Contribution to $\mathbb{K}$ Framework

Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

22/06/2011, 8th MC Meeting, Action IC0701, Limerick

Plan

$\mathbb{K}$ Project

Started in 2003 by Grigore Rosu at UIUC, motivated mainly by teaching programming languages and noticing that the existing semantic frameworks have limitations

Project thesis:

Rewriting gives an appropriate environment to formally define the semantics of real-life programming languages and to test and analyze programs written in those languages.

𝕂ontributors

Joint work between Formal Systems Laboratory (FSL) from University of Illinois at Urbana-Champaign (UIUC) lead by Grigore Roșu and Formal Methods in Software Engineering (FMSE) from Al. I. Cuza University (UAIC) lead by presenter

UIUC team:
Chucky Ellison, Michael Ilseman, Patrick Meredith, Grigore Roșu, Traian Șerbănuță, Andrei Ștefănescu, David Lazar

UAIC team:
Andrei Arusoae, Irina Mariuca Asavoae, Mihai Asavoae, Gheorghe Grigoras, Dorel Lucanu, Radu Mereuta, Elena Naum

Plan

1. Introduction
2. 𝕂 Framework
    - Motivation
    - K Framework Solution
    - Matching Logic
3. 𝕂ontributions
4. Conclusion

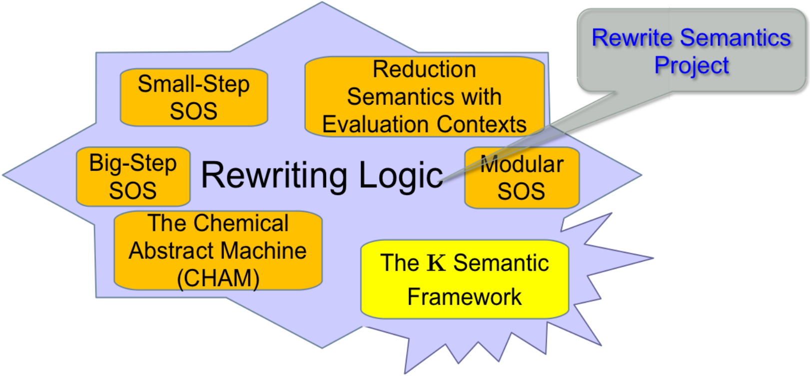
Motivation

- The Semantics of Programming languages is informally presented in manuals
- Each model checker, static verifier, run-time verifier of the same language L uses its own encoding of L
- Therefore programming languages must have formal semantics
- Executable specifications could help
  - Design and maintain mathematical definitions
  - Easily test/analyze language updates/extensions
  - Explore/Abstract non-deterministic executions
- one definition for L and develop the other tools w.r.t. this definition

Shortcomings of Existing Frameworks

- Hard to deal with control (except evaluation contexts)
  - halt, break/continue, exceptions
- Non-modular (except Modular SOS)
  - Adding new features require changing unrelated rules
- Lack of semantics for true concurrency (except CHAM)
  - Big-Step captures only all possible results of computation
  - Reduction approaches only give interleaving semantics
- Tedious to find next redex (except evaluation contexts)
  - One has to write the same descent rules for each construct
- Inefficient as interpreters (except for Big-Step SOS)

𝕂 Roots are in Rewriting Semantics Project



[J. Meseguer, G. Roșu, T. Șerbănuță]

Distinguishable $\mathbb{K}$ Features

$\mathbb{K}$ technique:
for expressive, modular, versatile, and clear PL definitions

$\mathbb{K}$ rewriting:
more concurrent than regular rewriting

Representable (e.g., in RWL) for execution, testing and analysis purposes

𝕂 Ingredients

- Computations
  - Sequences of tasks, including syntax
  - Capture the sequential fragment of programming languages
  - Syntax annotations specify order of evaluation
- Configurations
  - Multisets (bags) of nested cells
  - High potential for concurrency and modularity
- 𝕂 rules
  - Specify only what needed, precisely identify what changes
  - More concise, modular, and concurrent than regular rewrite rule

