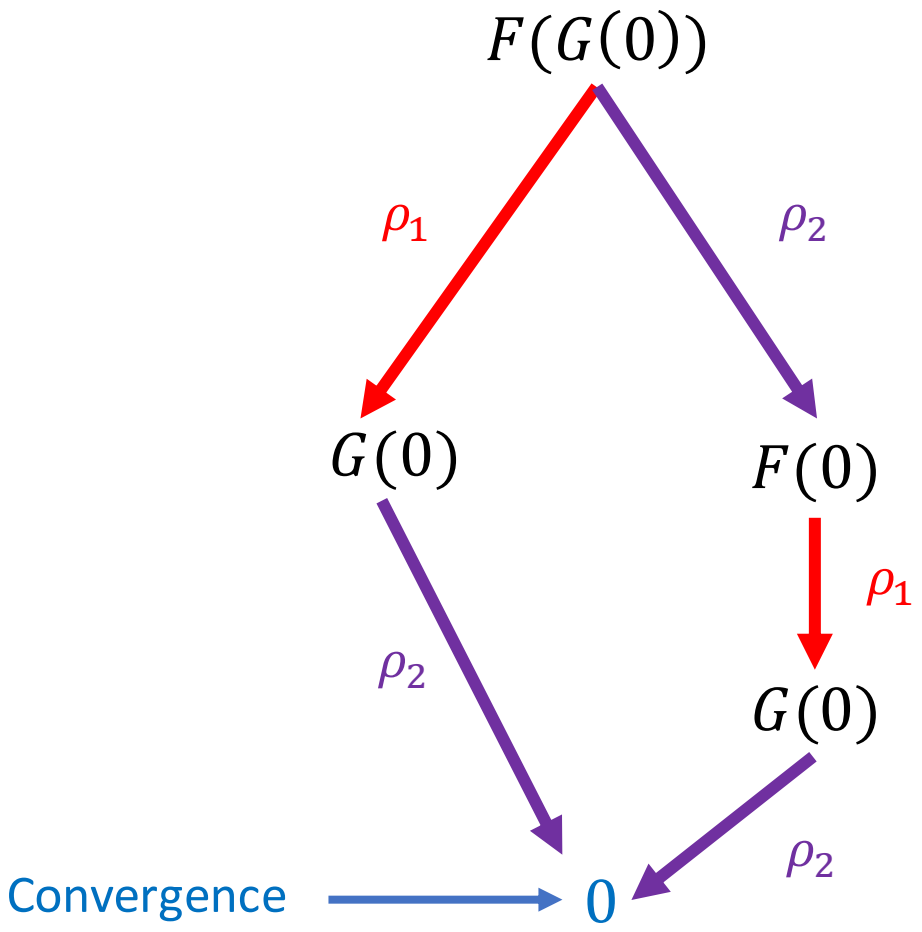
if and only if

all critical pairs are convergent

Critical Pair Lemma

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

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



Knuth-Bendix completion

Input:

A set of equation

A reduction ordering <

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

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

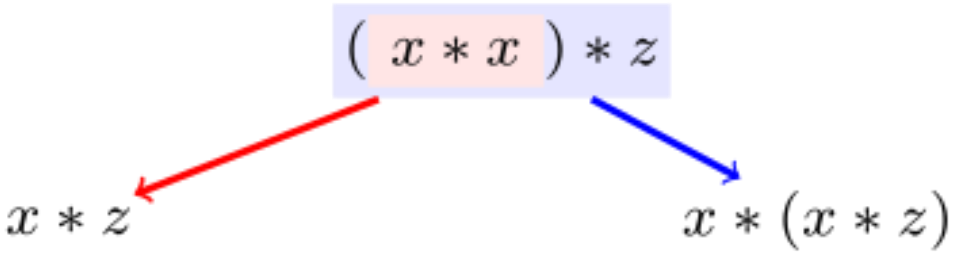
Basic rules:

Orienting	→	Transform $s = t$ to $s \to t$
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

$$(x * y) * z \rightarrow x * (y * z)$$
$$x * x \rightarrow x$$



No convergent

$$(x * y) * z \rightarrow x * (y * z)$$
$$x * x \rightarrow x$$
$$x * (x * z) \rightarrow x * z$$

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

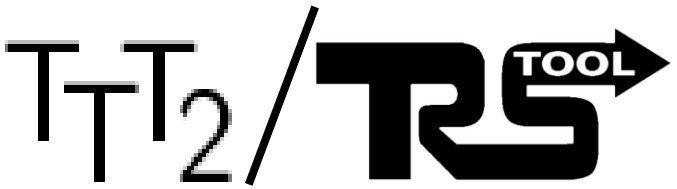
Knuth-Bendix completion

Completion for axioms of groups:

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

$$
\begin{aligned}
1 \cdot x &\to x \\
x^{-1} \cdot x &\to 1 \\
(x \cdot y) \cdot z &\to x \cdot (y \cdot z) \\
x^{-1} \cdot (x \cdot y) &\to y \\
1^{-1} &\to 1 \\
x \cdot 1 &\to x \\
(x^{-1})^{-1} &\to x \\
x \cdot x^{-1} &\to 1 \\
x \cdot (x^{-1} \cdot y) &\to y \\
(x \cdot y)^{-1} &\to y^{-1} x^{-1}
\end{aligned}
$$

Educational tools



Termination

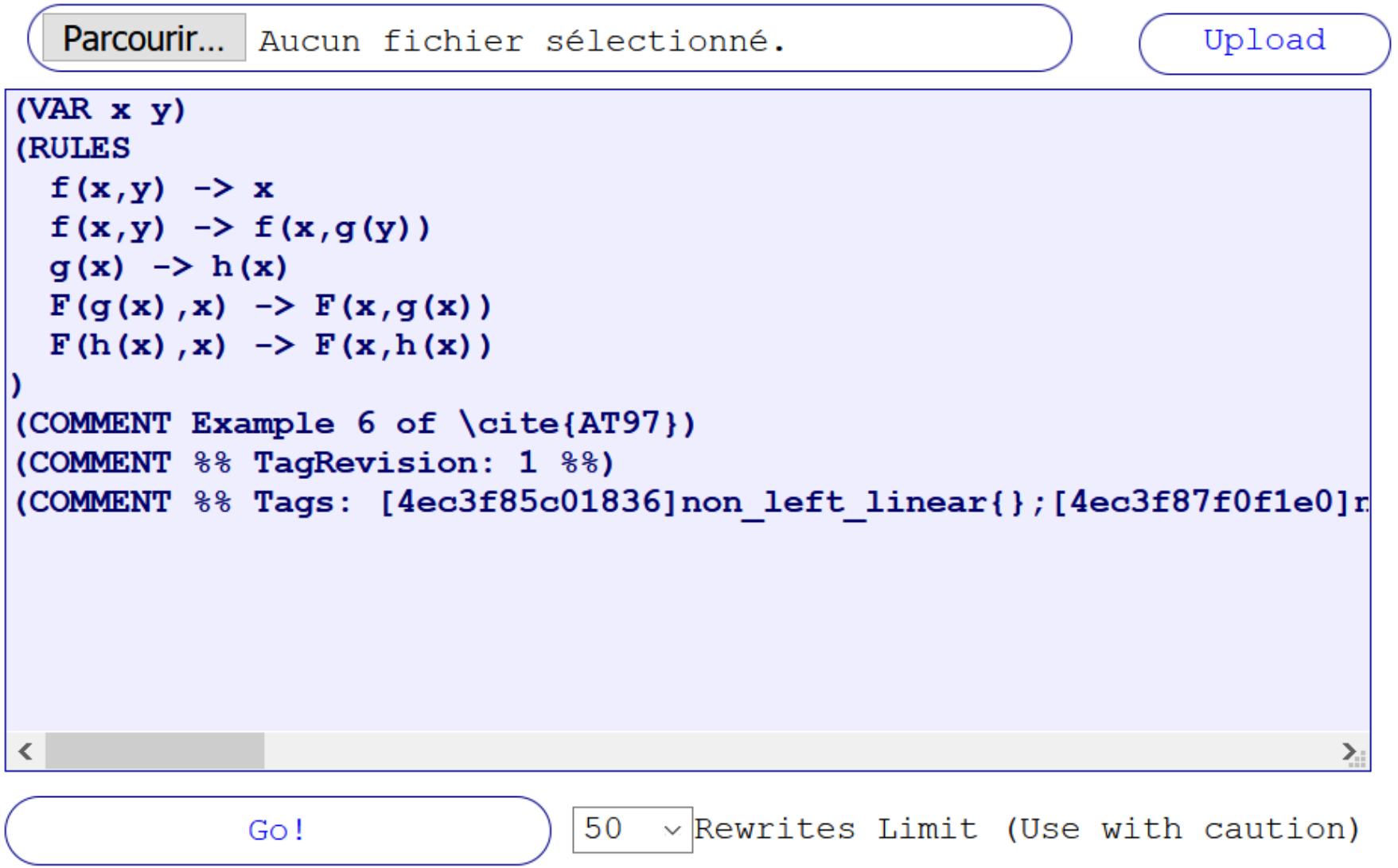
CSI

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) → 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)) → 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

```
(FUN
  f : a -> a
)
(VAR
  x : a
)
(RULES

)
```

