Make FunBlocks alive

Marvin FOURASTIE

Master project

Motivations

Educational purpose

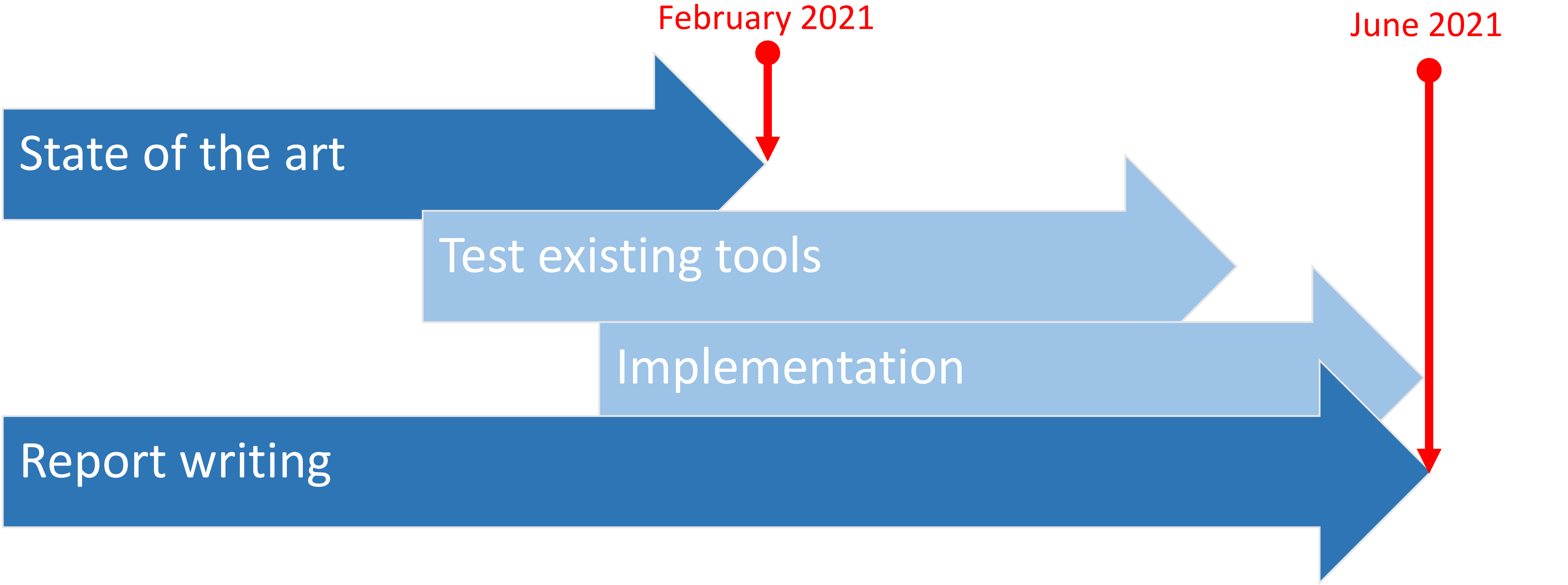
Imperative paradigm alternative

_ → _

Based on rewrite systems

FunBlocks

Road Map



February 2021

June 2021

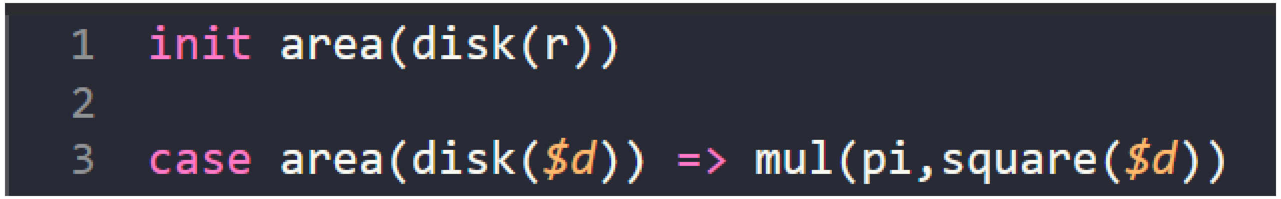
State of the art

Test existing tools

Implementation

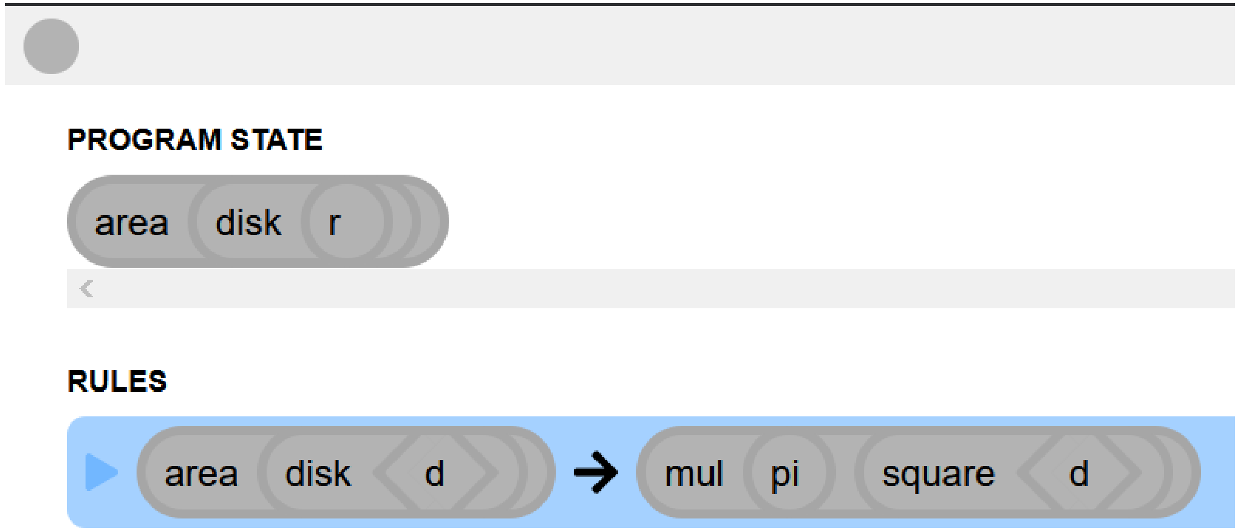
Report writing

FunBlocks



```
init area(disk(r))

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

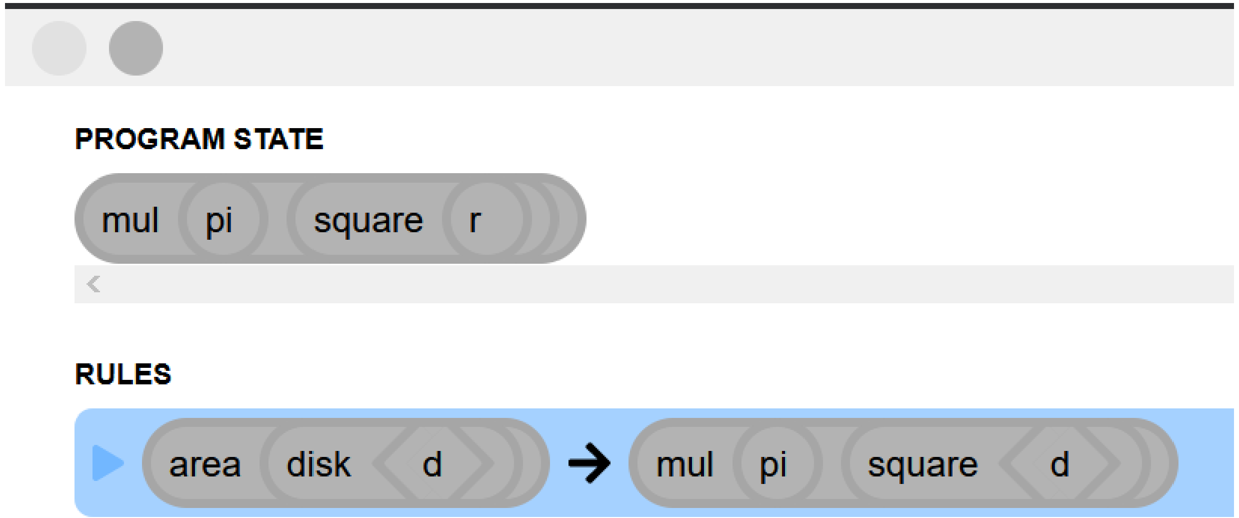


PROGRAM STATE

area disk r

RULES

area disk d → mul pi square d



PROGRAM STATE

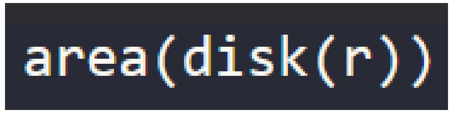
mul pi square r

RULES

area disk d → mul pi square d

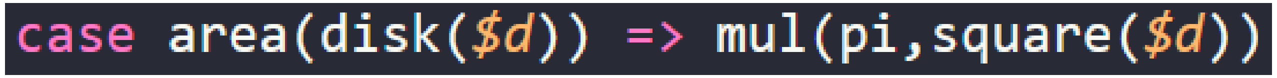
FunBlocks

Declarative paradigm →



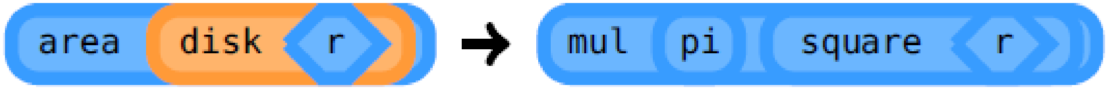
```
area(disk(r))
```

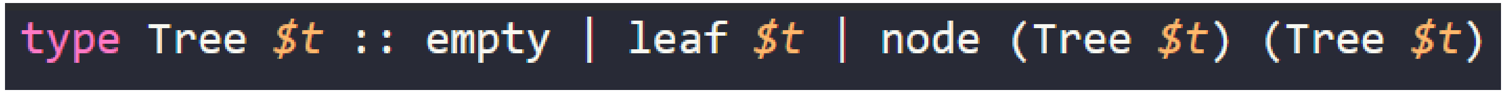
Based on rewrite systems →



```
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

Zero = {0}

Nat = Zero ∪ succ(Nat)

Empty = Λ

Stack = Empty ∪ push(Nat, Stack)

top : Stack → Nat

pop : Stack → Stack

alternate : Stack × Stack → Stack

Rewrite systems

Canonical rewrite system

top(push(x, y)) = x	top(push(x, y)) → x
pop(push(x, y)) = y	pop(push(x, y)) → y
alternate(Λ, z) = z	alternate(Λ , z) → z
alternate(push(x, y), z) = push(x, alternate(z, y))	alternate(push(x, y), z) → push(x, alternate(z, y))