K in a Nutshell

- the semantics is given by means of a set of K rules transforming the abstract syntax trees (ASTs) into results, eventually using some intermediate structures
- the notion of result is a generic one: could be either the output, the result of a type-checking algorithm, the result of a static analyser/verifier and so on
- the machine on which the programs are executed is abstractly described as a configuration of cells
- K Rewrite Abstract Machine (KRAM) executes the rewrite rules in faithful way

Running example: Cink

a kernel of C

- functions
- int expressions
- simple input/output
- basic flow control (if, if-else, while, sequential composition)
- pointers and arrays
- structures

in this talk

- a K semantic definition of Cink (without pointers and structures)

𝕂 computations and 𝕂 syntax

Computations

- Extend PL syntax with a “task sequentialization” operation
  - $t_1 \curvearrowright t_2 \curvearrowright \ldots \curvearrowright t_n$, where t_i are computational tasks
- Computational tasks: pieces of syntax (with holes), closures, ...
- Mostly under the hood, via intuitive PL syntax annotations

𝕂 Syntax: BNF syntax annotated with strictness

$$
\begin{array}{llrl}
\mathit{Exp} ::= \mathit{Id} & & & \\
\quad \mid & \mathit{Exp} + \mathit{Exp} & [\text{strict}] & \mathit{ERed} + \mathit{ERed} \rightleftharpoons \mathit{ERed} \curvearrowright \square + \mathit{ERed} \\
\quad \mid & \mathit{Exp} = \mathit{Exp} & [\text{strict(2)}] & E = \mathit{ERed} \rightleftharpoons \mathit{ERed} \curvearrowright E = \square \\
\mathit{Stmt} ::= & \mathit{Exp}\ ; & [\text{strict}] & \mathit{ERed}\ ; \rightleftharpoons \mathit{ERed} \curvearrowright \square\ ; \\
\quad \mid & \mathit{Stmt}\ \ \mathit{Stmt} & [\text{seqstrict}] & \mathit{SRed}\ \ S \rightleftharpoons \mathit{SRed} \curvearrowright \square\ \ S
\end{array}
$$

Heating syntax through strictness rules

Computation

y = x+2 ; x = 7 ;

𝕂 Syntax: BNF syntax annotated with strictness

$$
\begin{array}{lllcl}
Exp ::= Id & & & & \\
\quad \mid \ Exp + Exp & [\mathrm{strict}] & ERed + ERed & \rightleftharpoons & ERed \curvearrowright \square + ERed \\
\quad \mid \ Exp = Exp & [\mathrm{strict(2)}] & E = ERed & \rightleftharpoons & ERed \curvearrowright E = \square \\
Stmt ::= Exp \ ; & [\mathrm{strict}] & ERed \ ; & \rightleftharpoons & ERed \curvearrowright \square \ ; \\
\quad \mid \ Stmt \ Stmt & [\mathrm{seqstrict}] & SRed \ S & \rightleftharpoons & SRed \curvearrowright \square \ S
\end{array}
$$

Heating syntax through strictness rules

Computation

$y = x{+}2\ ;\ \curvearrowright \square\ \ x = 7\ ;$

𝕂 Syntax: BNF syntax annotated with strictness

$Exp ::= Id$			
$\mid\ Exp + Exp$	[strict]	$ERed + ERed$	$\rightleftharpoons\ ERed \curvearrowright \square + ERed$
$\mid\ Exp = Exp$	[strict(2)]	$E = ERed$	$\rightleftharpoons\ ERed \curvearrowright E = \square$
$Stmt ::= Exp\ ;$	[strict]	$ERed\ ;$	$\rightleftharpoons\ ERed \curvearrowright \square\ ;$
$\mid\ Stmt\ \ Stmt$	[seqstrict]	$SRed\ \ S$	$\rightleftharpoons\ SRed \curvearrowright \square\ \ S$