CSI 0.1 CSI 0.6 CSI 1.1 CSI 1.1✓

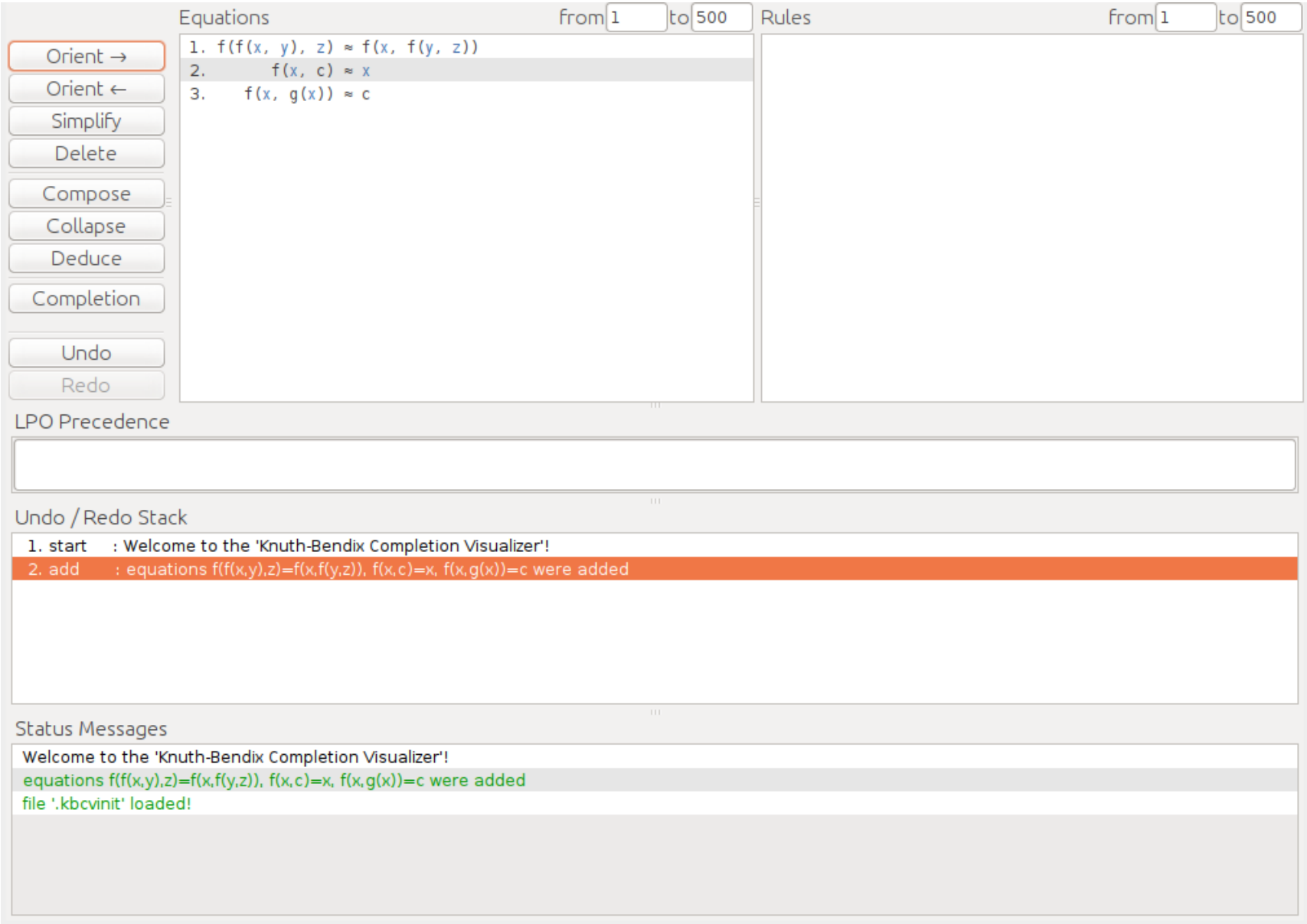
CSI 1.2.4 CSI 1.2.4✓ CSI^ho

UNR UNC CR

encode URL clear URL

reset submit

KBCV


Equations
from 1 to 500
Rules
from 1 to 500
Orient →
Orient ←
Simplify
Delete
Compose
Collapse
Deduce
Completion
Undo
Redo
1. f(f(x, y), z) ≈ f(x, f(y, z))
2. f(x, c) ≈ x
3. f(x, g(x)) ≈ c
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!

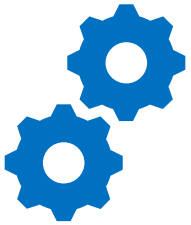
Performance tools

Termination	Confluence	Completion	Proof verification
MU-TERM	ACP	Maxcomp	CoLoR
NaTT	Saigawa		CeTA
AProVE			

Hybrid tools

CiME

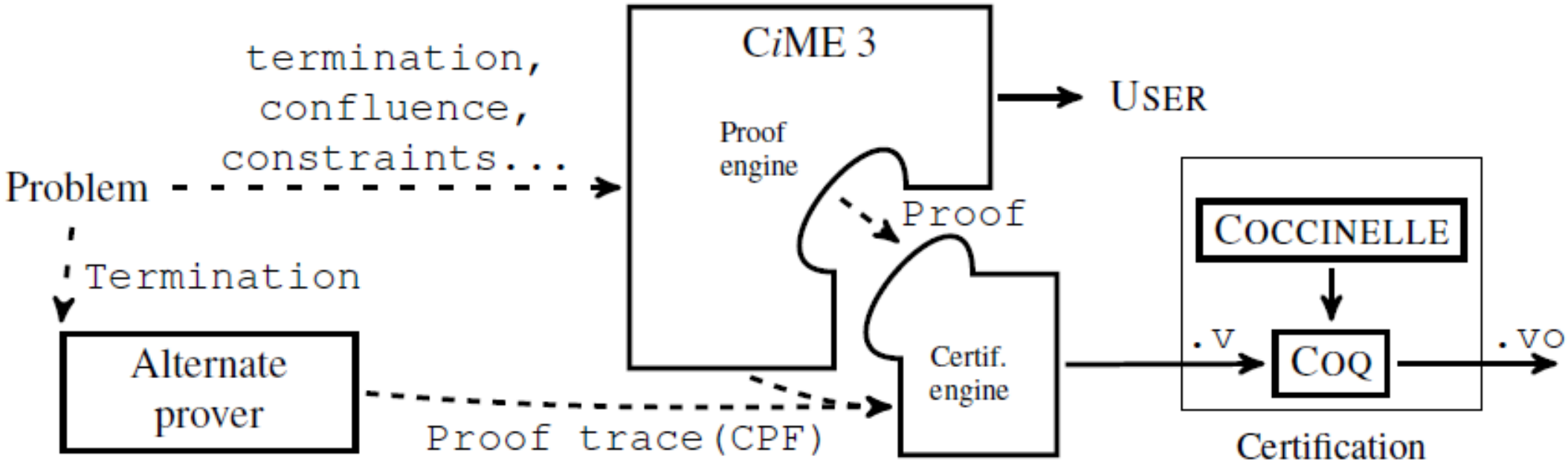
Rewriting toolkit



Proof engine

Proof certification

CiME


termination,
confluence,
constraints...
Problem
Termination
Alternate prover
CiME 3
Proof engine
Proof
Proof trace(CPF)
Certif. engine
User
Coccinelle
.v
Coq
.vo
Certification

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

Simplicity

Expressiveness

Performance

Maude

```
fmod BASIC-NAT is
        sort Nat .

        op 0 : -> Nat .
        op s : Nat -> Nat .
        op _+_ : Nat Nat -> Nat .

        vars N M : Nat .

        eq 0 + N = N .
        eq s(M) + N = s(M + N) .
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.maude
> red 100 ! .
Reduce in FACTORIAL : 100 ! .
rewrites: 201 in 0ms cpu (0ms real) (~ rewrites/second)
result NzNAT:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

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)

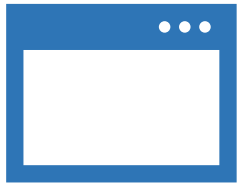
Maude Termination Tool (MTT)

} Maude Formal Environment (MFE)

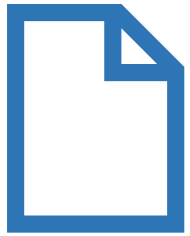
Tools overview

	Maude	CiME
Extensibility		
Still active		
I/O files		
Syntax		
Documentation		

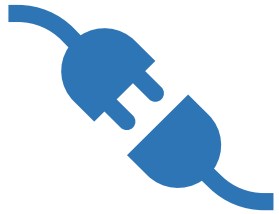
I/O in Maude



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

```
load vending-machine-grammar.maude
load buying-strats.maude
load file.maude

mod VENDING-MACHINE-IO is
              .
              .
              .
  rl < O : X | action : insert $, marking : M, Atts >
  => < O : X | action : idle,
       marking : downTerm(insertCoin('add-$, upTerm(M)), null), Atts >
     write(stdout, O, "one dollar introduced\n") .
```

I/O in Maude

Enter in external environment

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

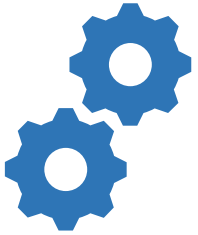
Input
Output

```
> insert $
one dollar introduced

> buy 1 a(s)
1 apples bought
```

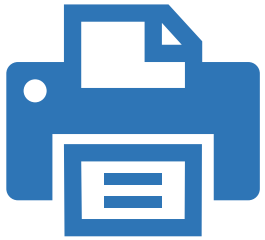
Parse input?

META-LEVEL module

Parse input

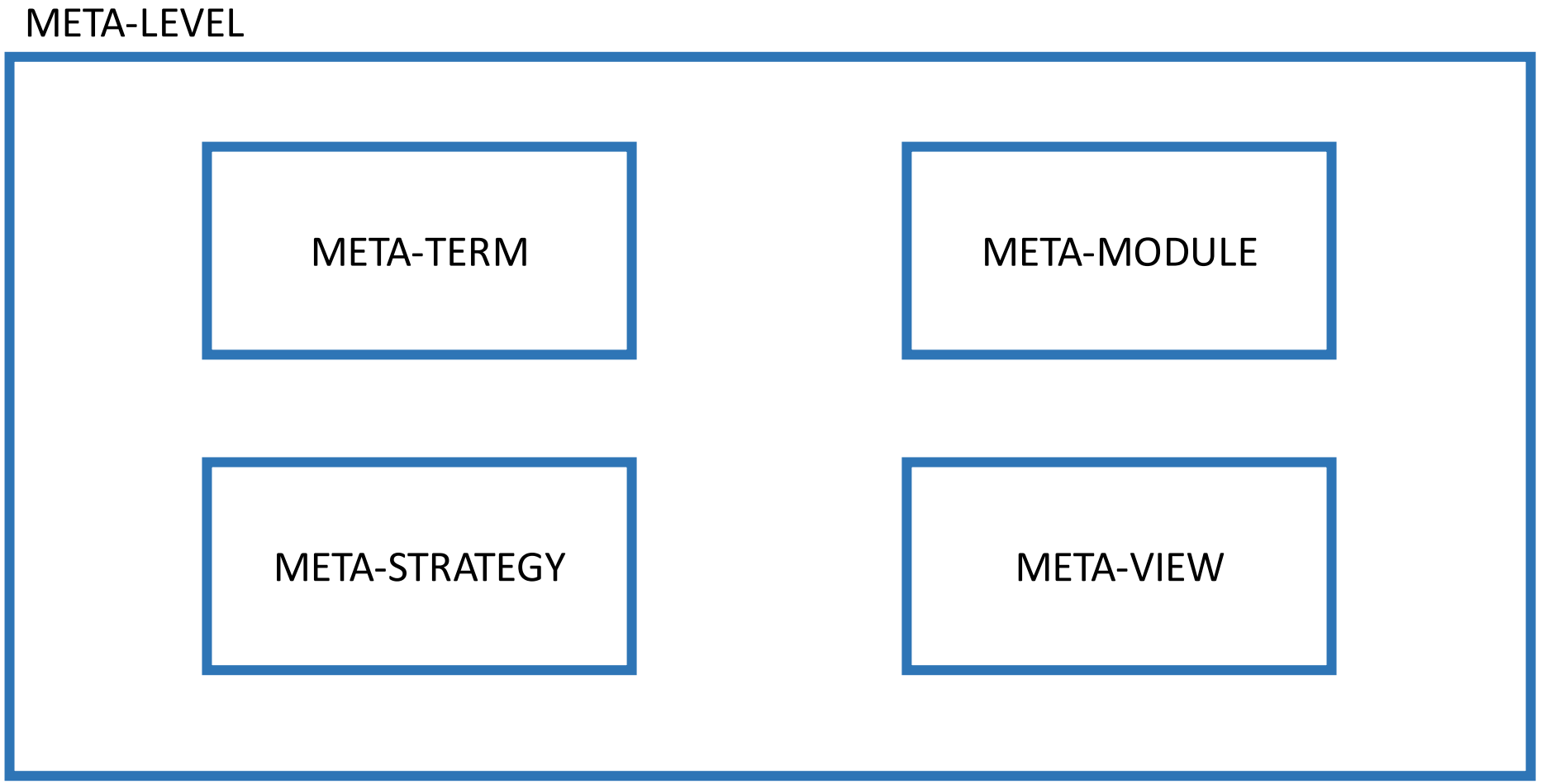


Execute



Print

META-LEVEL module



META-LEVEL operations

`metaParse()`	→	Parse a metarepresentation
`upModule()/downModule()`	→	Move through metarepresentations
`Reduce in META-LEVEL`	→	Use operation of the META-LEVEL module

Metalanguage

Funblocks

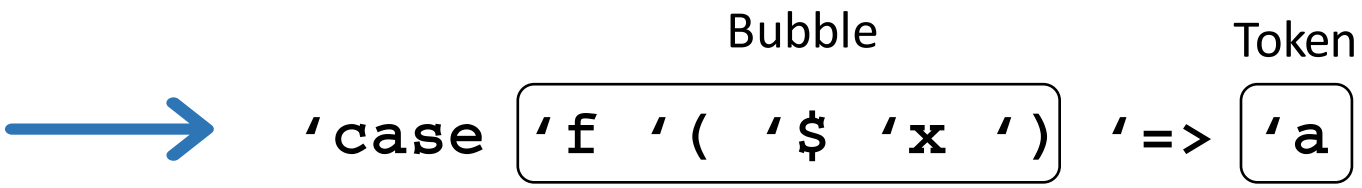
```
init area(disk(r))