Rewrite systems

Termination

Confluence

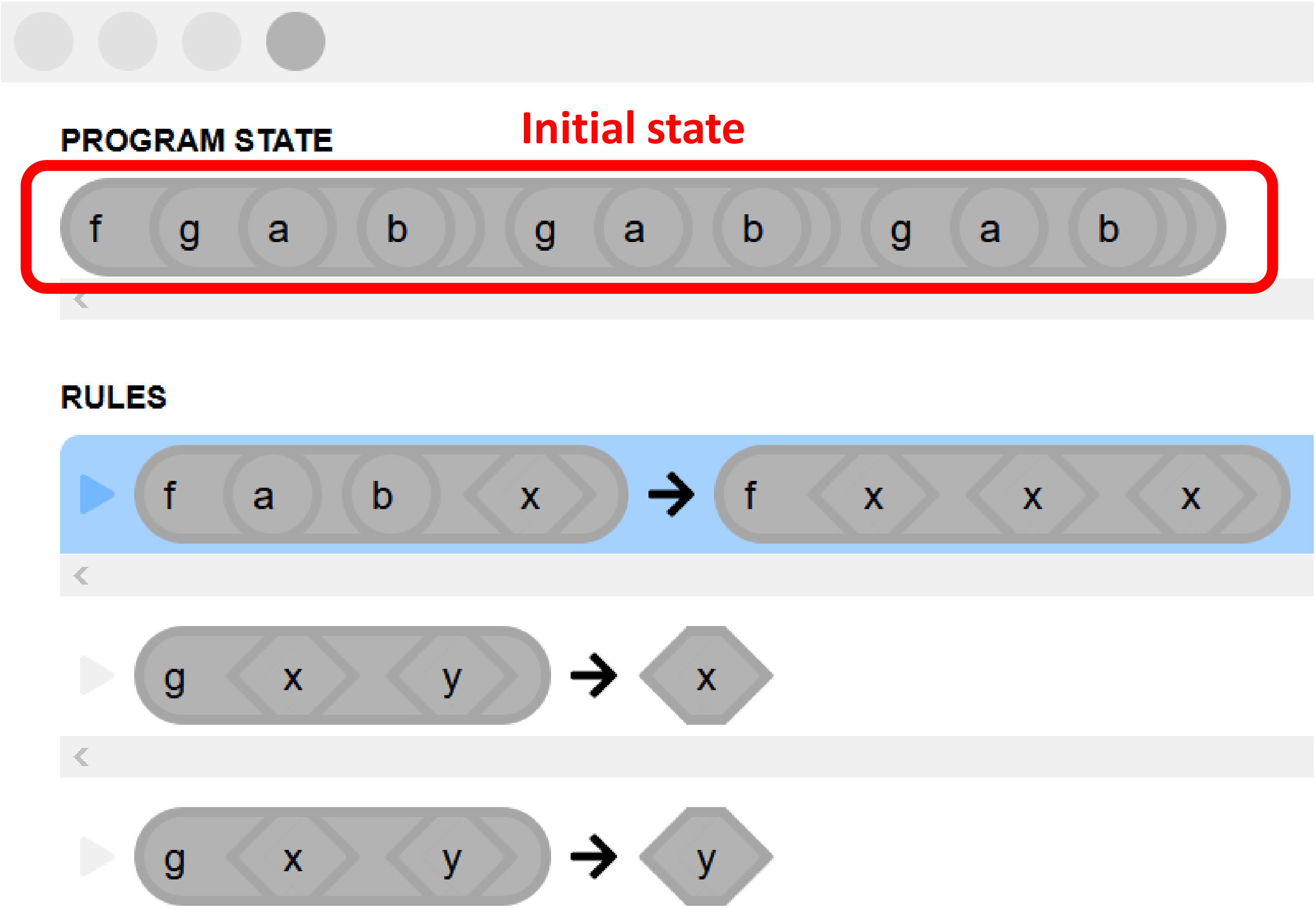
Soundness

Completeness

Correctness

Undecidable in general

Termination



Initial state

PROGRAM STATE

f g a b g a b g a b

RULES

f a b x → f x x x

g x y → x

g x y → y

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

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

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

$$\xrightarrow{\text{weight}}$$

$$w(a) = 1$$

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

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

Polynomial interpretation

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

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

$$w(x) = \boldsymbol{w}(\boldsymbol{x})$$

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

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

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

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

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

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

AProVE

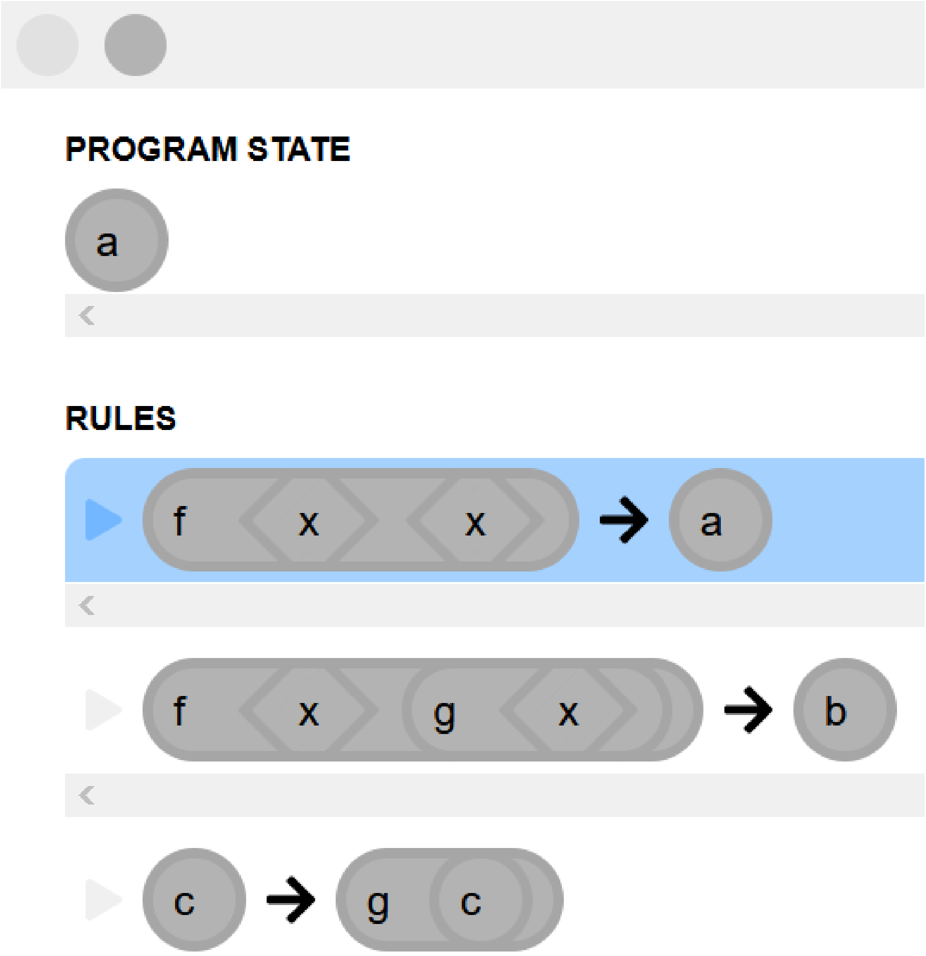
=

Direct proof
(polynomial, LBO, KBO,…)