Heating syntax through strictness rules

Computation

$$\mathrm{y = x{+}2} \curvearrowright \square ; \curvearrowright \square \ \mathrm{x = 7} \ ;$$

𝕂 Syntax: BNF syntax annotated with strictness

$$\begin{array}{lllrlcl}
\mathit{Exp} ::= & \mathit{Id} & & & & & \\
\mid & \mathit{Exp} + \mathit{Exp} & & [\mathrm{strict}] & \mathit{ERed} + \mathit{ERed} & \rightleftharpoons & \mathit{ERed} \curvearrowright \square + \mathit{ERed} \\
\mid & \mathit{Exp} = \mathit{Exp} & & [\mathrm{strict}(2)] & E = \mathit{ERed} & \rightleftharpoons & \mathit{ERed} \curvearrowright E = \square \\
\mathit{Stmt} ::= & \mathit{Exp}\ ; & & [\mathrm{strict}] & \mathit{ERed}\ ; & \rightleftharpoons & \mathit{ERed} \curvearrowright \square\ ; \\
\mid & \mathit{Stmt}\ \ \mathit{Stmt} & & [\mathrm{seqstrict}] & \mathit{SRed}\ \ S & \rightleftharpoons & \mathit{SRed} \curvearrowright \square\ \ S
\end{array}$$

Heating syntax through strictness rules

Computation

$$\mathrm{x} + 2 \curvearrowright \mathrm{y} = \Box \curvearrowright \Box ; \curvearrowright \Box \ \ \mathrm{x} = 7 \ ;$$

𝕂 Syntax: BNF syntax annotated with strictness

$$\begin{array}{lrl}
\mathit{Exp} ::= \mathit{Id} & & \\
\quad \mid \ \mathit{Exp} + \mathit{Exp} & [\mathrm{strict}] & \mathit{ERed} + \mathit{ERed} \rightleftharpoons \mathit{ERed} \curvearrowright \Box + \mathit{ERed} \\
\quad \mid \ \mathit{Exp} = \mathit{Exp} & [\mathrm{strict}(2)] & E = \mathit{ERed} \rightleftharpoons \mathit{ERed} \curvearrowright E = \Box \\
\mathit{Stmt} ::= \mathit{Exp}\ ; & [\mathrm{strict}] & \mathit{ERed}\ ; \rightleftharpoons \mathit{ERed} \curvearrowright \Box\ ; \\
\quad \mid \ \mathit{Stmt}\ \ \mathit{Stmt} & [\mathrm{seqstrict}] & \mathit{SRed}\ \ S \rightleftharpoons \mathit{SRed} \curvearrowright \Box\ \ S
\end{array}$$

Heating syntax through strictness rules

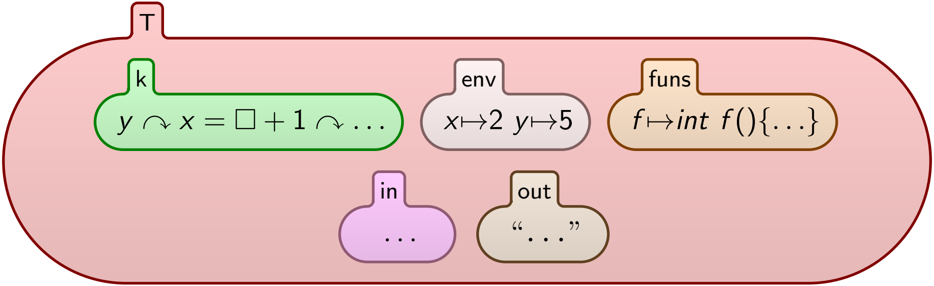
Computation

$$x \curvearrowright \square + 2 \curvearrowright y = \square \curvearrowright \square ; \curvearrowright \square \ \ x = 7 \ ;$$