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)

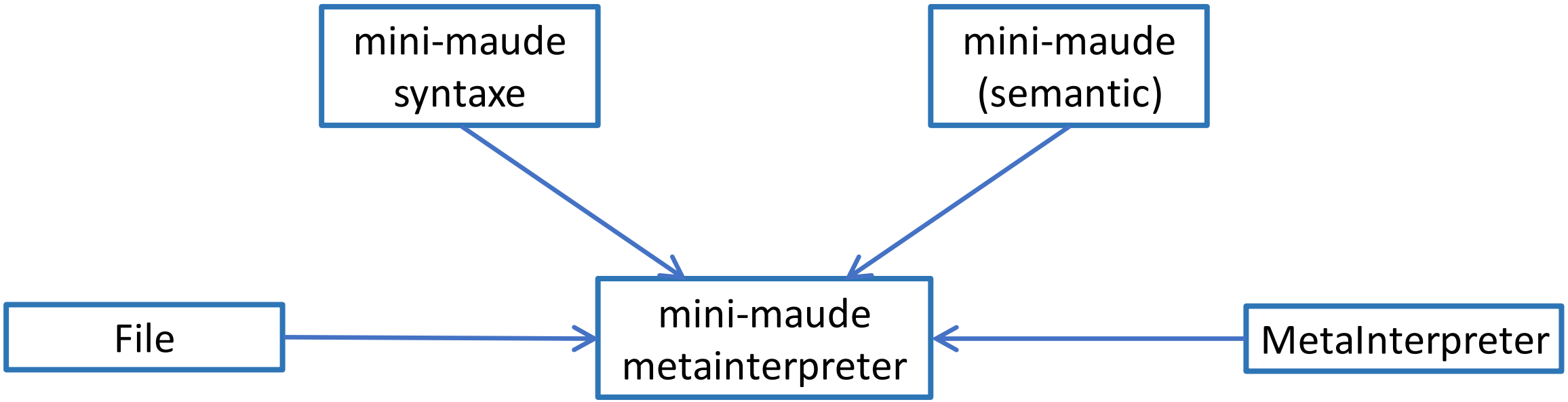
Metalanguage

```
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}
```

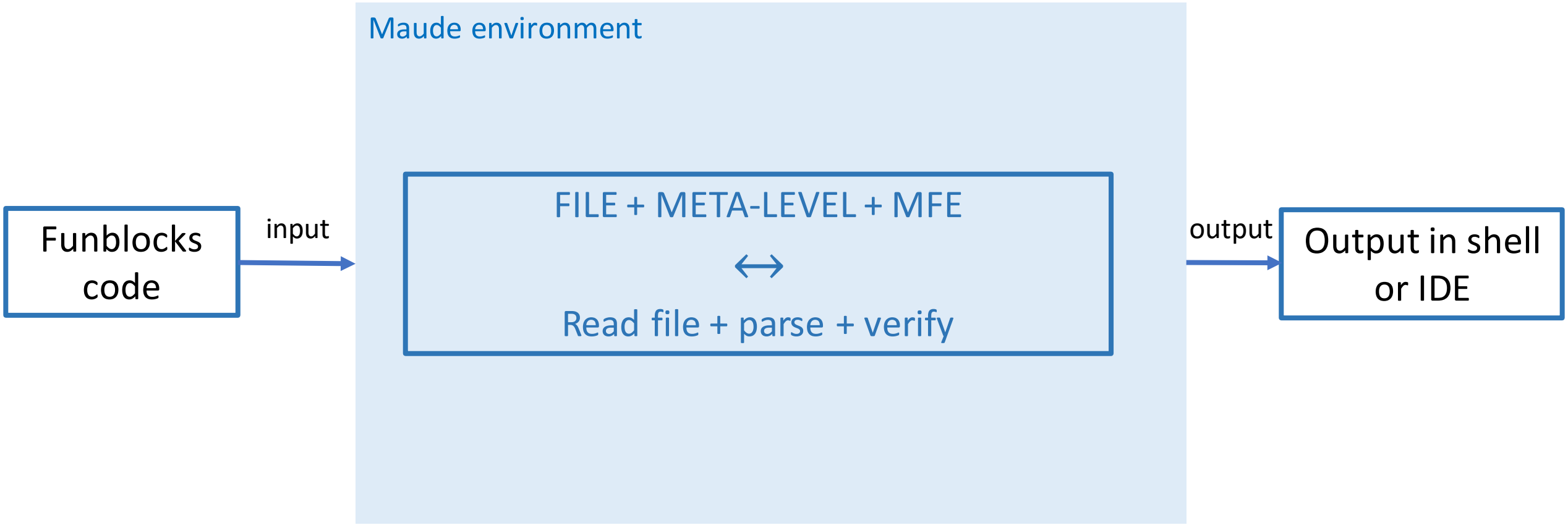
MINI-MAUDE


mini-maude
syntaxe
mini-maude
(semantic)
File
mini-maude
metainterpreter
MetaInterpreter