+

Dependency pairs and
size-change principle

Confluence



PROGRAM STATE

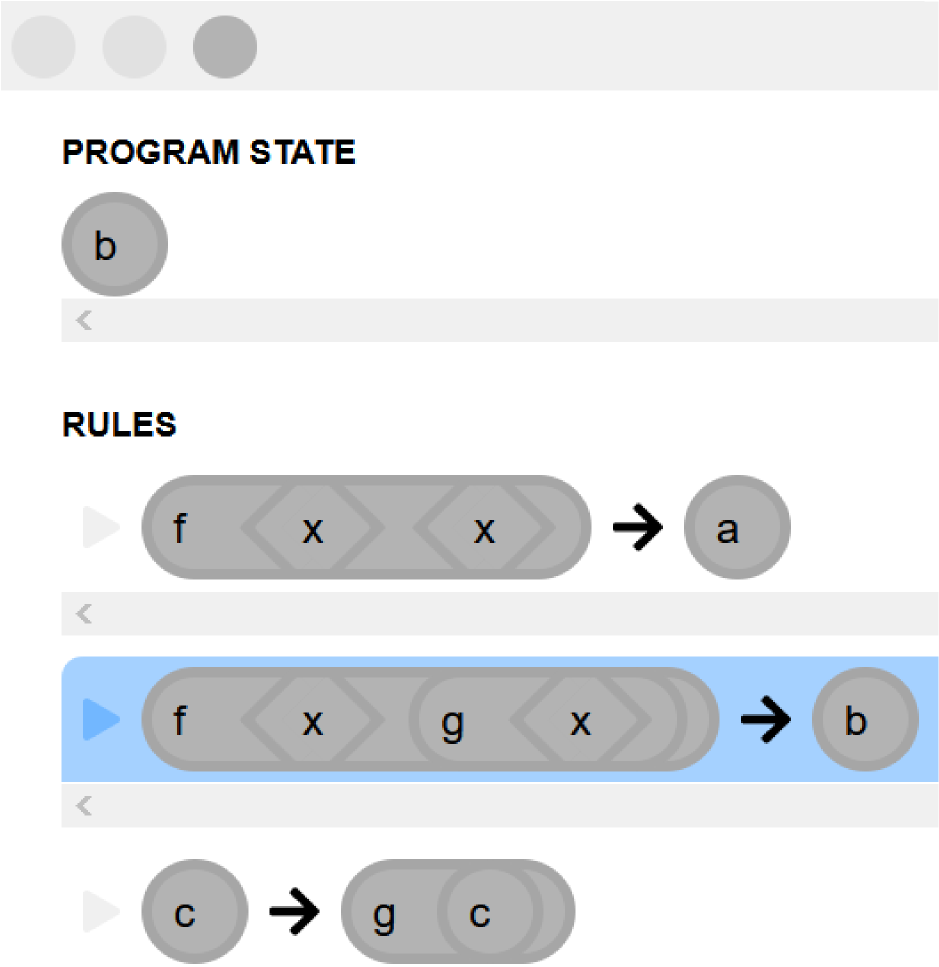
a

RULES

f x x → a

f x g x → b

c → g c



PROGRAM STATE

b

RULES

f x x → a

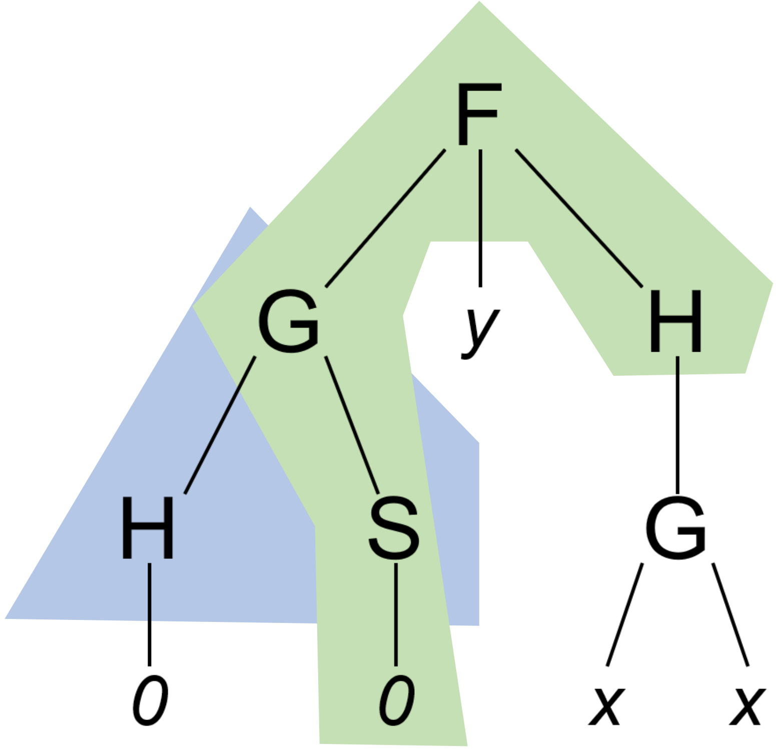
f x g x → b

c → g c

Overlap and critical pairs

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

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



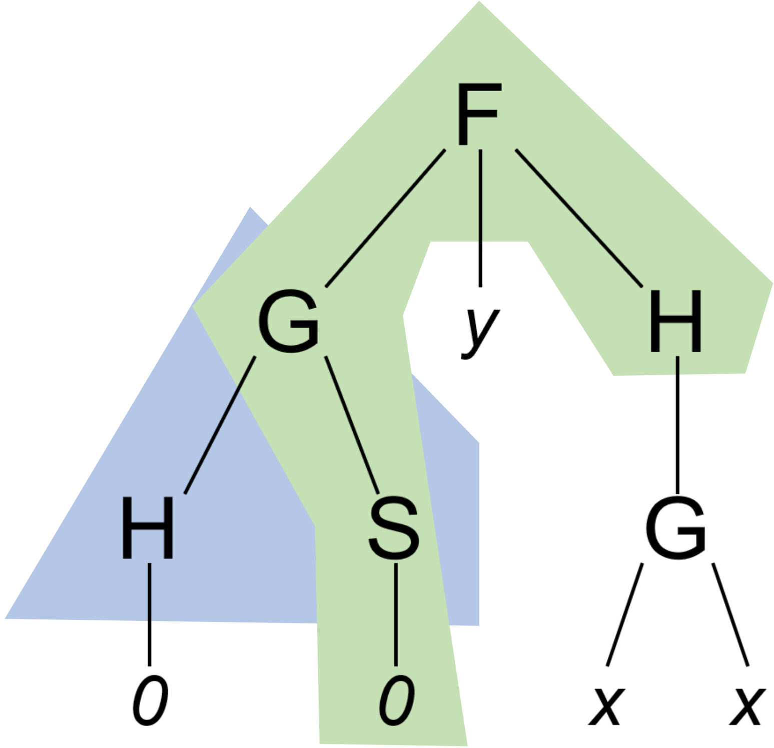
Overlap and critical pairs

Overlapping:

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

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

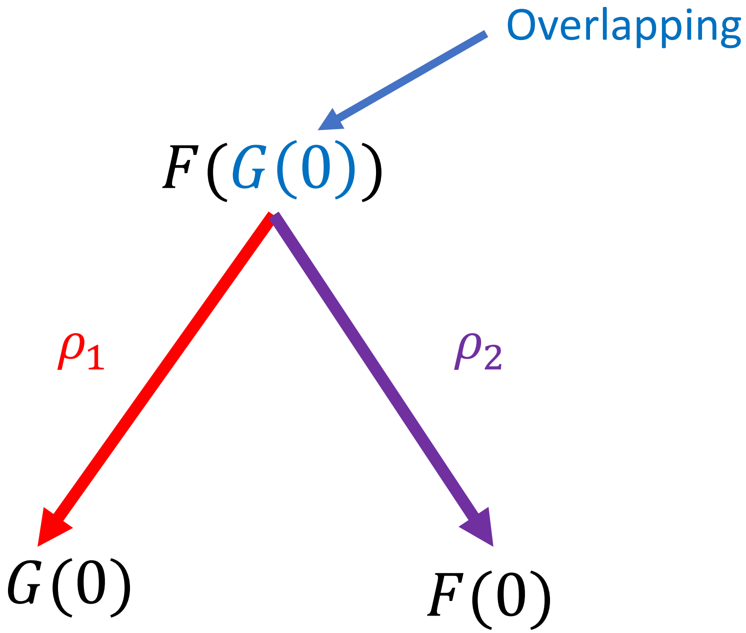
$G(H(\square), S(\square))$



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

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

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

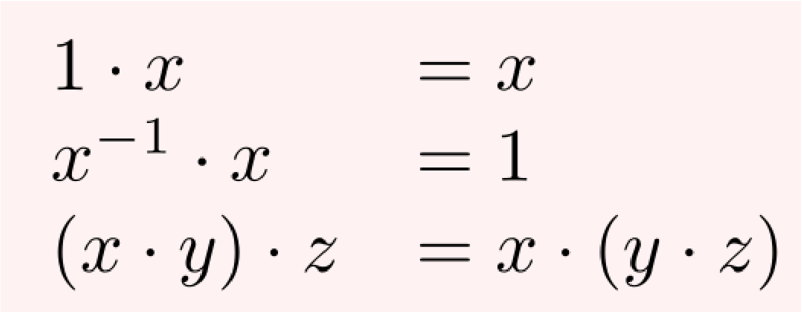
$$\begin{tikzcd}
 & F(G(0)) \arrow[dl, "\rho_1"'] \arrow[dr, "\rho_2"] & \\
G(0) \arrow[ddr, "\rho_2"] & & F(0) \arrow[d, "\rho_1"] \\
 & & G(0) \arrow[dl, "\rho_2"] \\
\text{Convergence} \arrow[r] & 0 &
\end{tikzcd}$$

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 $\longrightarrow$ Add $s = t$ in the set of equation

$$\begin{aligned}(x*y)*z &\to x*(y*z)\\ x*x &\to x\end{aligned}$$

$$\begin{tikzcd} & (x*x)*z \arrow[dl] \arrow[dr] & \\ x*z & & x*(x*z) \end{tikzcd}$$

No convergent

$\longrightarrow$

$$\begin{aligned}(x*y)*z &\to x*(y*z)\\ x*x &\to x\\ &\boxed{x*(x*z) \to x*z}\end{aligned}$$

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:

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

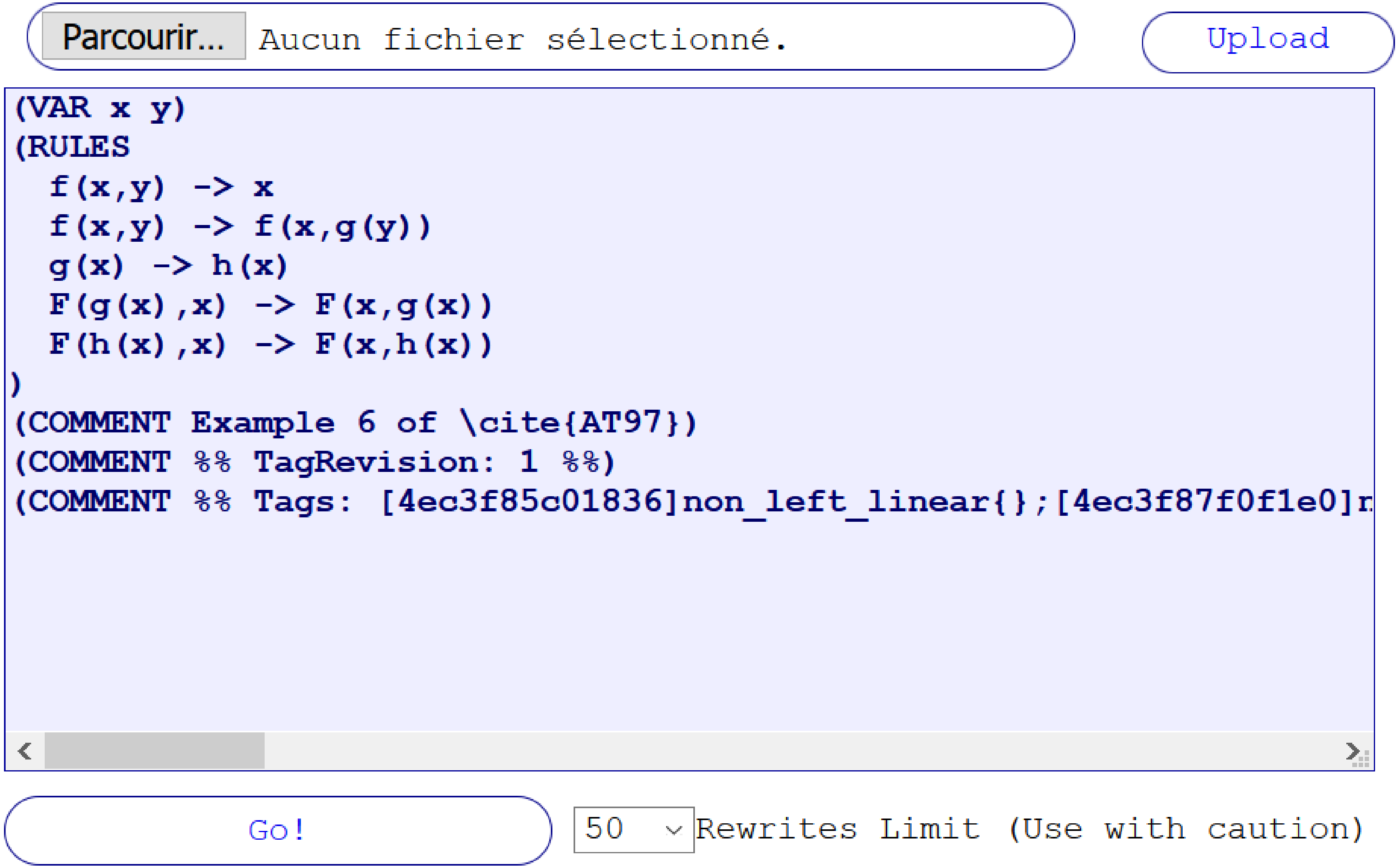
$$\longrightarrow$$

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

Educational tools

TTT2/TRS TOOL	CSI	KBCV
Termination	Confluence	Completion

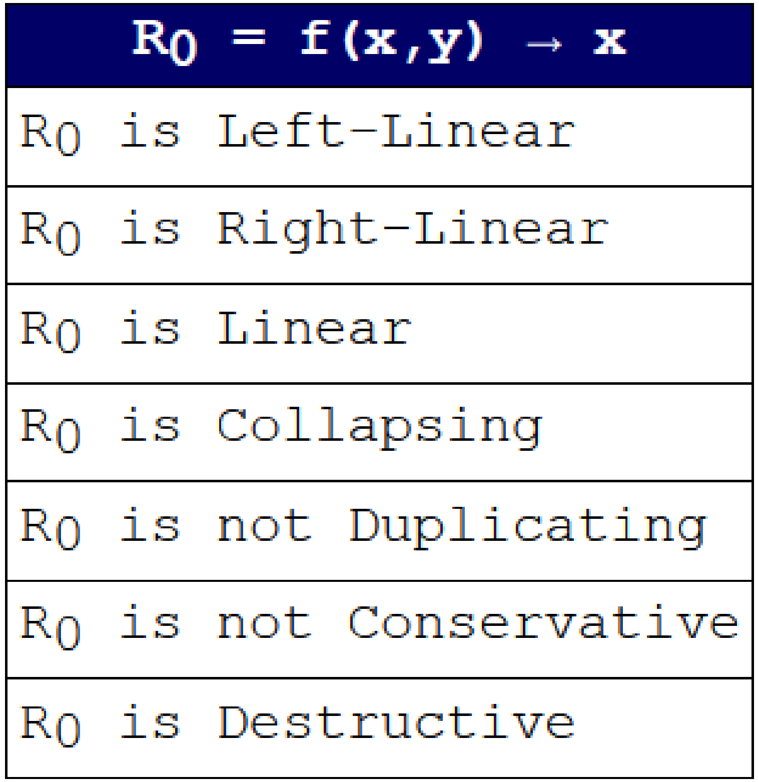
TRS tool



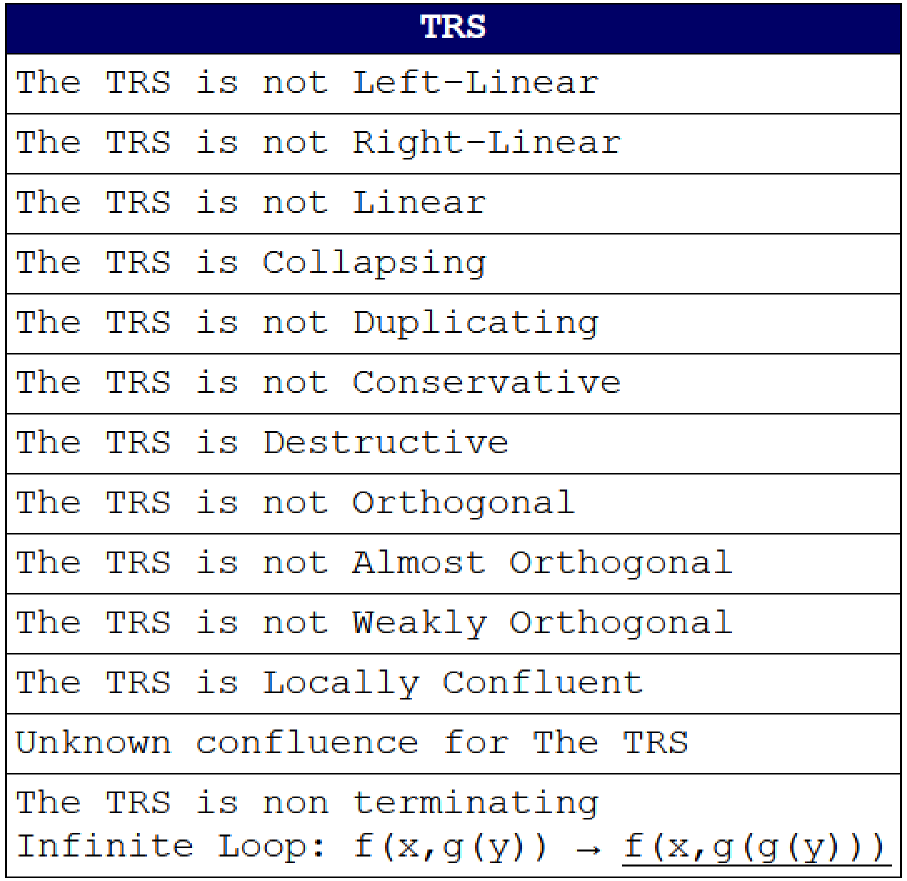
```
(VAR x y)
(RULES
  f(x,y) -> x
  f(x,y) -> f(x,g(y))
  g(x) -> h(x)
  F(g(x),x) -> F(x,g(x))
  F(h(x),x) -> F(x,h(x))
)
(COMMENT Example 6 of \cite{AT97})
(COMMENT %% TagRevision: 1 %%)
(COMMENT %% Tags: [4ec3f85c01836]non_left_linear{};[4ec3f87f0f1e0]r
```

50 Rewrites Limit (Use with caution)

TRS tool

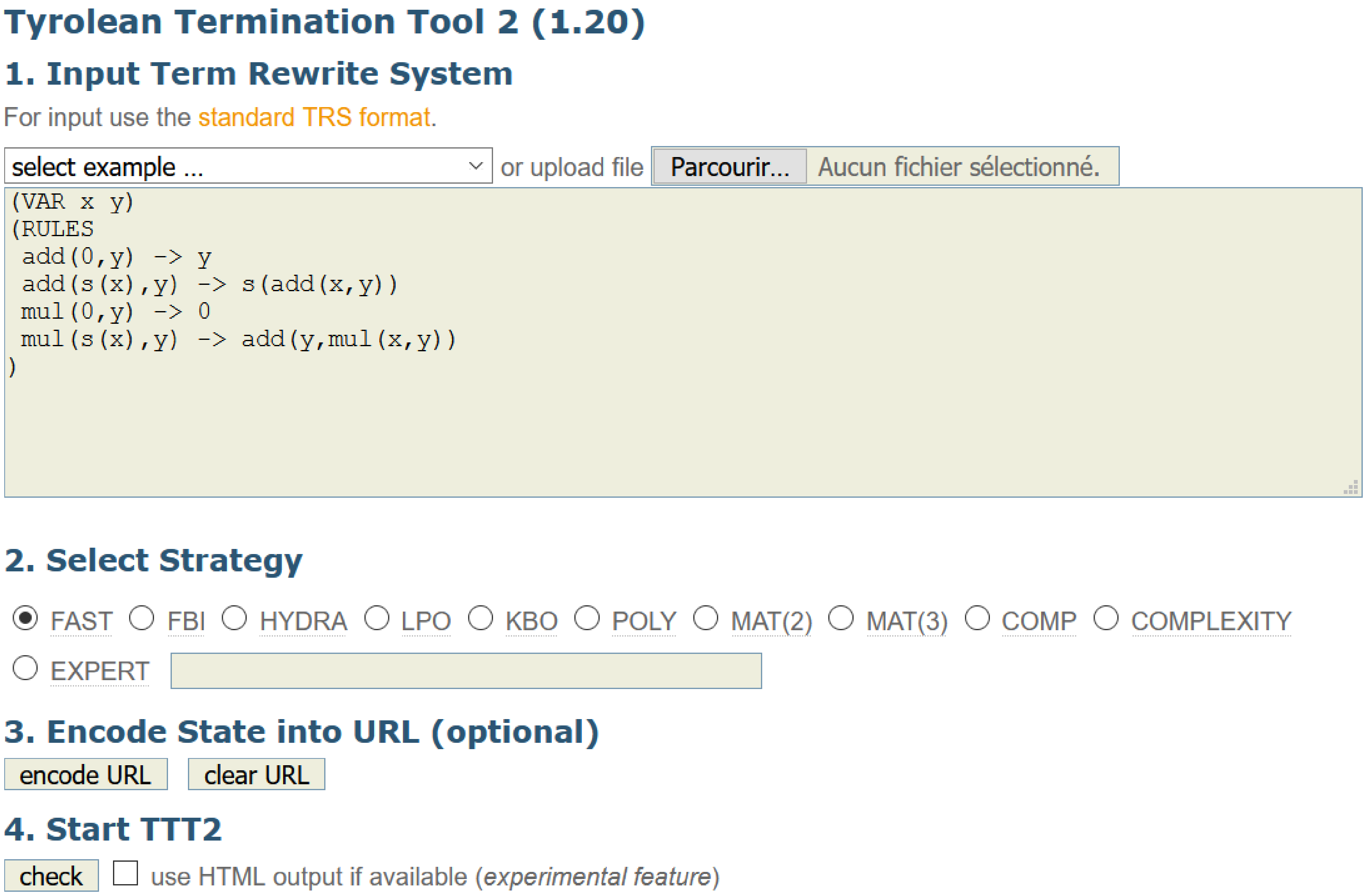


$R_0 = f(x,y) \to x$
R_0 is Left-Linear
R_0 is Right-Linear
R_0 is Linear
R_0 is Collapsing
R_0 is not Duplicating
R_0 is not Conservative
R_0 is Destructive