𝕂 Syntax: BNF syntax annotated with strictness

$Exp ::= Id$		
$\mid\ Exp + Exp$	[strict]	$ERed + ERed \rightleftharpoons ERed \curvearrowright \square + ERed$
$\mid\ Exp = Exp$	[strict(2)]	$E = ERed \rightleftharpoons ERed \curvearrowright E = \square$
$Stmt ::= Exp\ ;$	[strict]	$ERed\ ; \rightleftharpoons ERed \curvearrowright \square\ ;$
$\mid\ Stmt\ \ Stmt$	[seqstrict]	$SRed\ \ S \rightleftharpoons SRed \curvearrowright \square\ \ S$

Cink Configuration



T

k

$y \curvearrowright x = \square + 1 \curvearrowright \ldots$

env

$x \mapsto 2 \ y \mapsto 5$

funs

$f \mapsto int \ f()\{\ldots\}$

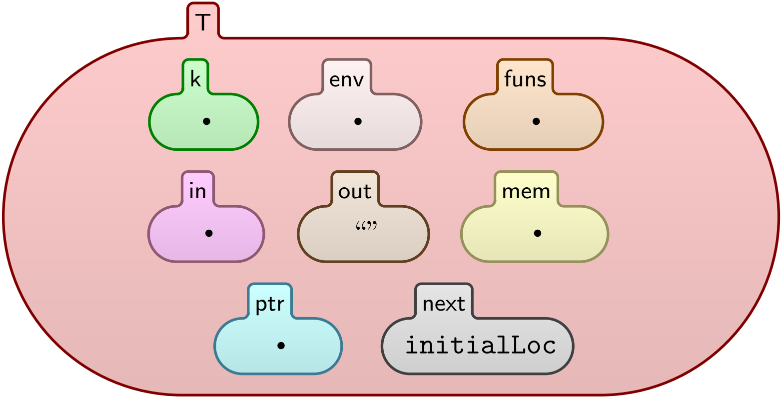
in

$\ldots$

out

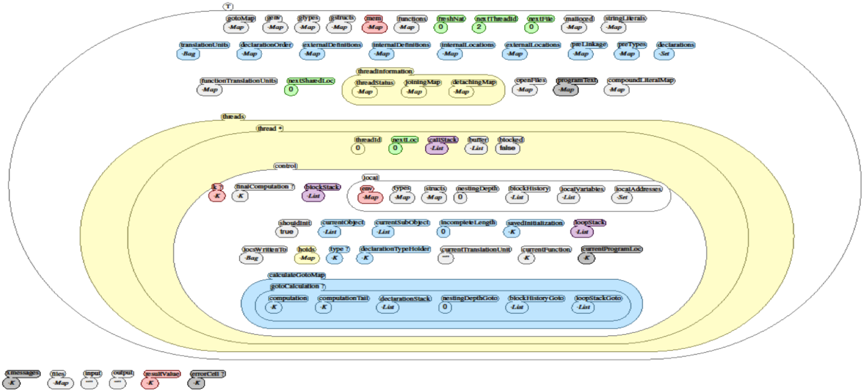
"$\ldots$"

Cink (with pointers and arrays) Configuration



C Configuration

MODULE C-CONFIGURATION
IMPORTS C-SYNTAX
IMPORTS COMMON-C-CONFIGURATION



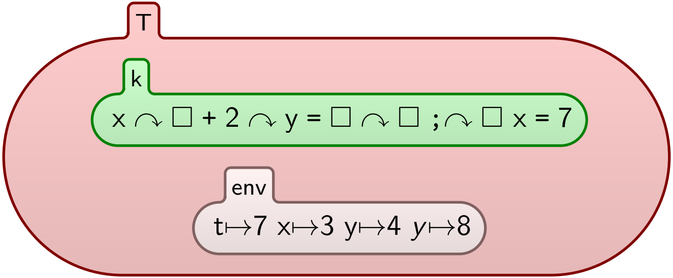
END MODULE

1

𝕂 rules: expressing natural language into rules

Reading from environment

If a local variable X is the next thing to be processed . . .
. . . and if X is mapped to a value V in the environment . . .
. . . then process X, replacing it by V