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
```

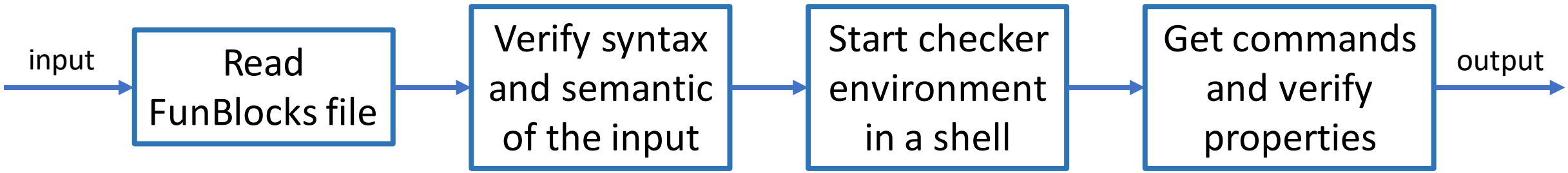
System diagram

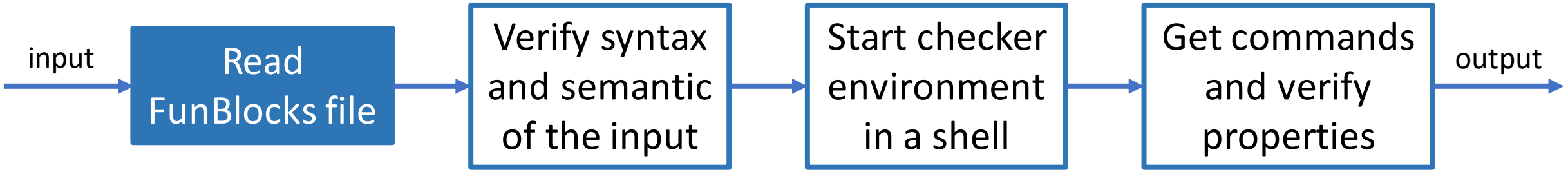


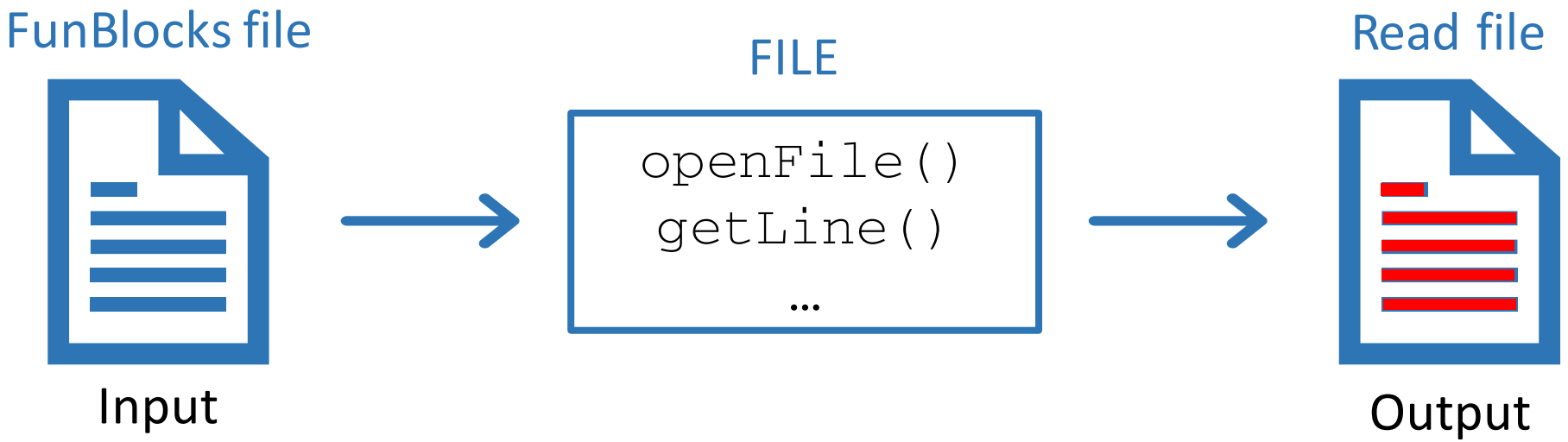
FILE + META-LEVEL + MFE

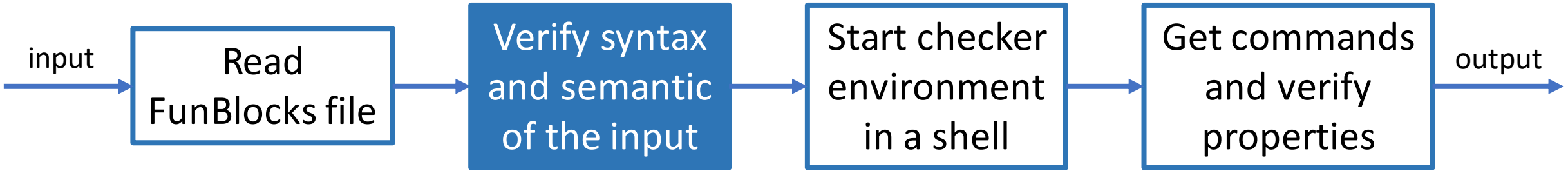
↔

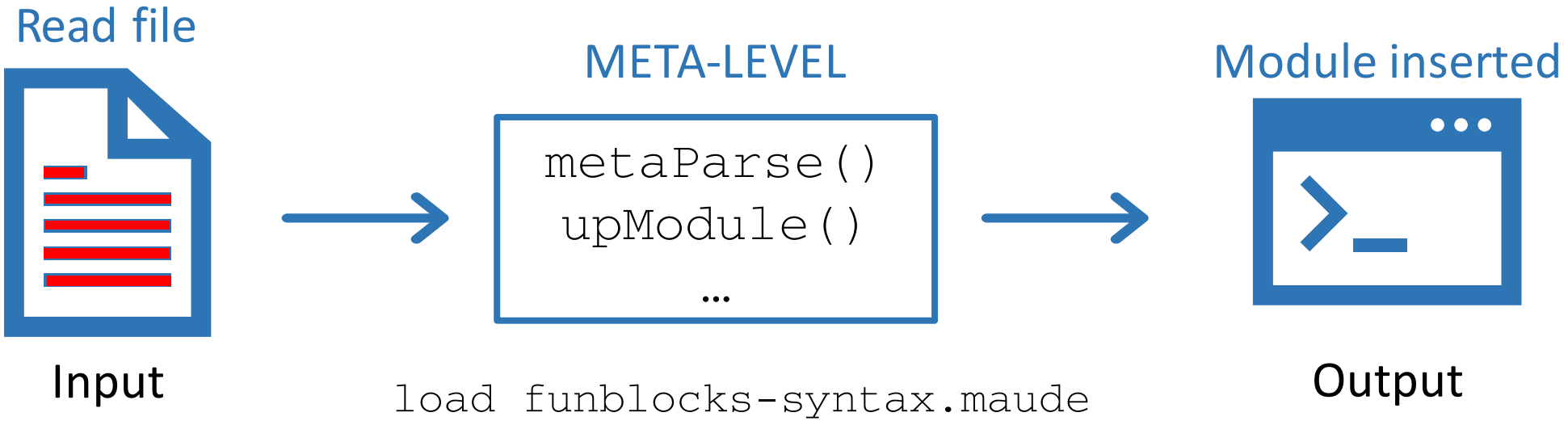
Read file + parse + verify



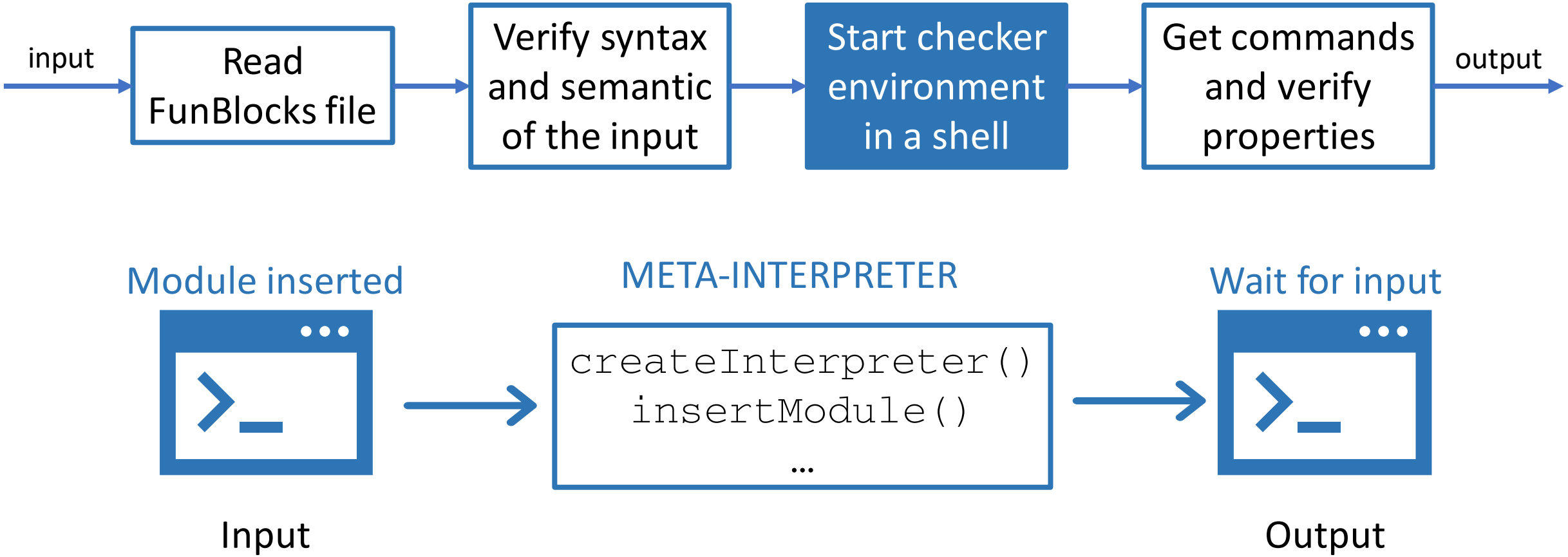

input
Read FunBlocks file
Verify syntax and semantic of the input
Start checker environment in a shell
Get commands and verify properties
output

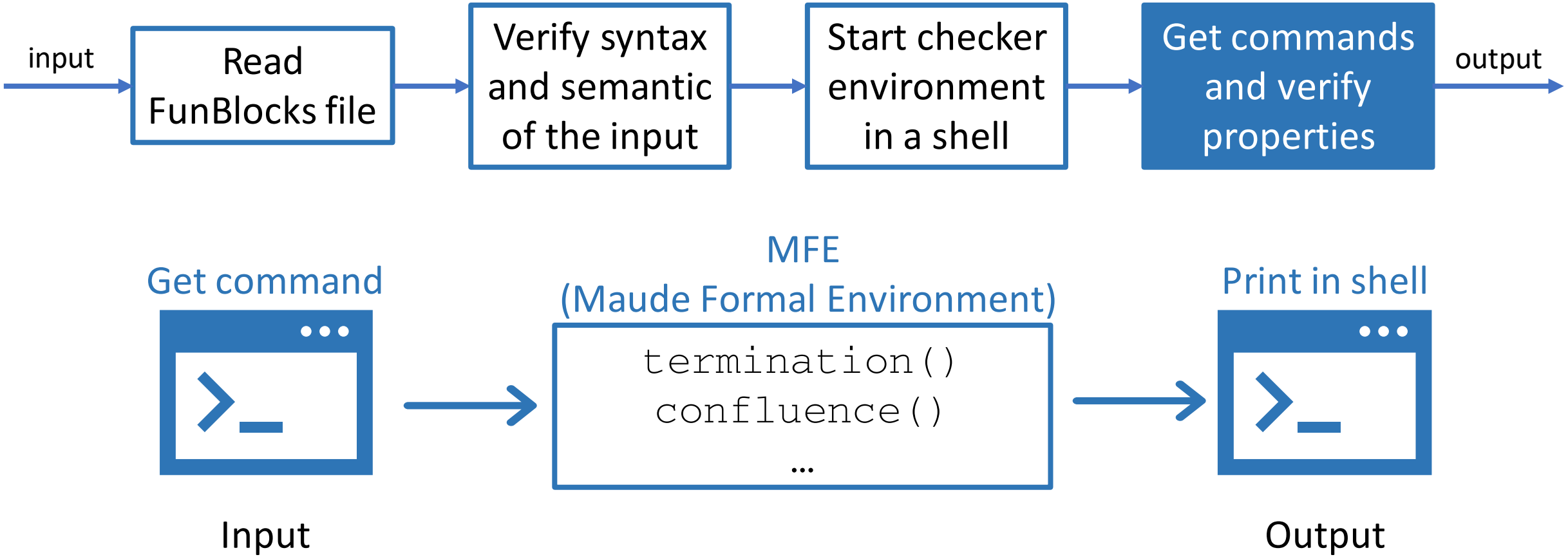

FunBlocks file
Input
FILE
openFile()
getLine()
…
Read file
Output



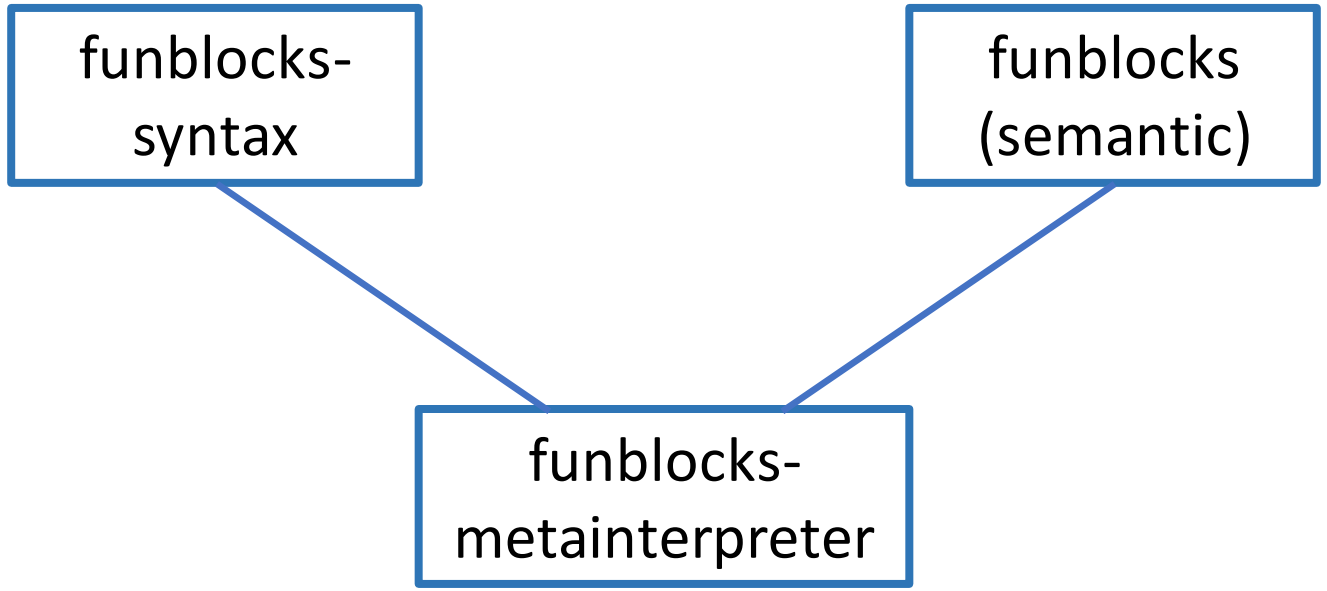