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

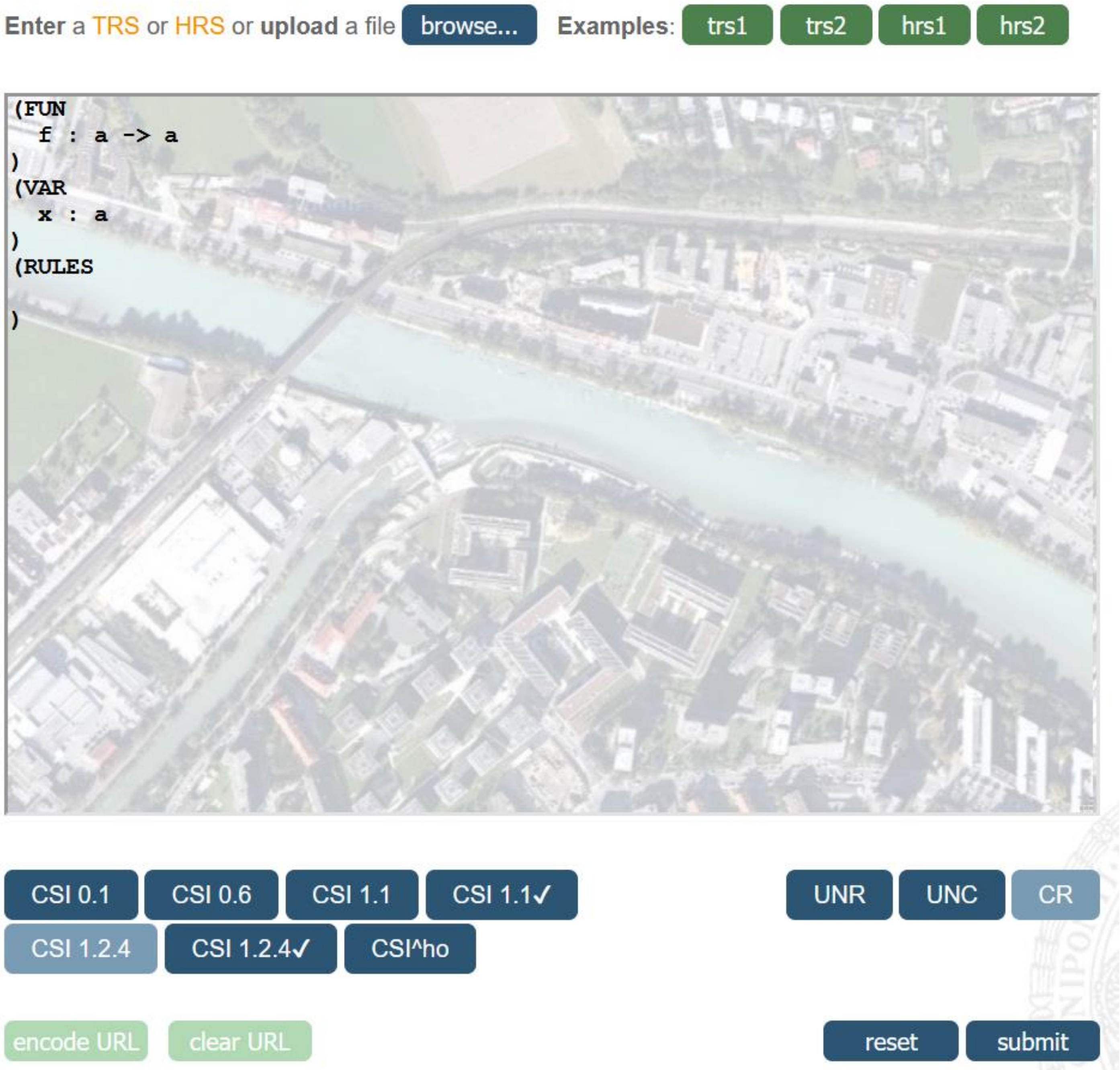
3. Encode State into URL (optional)