T

k

x ↷ ☐ + 2 ↷ y = ☐ ↷ ☐ ; ↷ ☐ x = 7

env

t↦7 x↦3 y↦4 *y*↦8

𝕂 rules: expressing natural language into rules

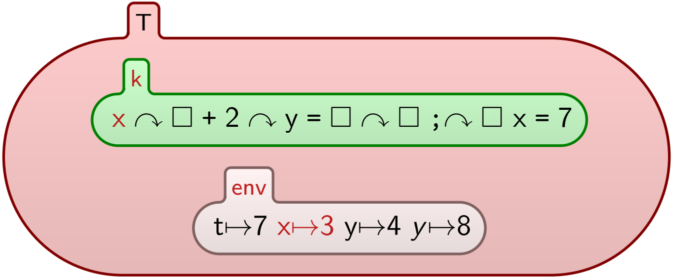
Focusing on the relevant part

Reading from environment

If a local variable X is the next thing to be processed . . .

. . . and if X is mapped to a value V in the environment . . .

. . . then process X, replacing it by V



$x \curvearrowright \square + 2 \curvearrowright y = \square \curvearrowright \square \,; \curvearrowright \square\ x = 7$

$t \mapsto 7\ \ x \mapsto 3\ \ y \mapsto 4\ \ y \mapsto 8$

𝕂 rules: expressing natural language into rules

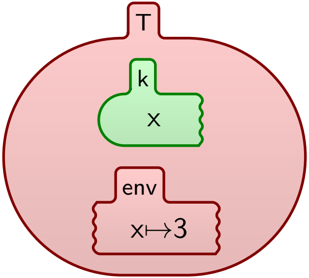
Unnecessary parts of the cells are abstracted away

Reading from environment

If a local variable X is the next thing to be processed …

…and if X is mapped to a value V in the environment …

…then process X, replacing it by V

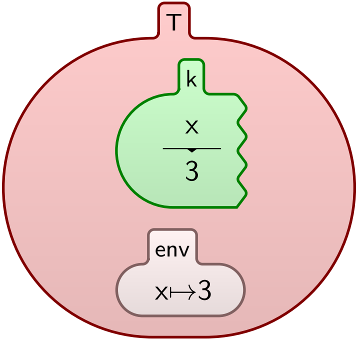


𝕂 rules: expressing natural language into rules

Underlining what to replace, writing the replacement under the line

Reading from environment

If a local variable *X* is the next thing to be processed . . .
. . . and if *X* is mapped to a value *V* in the environment . . .
. . . then process *X*, replacing it by *V*



𝕂 rules: expressing natural language into rules

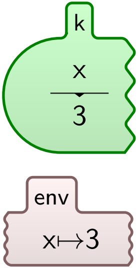
Configuration Abstraction: Keep only the relevant cells

Reading from environment

If a local variable X is the next thing to be processed …

… and if X is mapped to a value V in the environment …

… then process X, replacing it by V



𝕂 rules: expressing natural language into rules

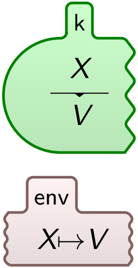
Generalize the concrete instance

Reading from environment

If a local variable X is the next thing to be processed …

… and if X is mapped to a value V in the environment …

… then process X, replacing it by V



𝕂 rules: expressing natural language into rules

Voilà!