```
load funblocks-syntax.maude
load funblocks-semantic.maude
```


input
Read FunBlocks file
Verify syntax and semantic of the input
Start checker environment in a shell
Get commands and verify properties
output
Module inserted
META-INTERPRETER
Wait for input
createInterpreter()
insertModule()
…
Input
Output


input
Read FunBlocks file
Verify syntax and semantic of the input
Start checker environment in a shell
Get commands and verify properties
output
Get command
MFE
(Maude Formal Environment)
termination()
confluence()
…
Print in shell
Input
Output

FunBlocks checker system



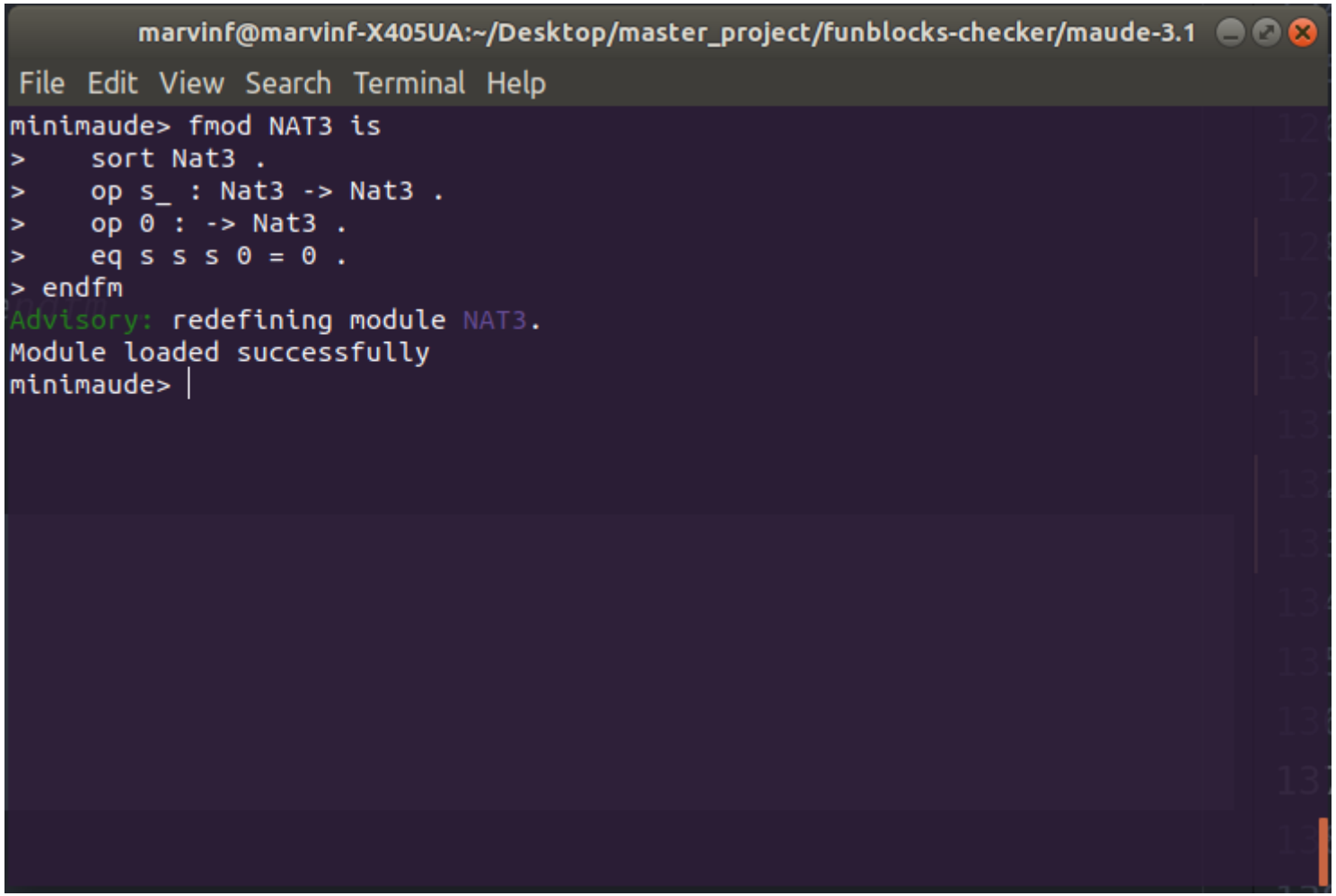
funblocks-
metainterpreter

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

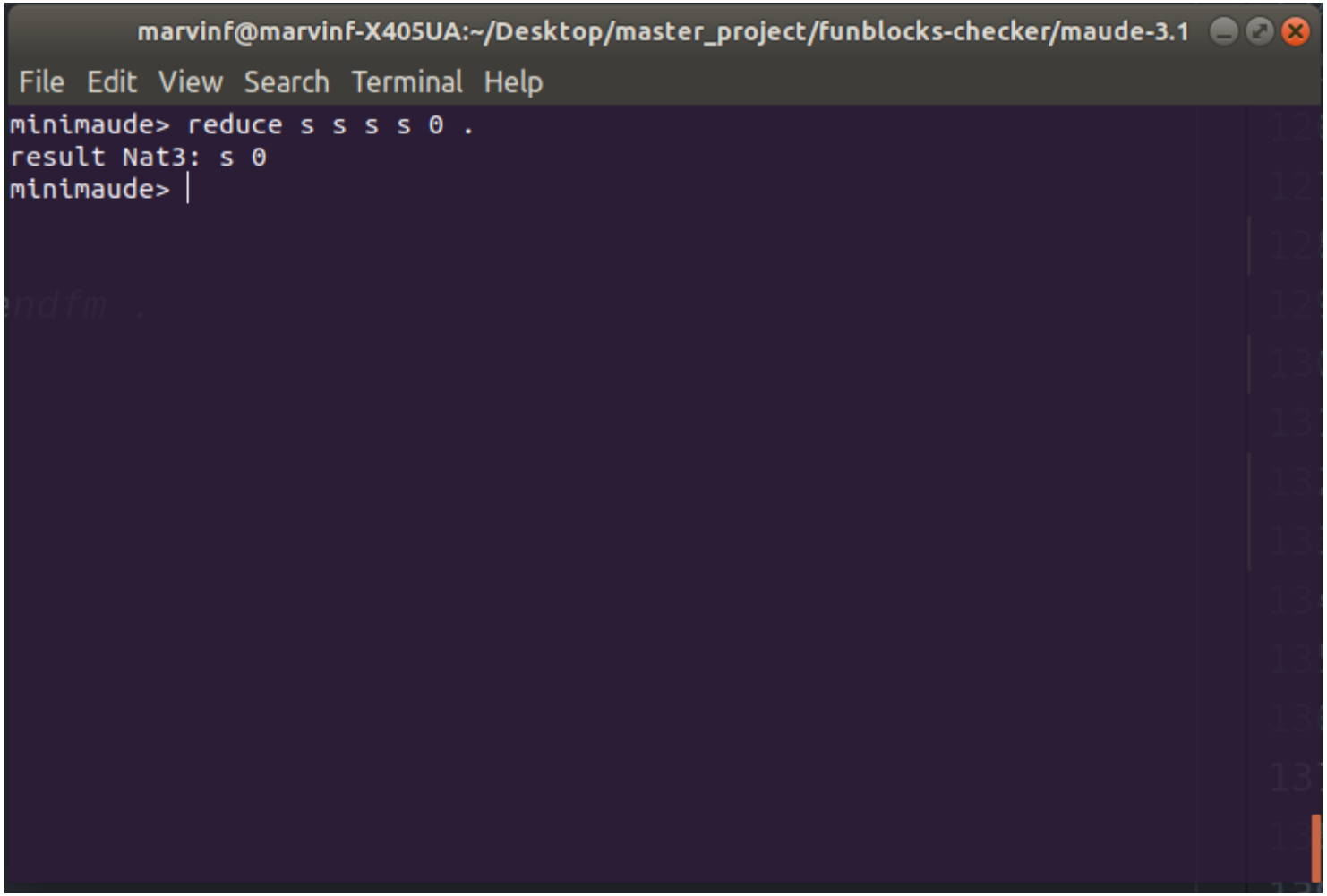
funblocks-
metainterpreter

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

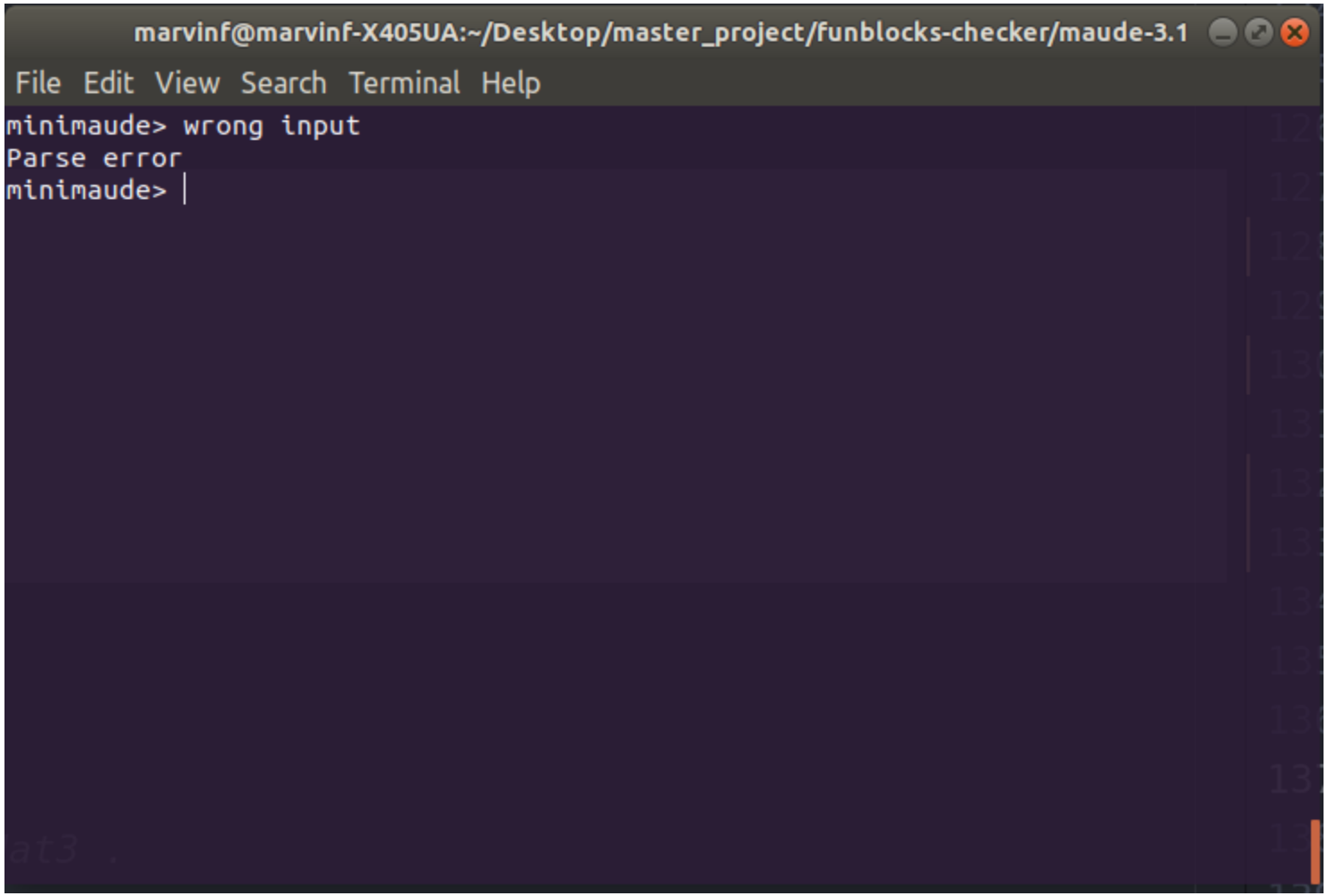
Mini-Maude: insert module



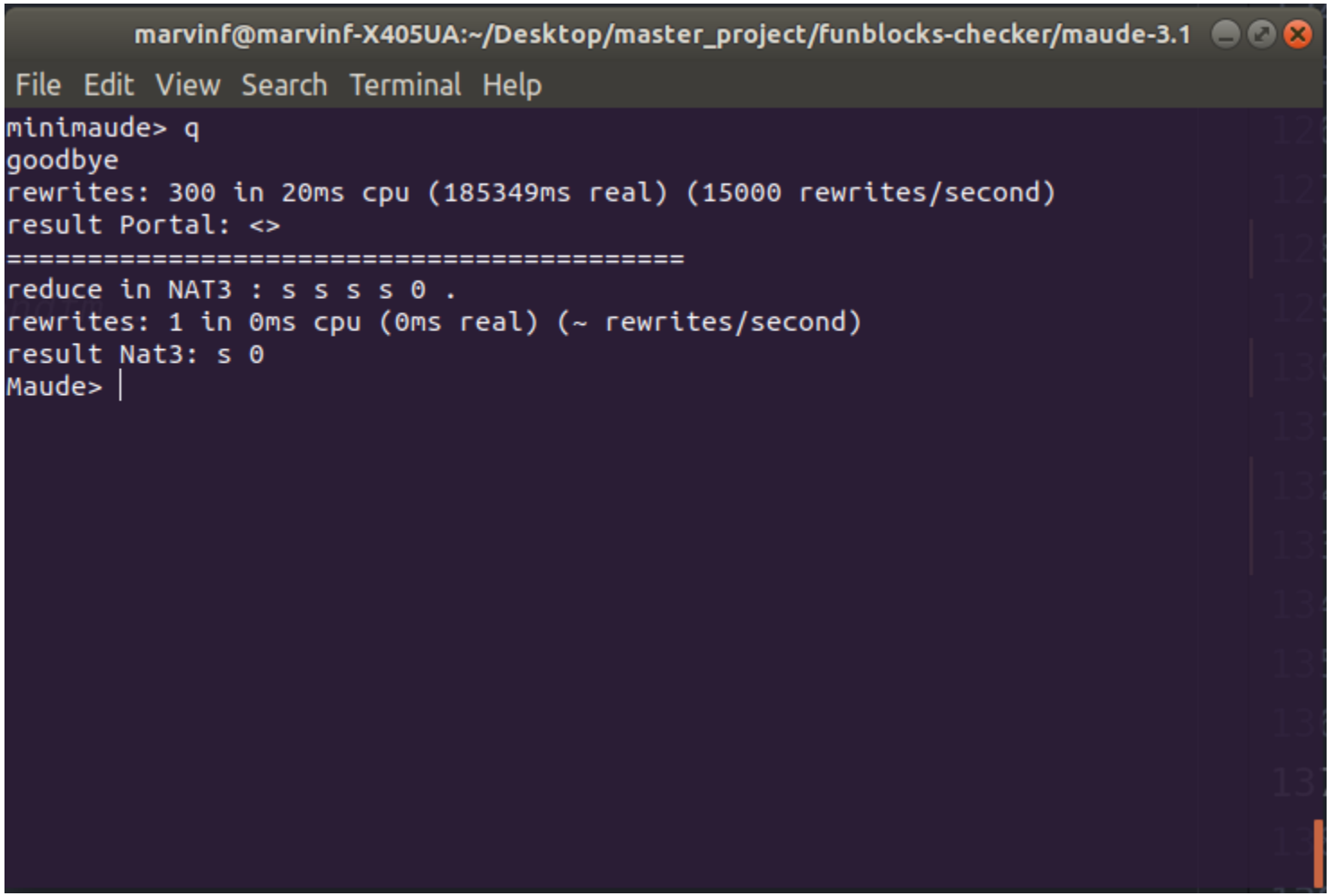
Mini-Maude: use « reduce » command



Mini-Maude: wrong command

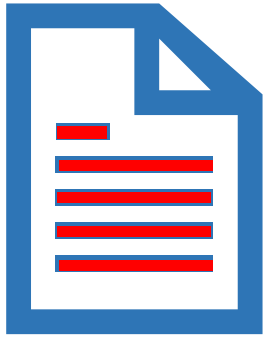


Mini-Maude: quit environment



Funblocks checker

FunBlocks code



Input

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

Module inserted

Output

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

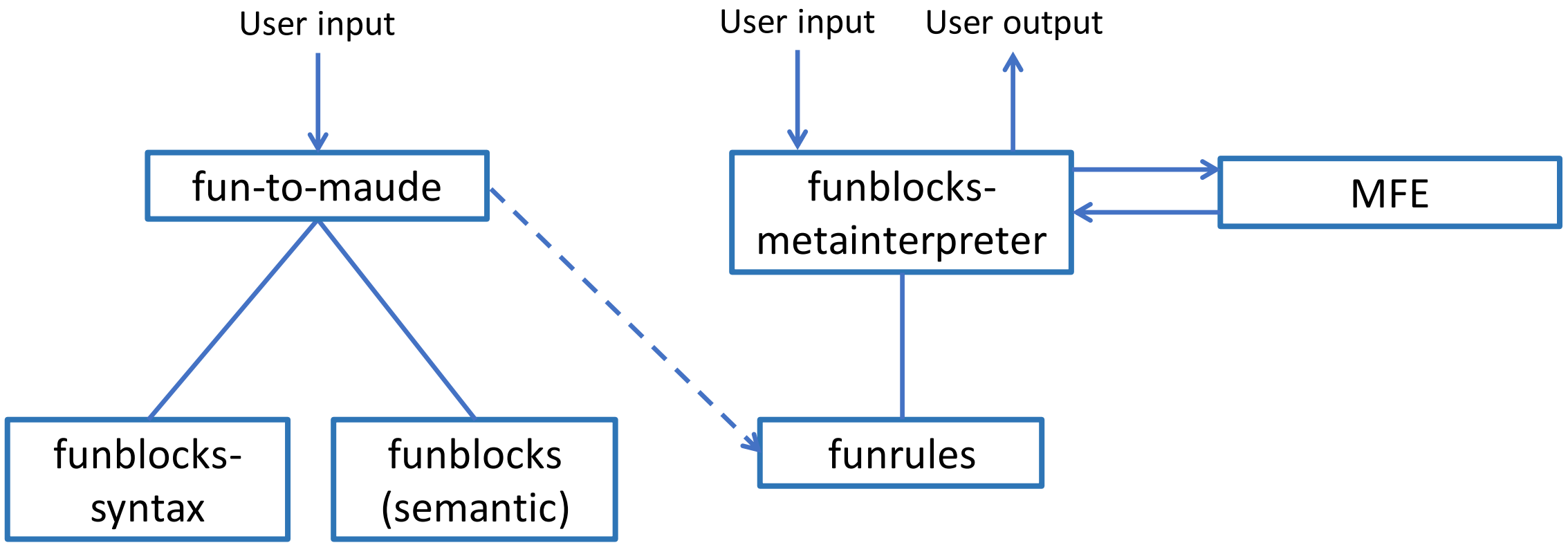
What's next ?