encode URL clear URL

4. Start TTT2

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

CSI

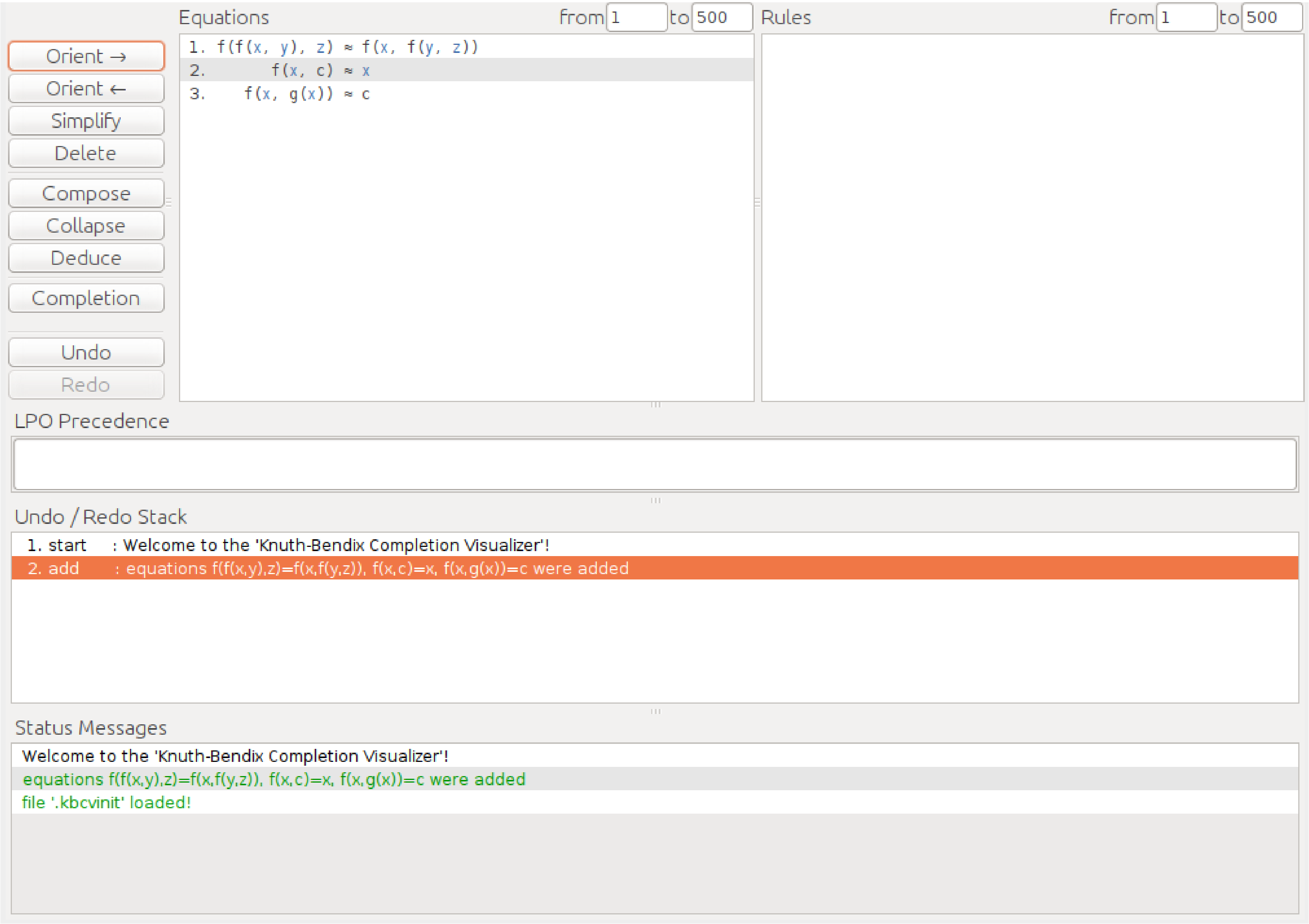


Enter a TRS or HRS or upload a file

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

)
```

KBCV



Equations

```
1. f(f(x, y), z) ≈ f(x, f(y, z))
2.       f(x, c) ≈ x
3.    f(x, g(x)) ≈ c
```

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

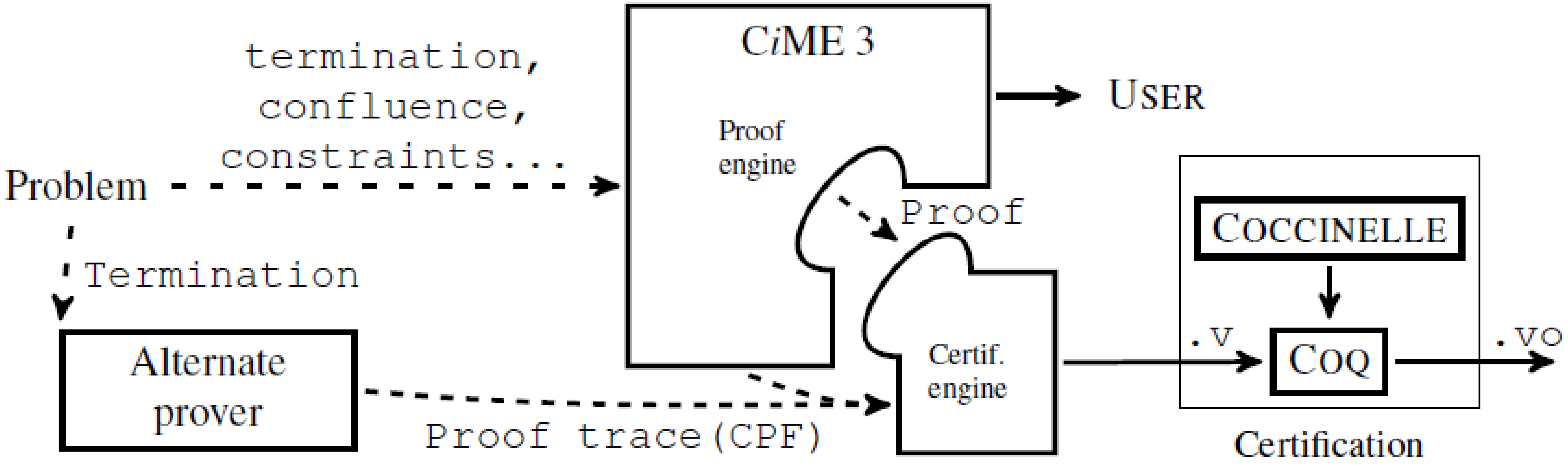
_ → _

Rewriting toolkit

Proof engine

Proof certification

CiME


termination,
confluence,
constraints...
Problem
Termination
Alternate prover
Proof trace(CPF)
CiME 3
Proof engine
Proof
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

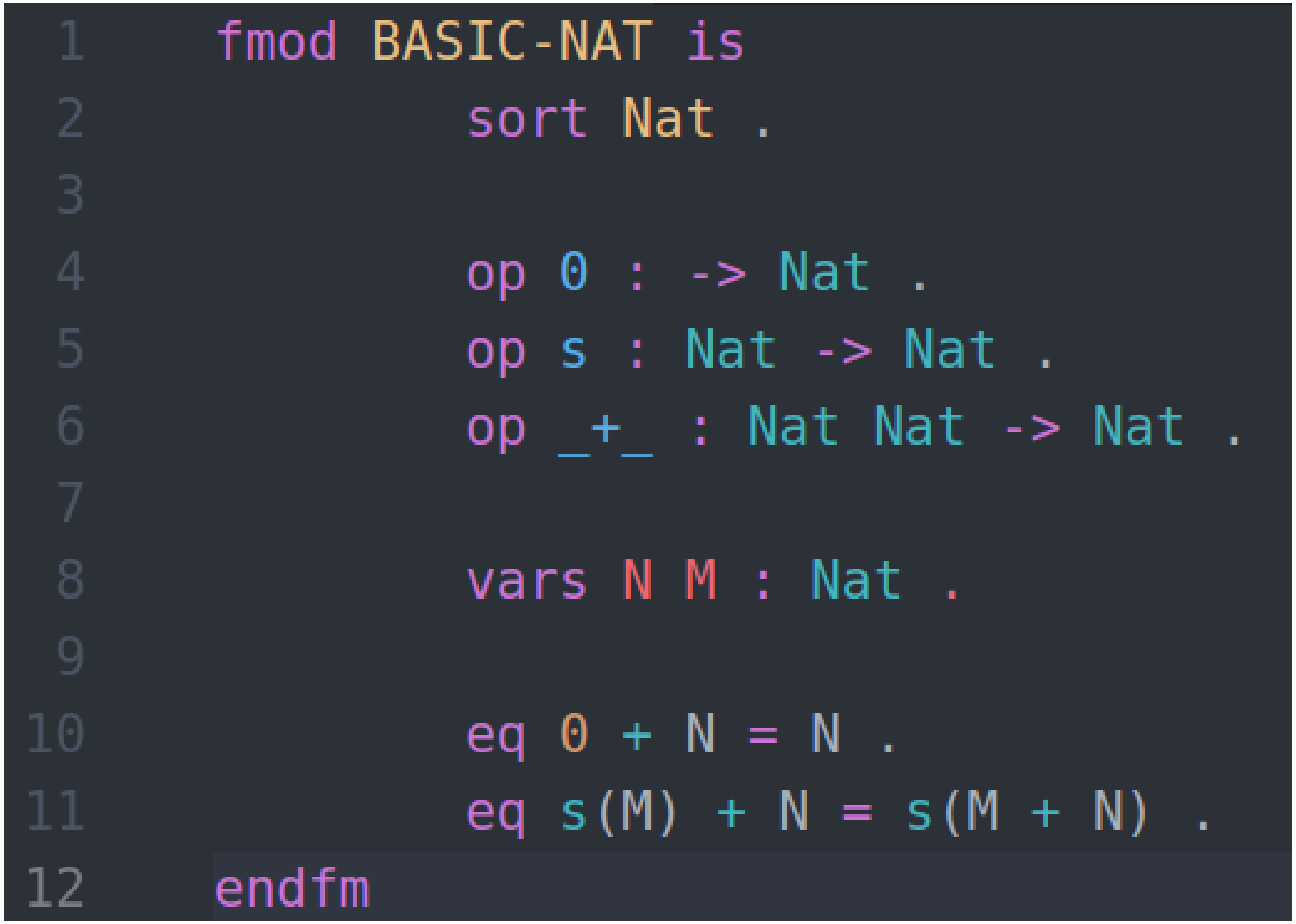
MaudEƎ

Simplicity

Expressiveness

Performance

Maude



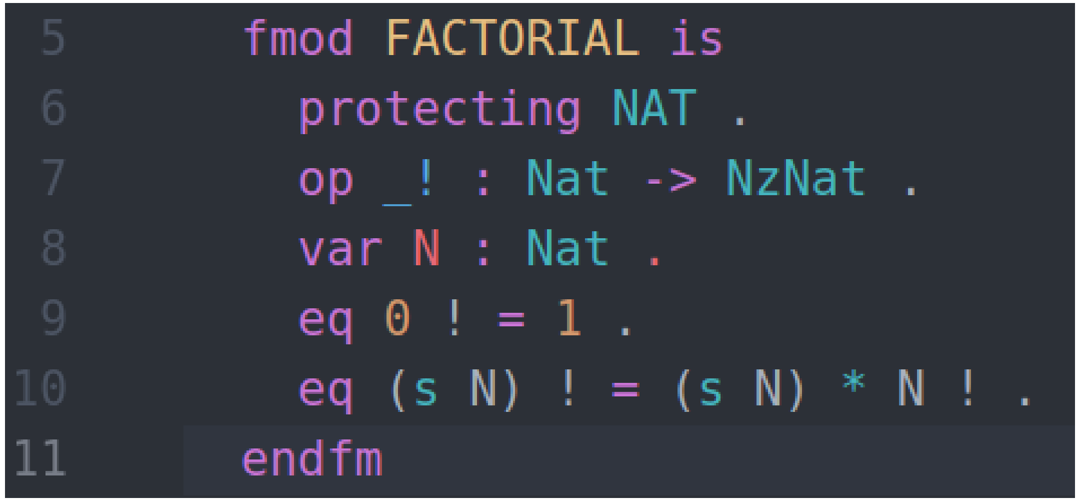
```
fmod BASIC-NAT is
    sort Nat .

    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
```

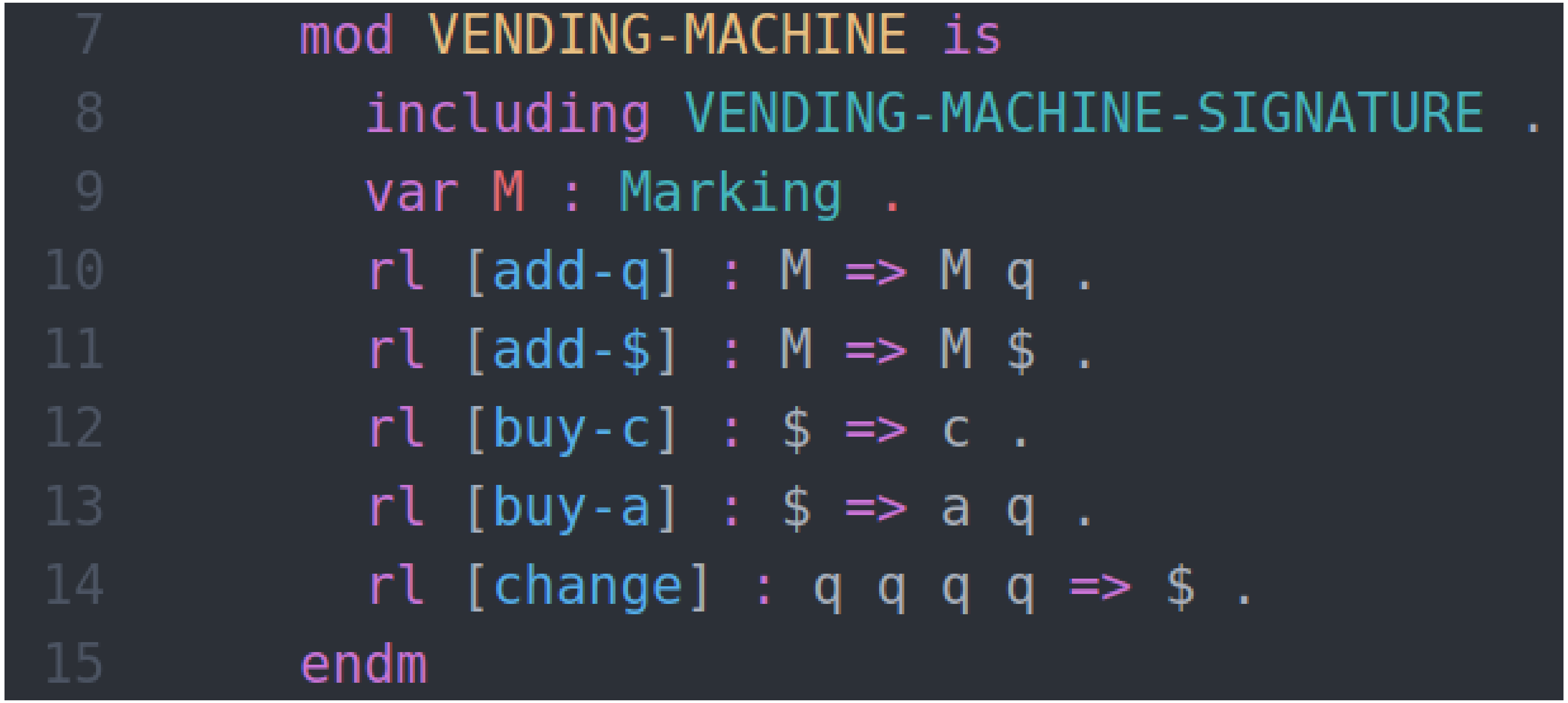
Maude



```
fmod FACTORIAL is
  protecting NAT .
  op _! : Nat -> NzNat .
  var N : Nat .
  eq 0 ! = 1 .
  eq (s N) ! = (s N) * N ! .
endfm
```

```
> load factorial.maude
> red 100 ! .
Reduce in FACTORIAL : 100 ! .
rewrites: 201 in 0ms cpu (0ms real) (~ rewrites/second)
result NzNAT:
9332621544394415268169923885626670049071596826438162146
8592963895217599993229915608941463976156518286253697920
827223758251185210916864000000000000000000000000
```

Maude



```
mod VENDING-MACHINE is
  including VENDING-MACHINE-SIGNATURE .
  var M : Marking .
  rl [add-q] : M => M q .
  rl [add-$] : M => M $ .
  rl [buy-c] : $ => c .
  rl [buy-a] : $ => a q .
  rl [change] : q q q q => $ .
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	+	−

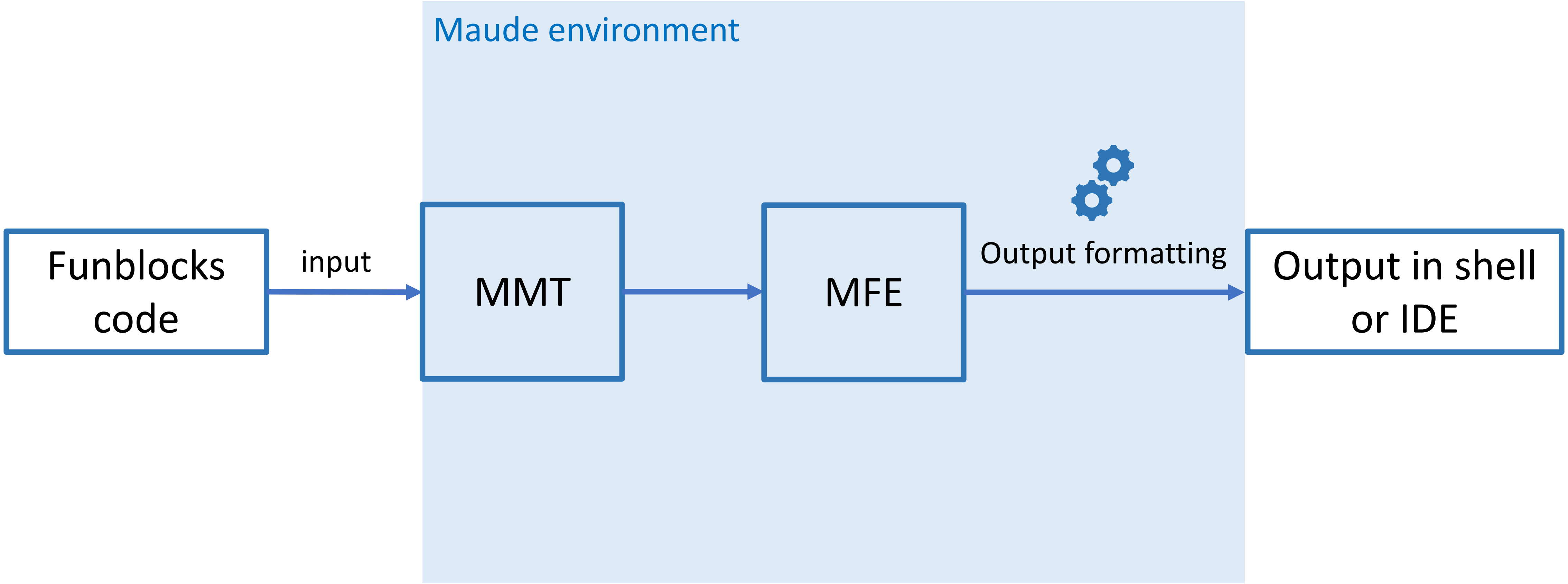
Maude MSOS Tool (MMT)