```
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

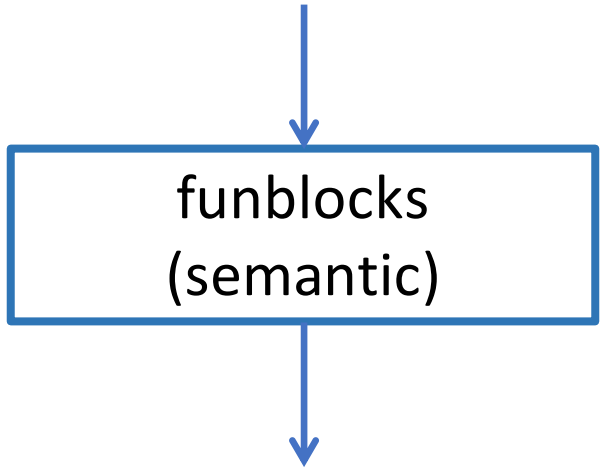


FUNBLOCK-SYNTAX

```
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

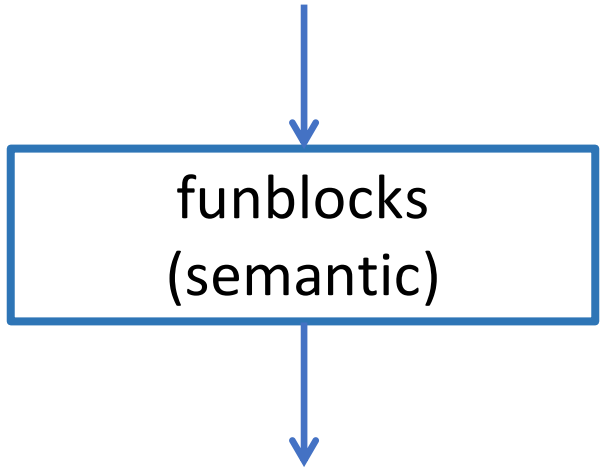
```
‘case ‘$ ‘x ‘=> ‘id ‘( ‘a ‘)
```




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

FUNBLOCKS

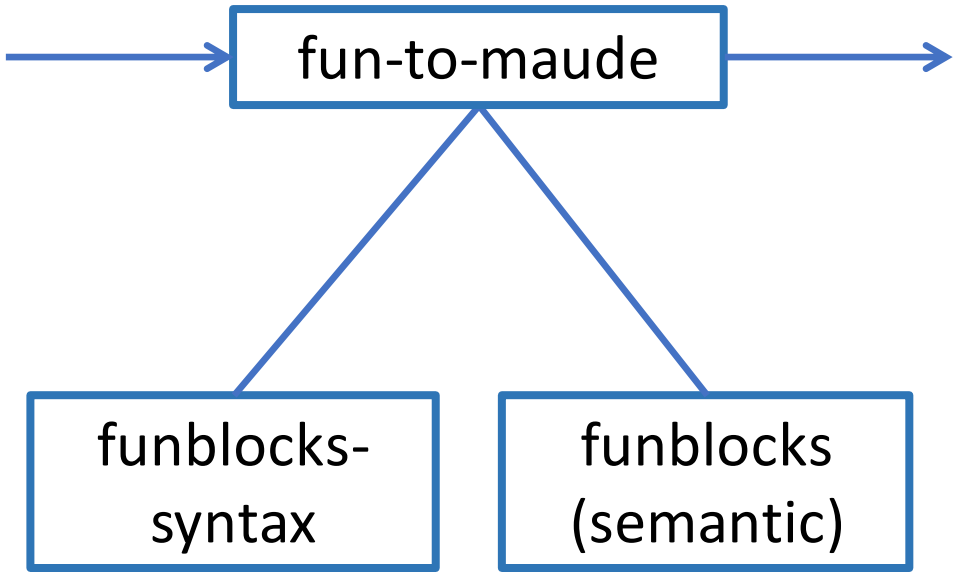
```
‘case ‘compose ‘( ‘f ‘, ‘g ‘, ‘$ ‘x ‘) ‘=>  ‘f ‘( ‘g ‘( ‘$ ‘x ‘) ‘)
```




```
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
```

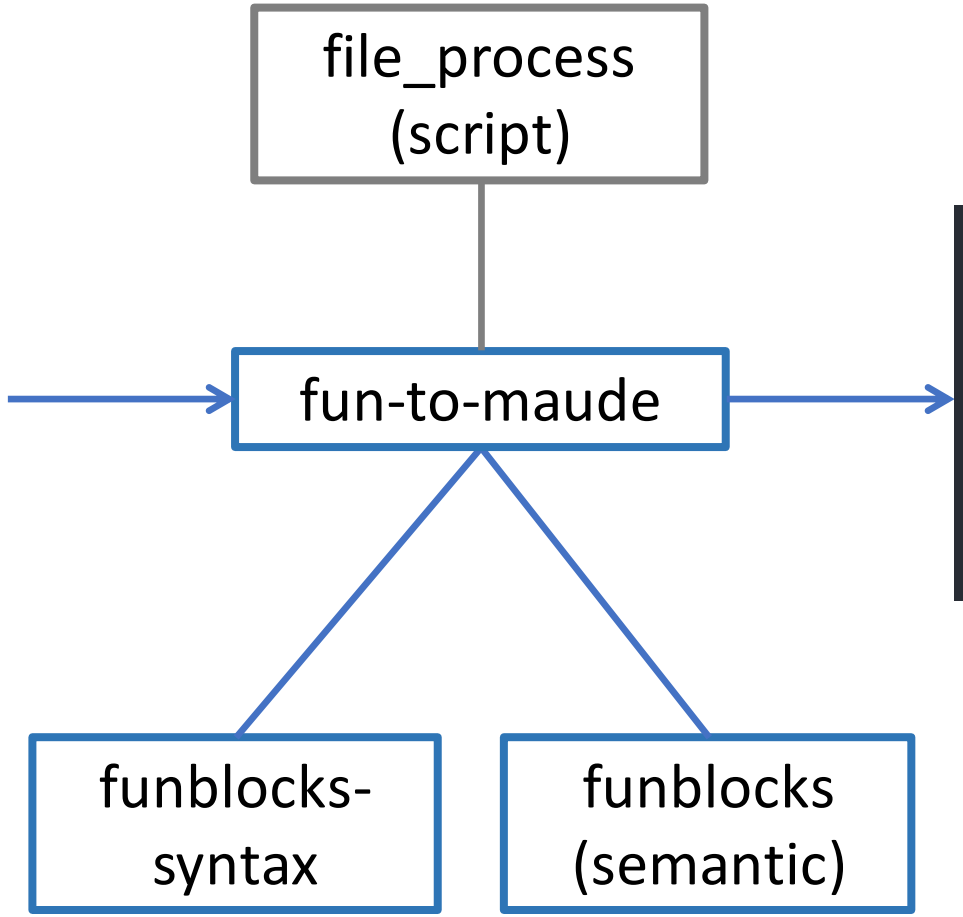
```
case id($x) => $x
```

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

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

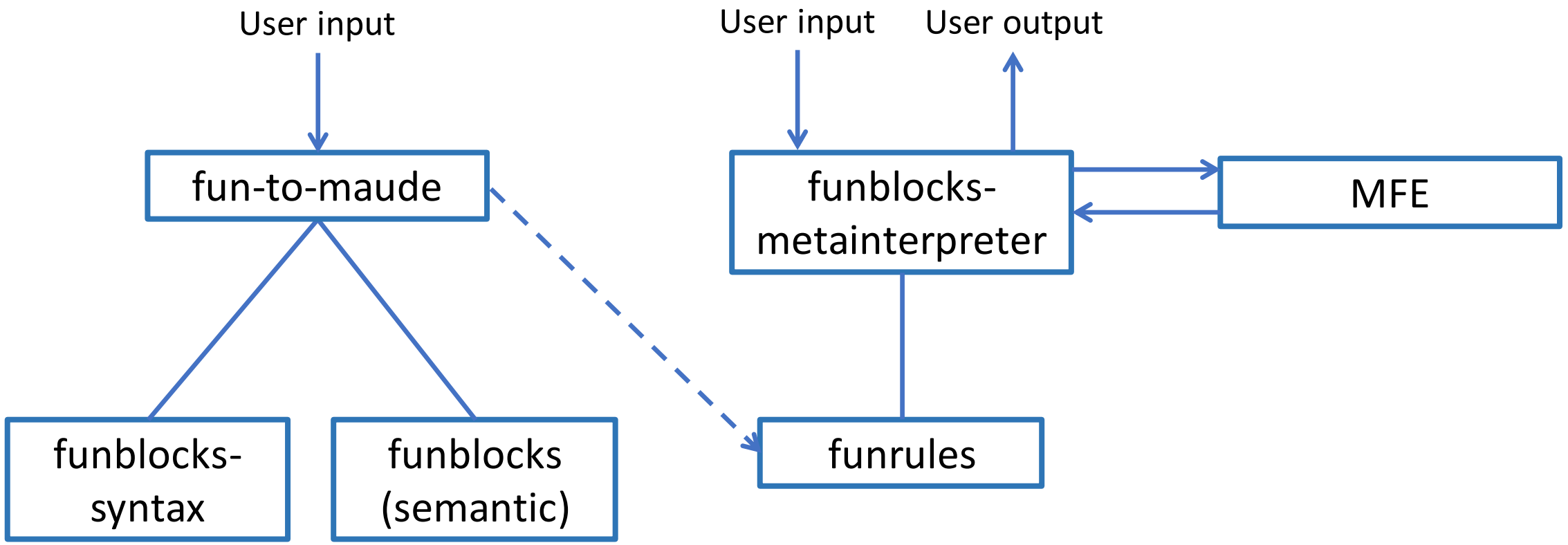
FUN-TO-MAUDE

```
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] } ] }
```

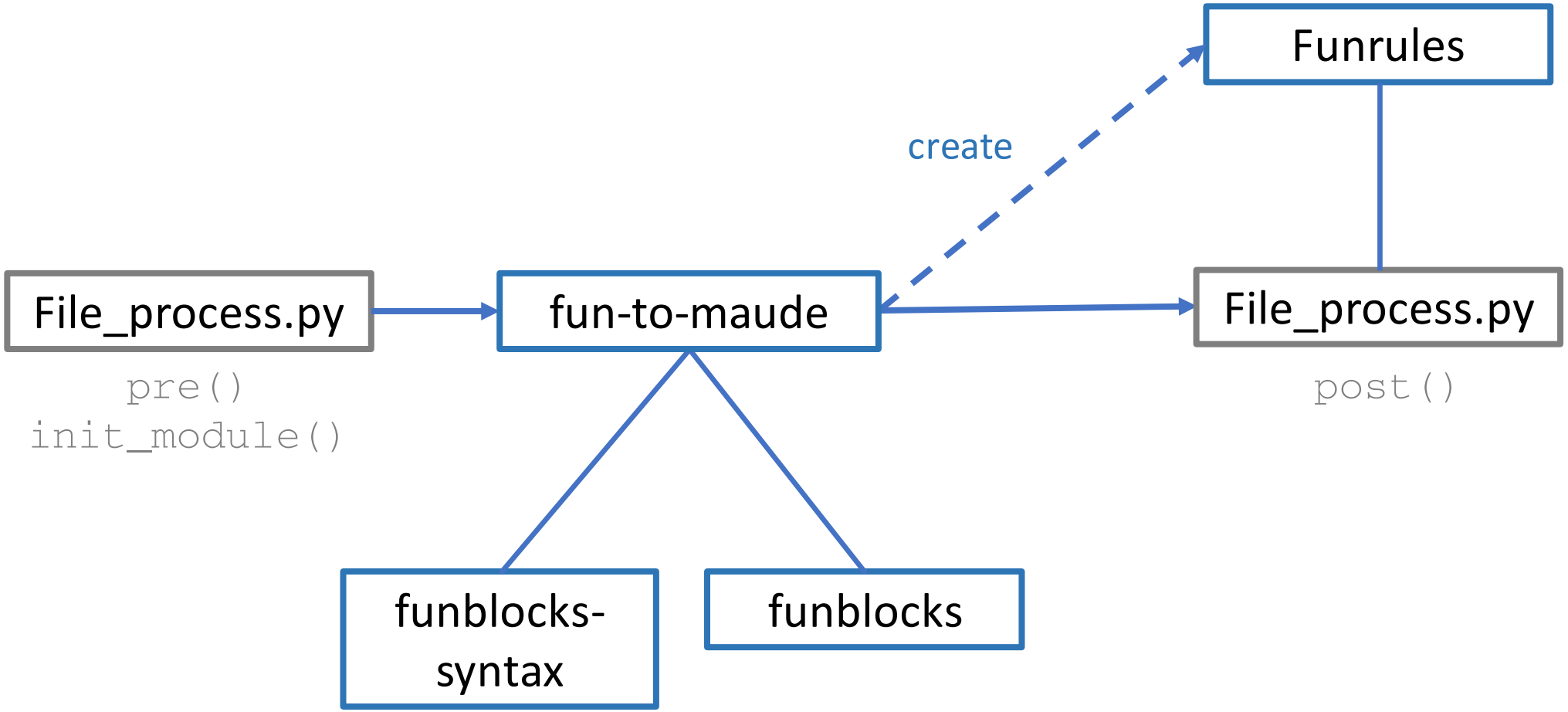
Static Typing

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

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

```
fmod MY-LIST{T :: TRIV} is

  protecting LIST{T} .
  protecting NAT .
  protecting QID .

  subsort Qid < T$Elt < List{T} .
  vars $x $y : T$Elt .
  vars $tail : List{T} .

--- constructed with 'type'
  op empty : -> List{T} .
  op ajoute : T$Elt List{T} -> List{T} .
  op cons : T$Elt List{T} -> List{T} .

--- constructed with 'case'
  eq ajoute($x, cons($y, $tail)) = cons($y, ajoute($x, $tail)) .
  eq ajoute($x, empty) = cons($x, empty) .

endfm
```

Execution


Funrules
create
File_process.py
fun-to-maude
File_process.py
pre()
init_module()
post()
funblocks-
syntax
funblocks

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)

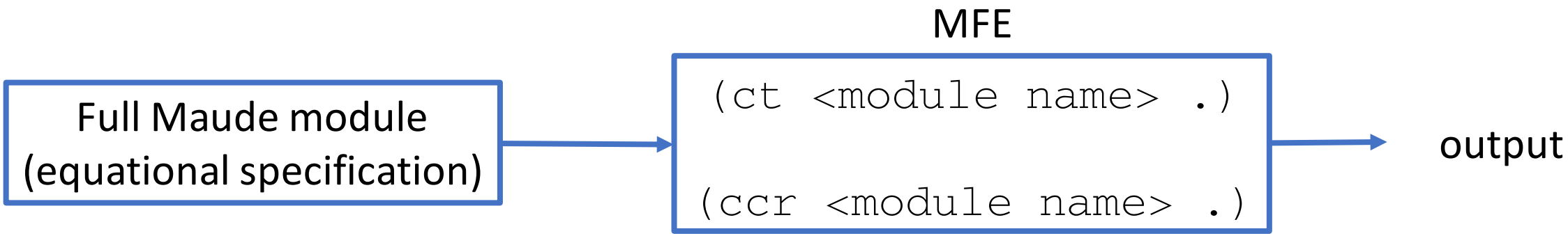
Maude++ 2.7

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

Rules

Represent theory

Terminating and confluent

Represent states changes

No constraints on termination and confluence

Reductions using rewrite logic

Parsing in Maude

reduce

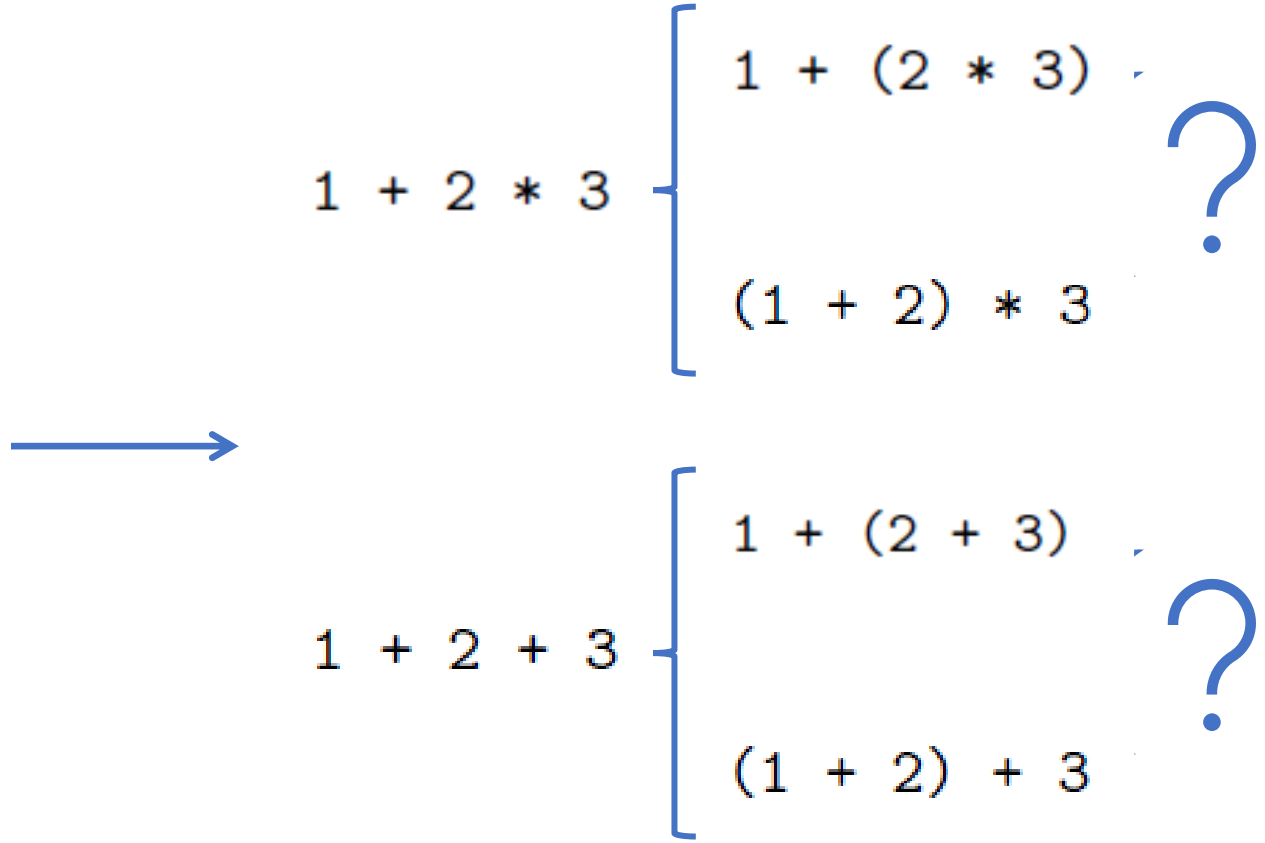
```
red 1 + (1 + 1) .
```

rewrite

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

Precedence and gathering

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



Precedence and gathering

prec ⟶ Indication of « priority » using a natural number

gather ⟶ E : precedence value ≤ 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)] .
```

```
1 + 2 * 3  ==  1 + (2 * 3)
1 + 2 + 3  ==  (1 + 2) + 3
```