Language specification

Based on transition rules

Modularity

System diagram



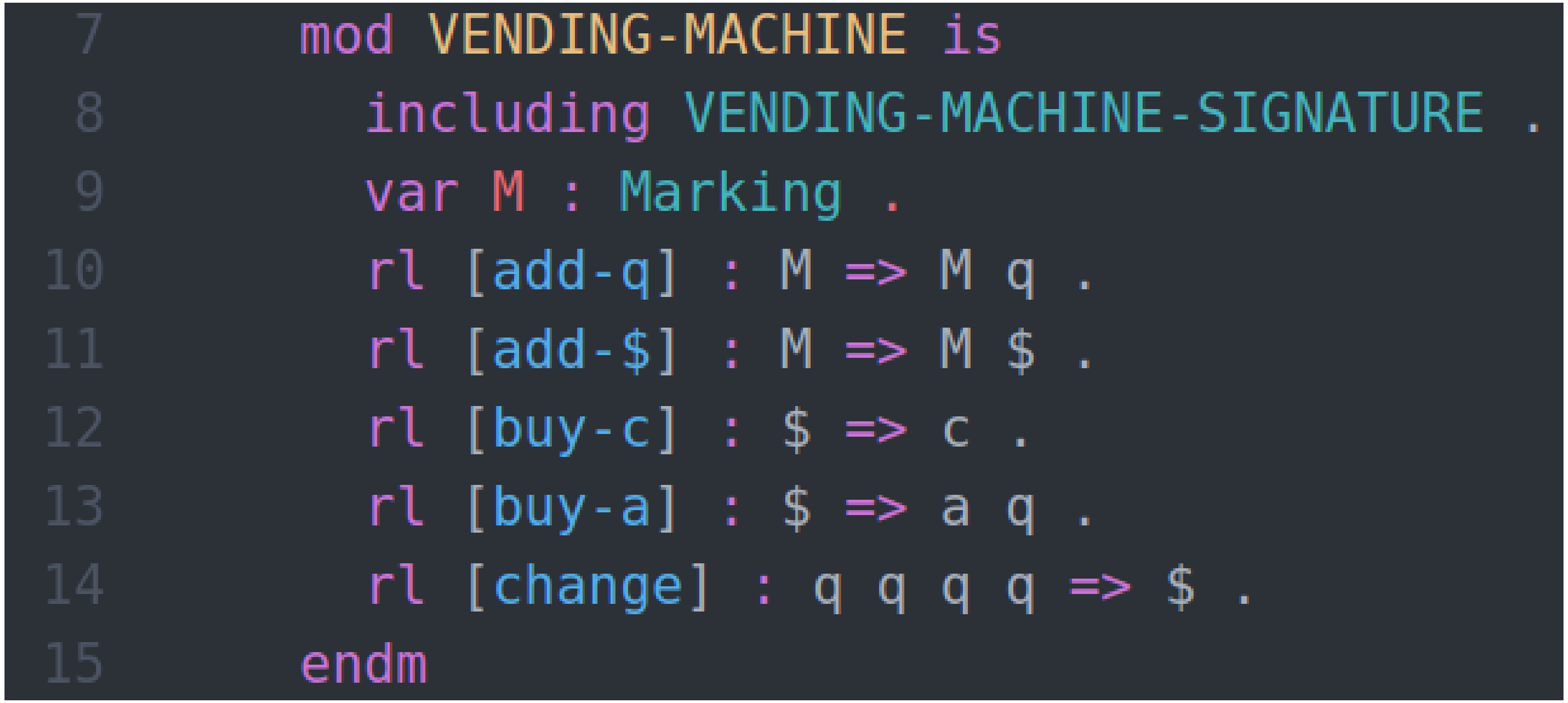
I/O in Maude

Standard input/output

File handling

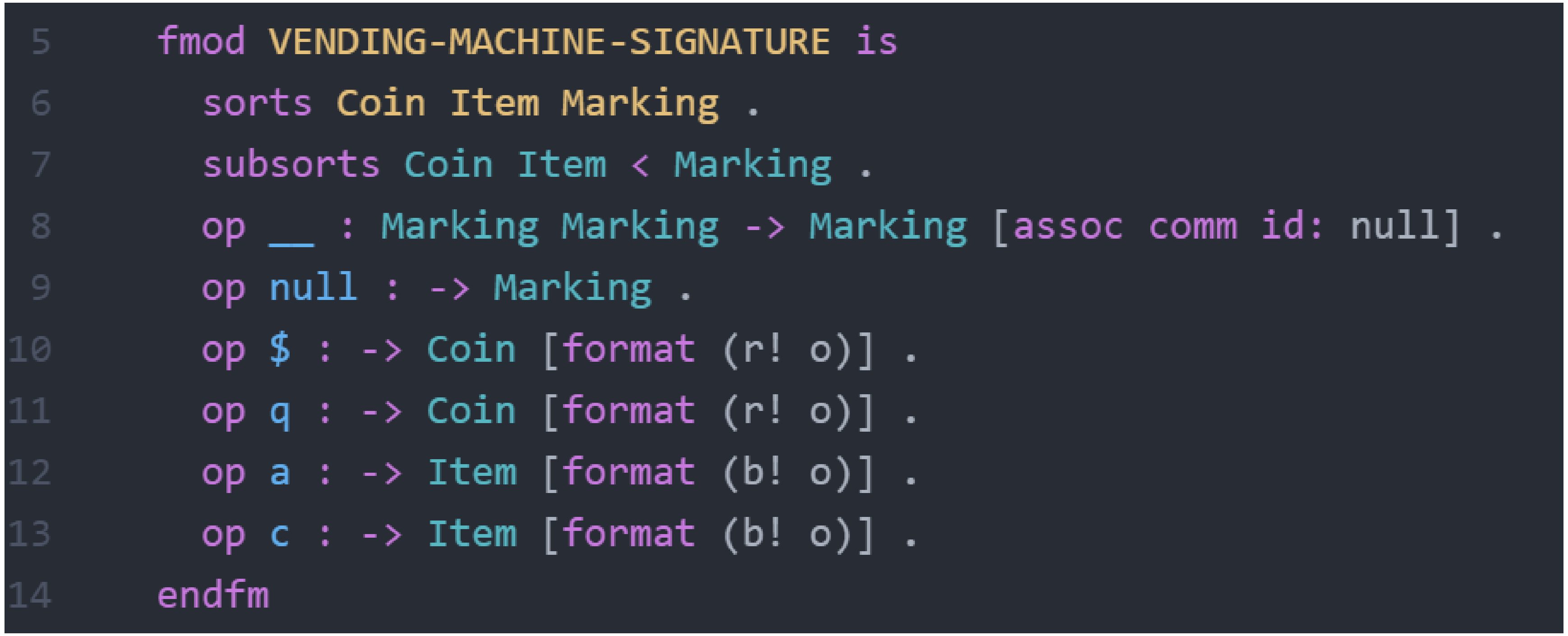
Sockets

I/O in Maude



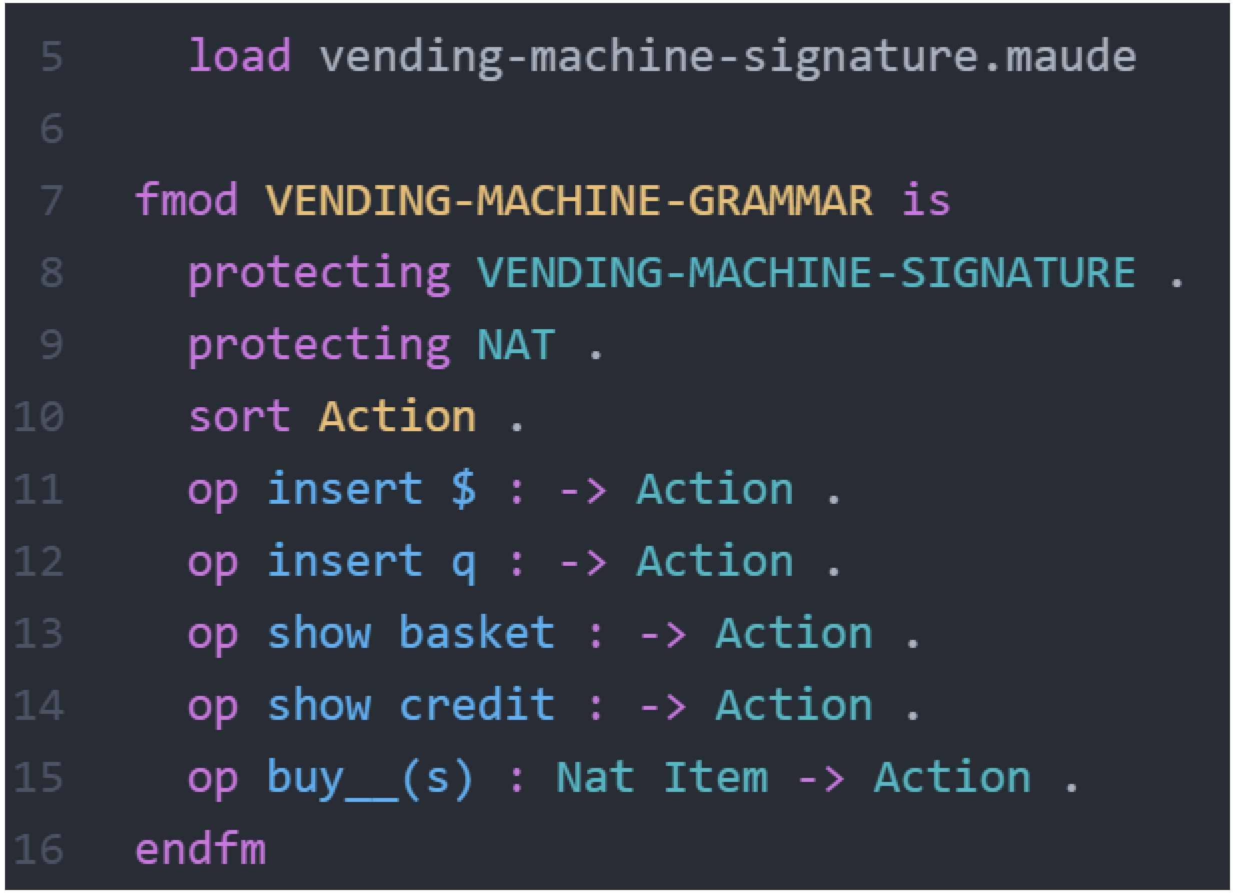
```
mod VENDING-MACHINE is
  including VENDING-MACHINE-SIGNATURE .
  var M : Marking .
  rl [add-q] : M => M q .
  rl [add-$] : M => M $ .
  rl [buy-c] : $ => c .
  rl [buy-a] : $ => a q .
  rl [change] : q q q q => $ .
endm
```