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 ⟶ & 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

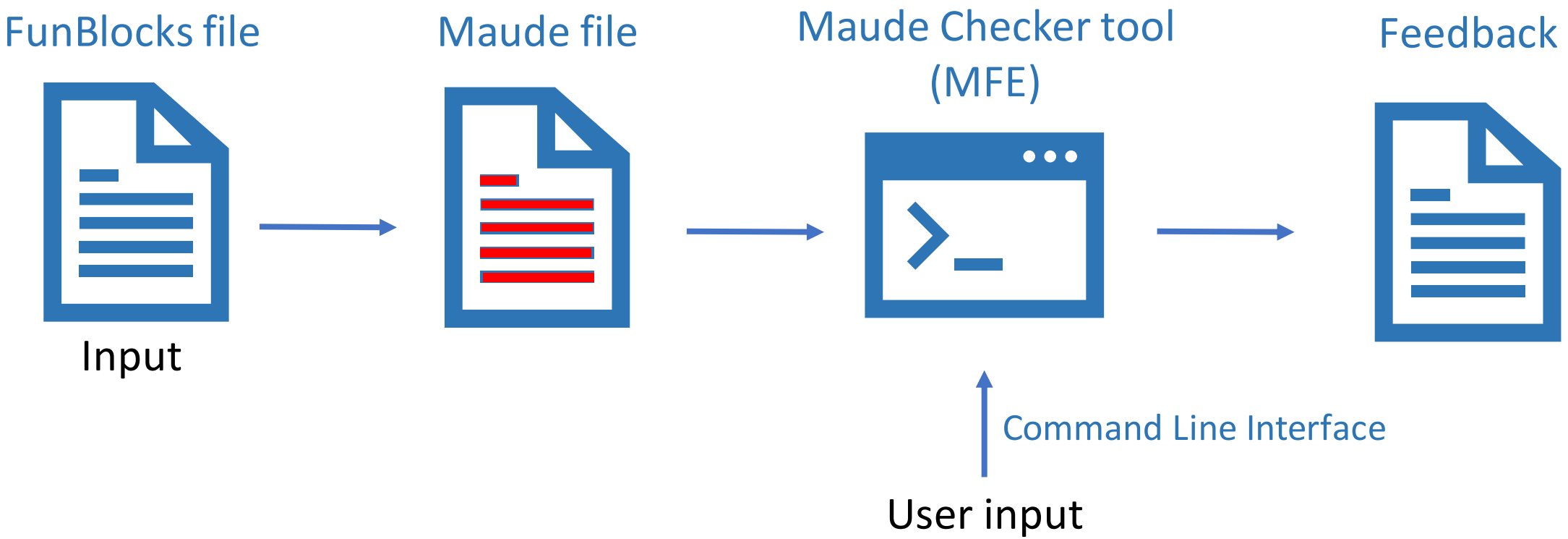
If we have `Int < IntList`

```
op _<:_ : Int IntList -> IntList .
op _:>_ : IntList Int -> IntList .
```

(e E)

(E e)

Pipeline


FunBlocks file
Maude file
Maude Checker tool (MFE)
Feedback
Input
Command Line Interface
User input

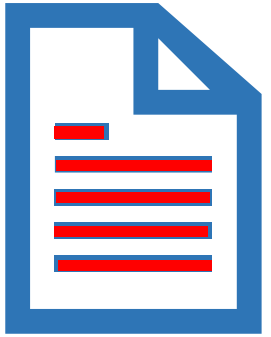
Pipeline

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 .
```

```
type Nat :: a | b | succ Nat
rule f :: Nat -> Nat -> Nat => Nat
rule g :: Nat -> Nat => Nat
case f(a,b,$x) => f($x,$x,$x)
case g($x,$y) => $x
case g($x,$y) => $y
```

```
(fmod Nat is
 sort Nat .
op a : -> Nat .
op b : -> Nat .
op succ : Nat -> Nat .
endfm)
(fmod FUNRULES is
including Nat .
op f : Nat Nat Nat -> Nat .
op g : Nat Nat  -> Nat .
var $x : Nat .
var $x : Nat .
var $x : Nat .
var $x : Nat .
eq f(a, b, $x) = f($x, $x, $x) .
var $x : Nat .
var $y : Nat .
eq g($x, $y) = $x .
var $x : Nat .
var $y : Nat .
eq g($x, $y) = $y .
 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.
```

```
type Nat :: zero | succ Nat
type Tree $T :: empty | leaf $T | node (Tree $T) (Tree $T)
rule depth $T :: Tree $T => Nat
rule add :: Nat -> Nat => Nat
case add($x, add($y, zero)) => $x
case depth(empty) => zero
case succ($z) => zero
```

```
(fmod Nat is
 sort Nat .
op zero : -> Nat .
op succ : Nat -> Nat .
endfm)
(view $T from TRIV to Nat is sort Elt to Nat .)
(fmod Tree{T :: TRIV} is
 sort Tree{T} .
op empty : -> Tree{T} .
op leaf : T$Elt -> Tree{T} .
op node : Tree{T} Tree{T} -> Tree{T} .
endfm)
(fmod FUNRULES is
including Nat .
including Tree{$T} .
op depth : Tree{$T} -> Nat .
op add : Nat Nat  -> Nat .
var $x : Nat .
var $y : Nat .
eq add($x, add($y, zero)) = $x .
eq depth(empty) = zero .
var $z : Nat .
eq succ($z) = zero .
 endfm)
```

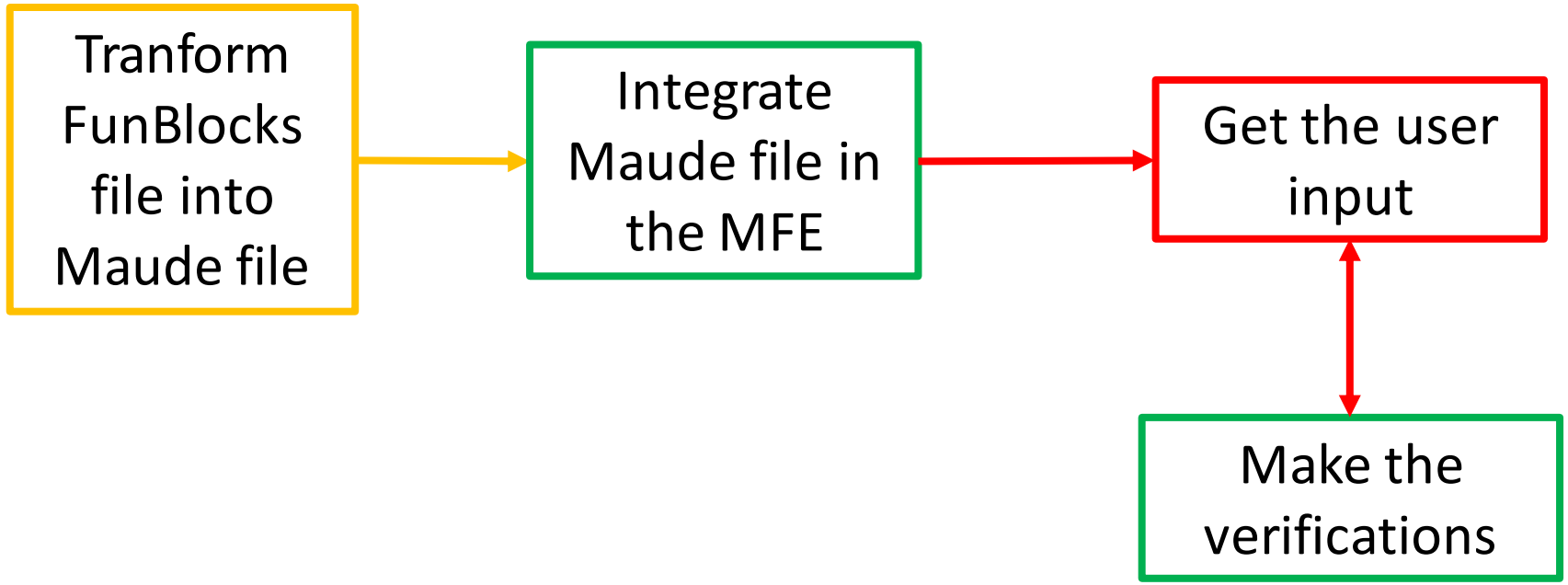
```
Success: The module FUNRULES is terminating.

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'.
```

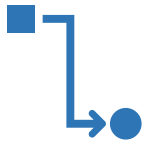
« TODO » list


Tranform FunBlocks file into Maude file
Integrate Maude file in the MFE
Get the user input
Make the verifications

```
(fmod Bool is ... endfm)
(fmod Rel is ... endfm)
(view $T from TRIV to Rel is sort Elt to Rel . endv)
(fmod List{T :: TRIV} is
  ...
  op cons : List{T} T$Elt -> List{T} .
endfm)
(fmod FUNRULES is
  including Bool .
  including Rel .
  including List{$T} .
  op size : List{$T} -> Rel .
  op isEmpty : List{$T} -> Bool .
  eq size(empty) = zero .
  var $x : Rel .
  eq size(cons(empty, $x)) = succ(zero) .
endfm)
```

```
(fmod Bool is ... endfm)
(fmod Rel is ... endfm)
(view $T1 from TRIV to Bool is sort Elt to Bool . endv)
(view $T2 from TRIV to Rel is sort Elt to Rel . endv)
(fmod List{T :: TRIV} is
  ...
  op cons : List{T} T$Elt -> List{T} .
endfm)
(fmod FUNRULES is
  including Bool .
  including Rel .
  including List{$T1} .
  including List{$T2} .
  op size : List{$T2} -> Rel .
  op isEmpty : List{$T2} -> Bool .
  eq size(empty) = zero .
  var $x : Rel .
  eq size(cons(empty, $x)) = succ(zero) .
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