I/O in Maude



```
fmod VENDING-MACHINE-SIGNATURE is
  sorts Coin Item Marking .
  subsorts Coin Item < Marking .
  op __ : Marking Marking -> Marking [assoc comm id: null] .
  op null : -> Marking .
  op $ : -> Coin [format (r! o)] .
  op q : -> Coin [format (r! o)] .
  op a : -> Item [format (b! o)] .
  op c : -> Item [format (b! o)] .
endfm
```

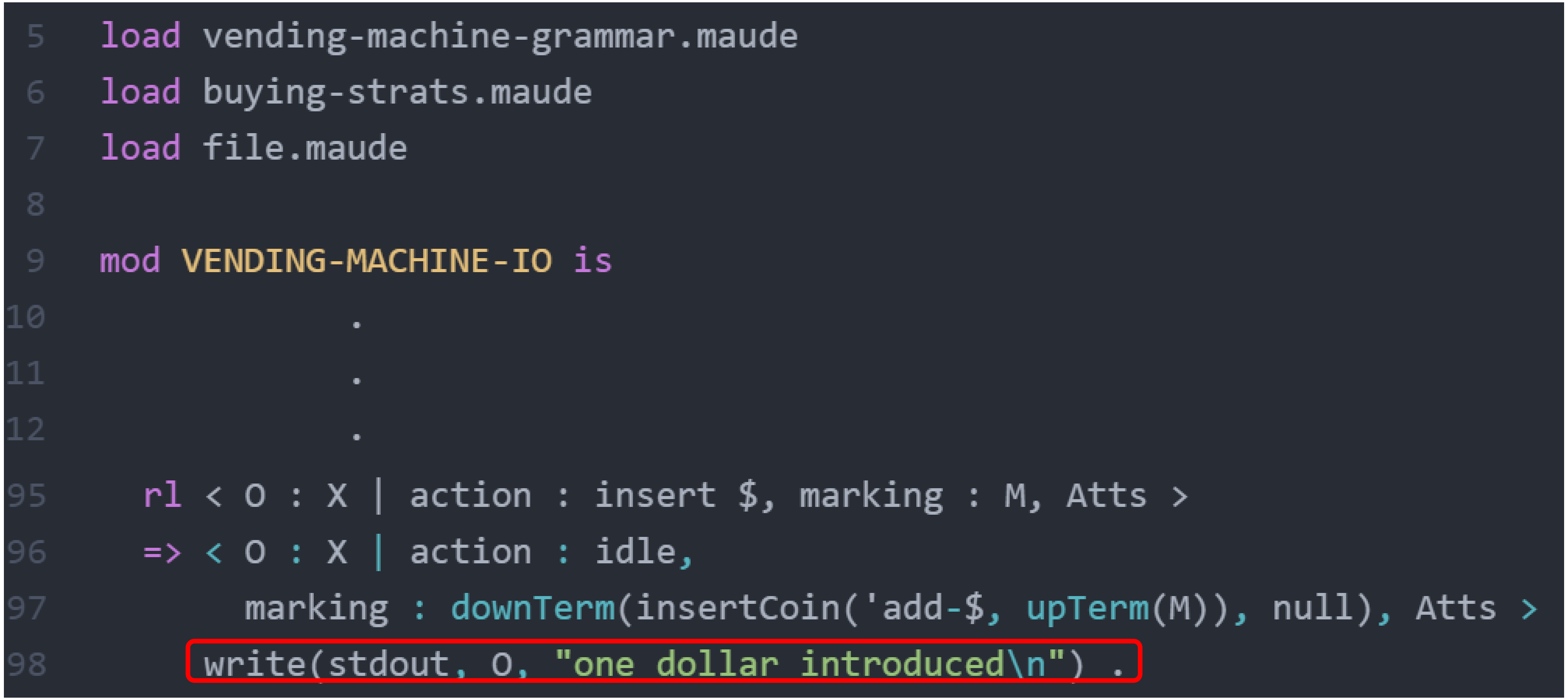
I/O in Maude



```
  load vending-machine-signature.maude

fmod VENDING-MACHINE-GRAMMAR is
  protecting VENDING-MACHINE-SIGNATURE .
  protecting NAT .
  sort Action .
  op insert $ : -> Action .
  op insert q : -> Action .
  op show basket : -> Action .
  op show credit : -> Action .
  op buy__(s) : Nat Item -> Action .
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?

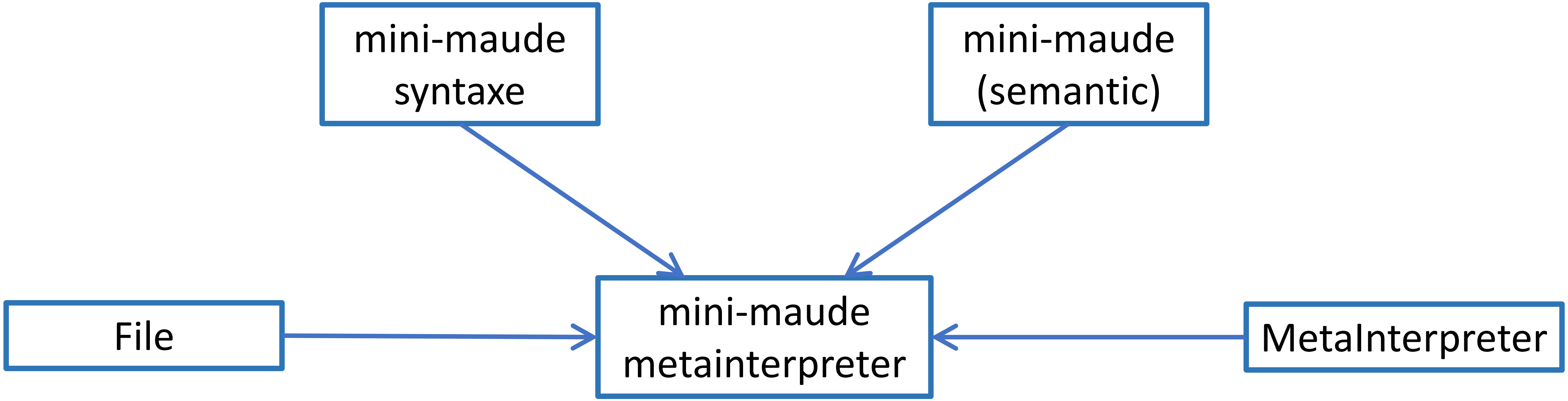
META-LEVEL module

Parse input

Execute

Print

MINI-MAUDE


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

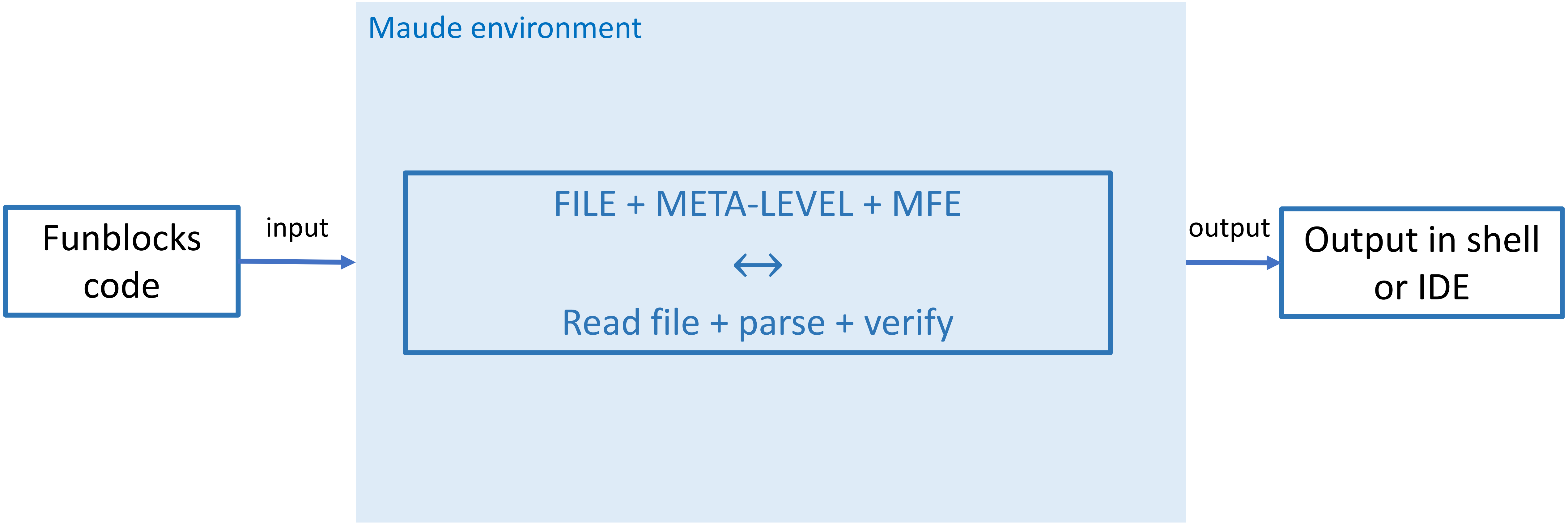
MINI-MAUDE

```
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



Maude environment

Funblocks code

input

FILE + META-LEVEL + MFE

↔

Read file + parse + verify

output

Output in shell or IDE

What's next?

META-LEVEL module

Handle and verify FunBlock file input

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