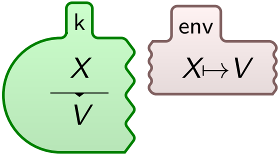
Reading from environment

If a local variable X is the next thing to be processed …

… and if X is mapped to a value V in the environment …

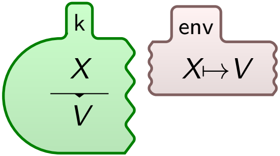
… then process X, replacing it by V

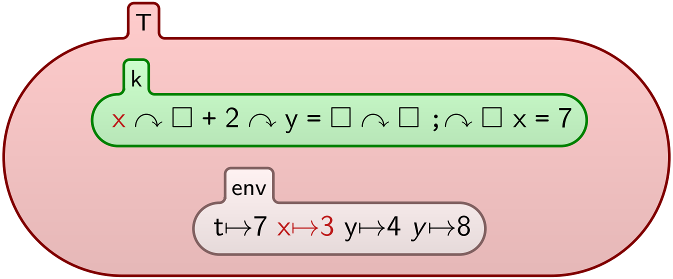
graphic ASCII



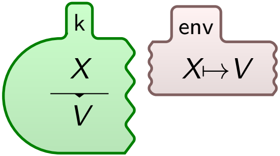
`rule <k>X => V<_/k> <env_> X ↦ V<_env>`

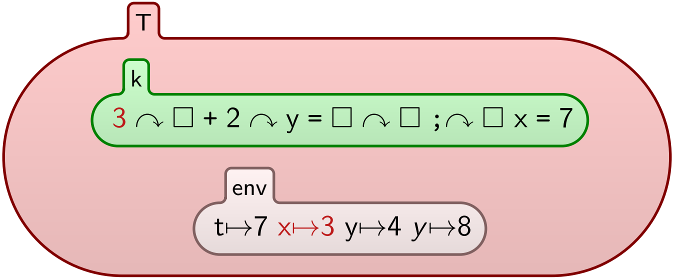
Rules at Work





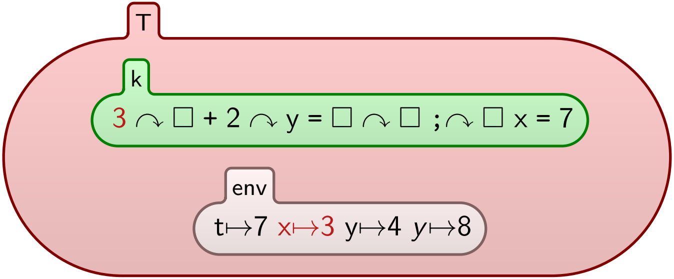
Rules at Work





Rules at Work

$$ERed + ERed \rightleftharpoons ERed \curvearrowright \square + ERed$$



T

k

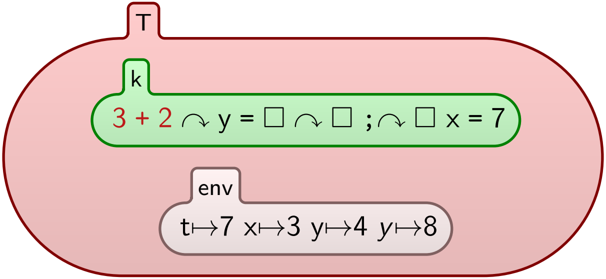
$3 \curvearrowright \square + 2 \curvearrowright \mathrm{y} = \square \curvearrowright \square\ ; \curvearrowright \square\ \mathrm{x} = 7$

env

$\mathrm{t} \longmapsto 7\ \mathrm{x} \longmapsto 3\ \mathrm{y} \longmapsto 4\ y \longmapsto 8$

Rules at Work

$$\mathit{ERed} + \mathit{ERed} \rightleftharpoons \mathit{ERed} \curvearrowright \square + \mathit{ERed}$$



T

k

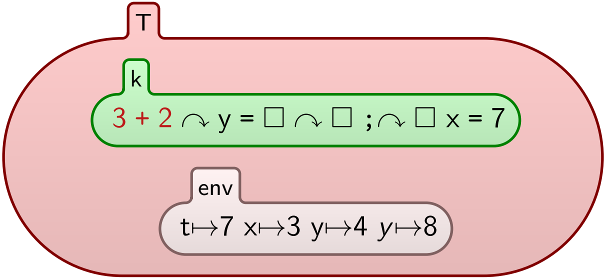
$3 + 2 \curvearrowright y = \square \curvearrowright \square\ ; \curvearrowright \square\ x = 7$

env

$t \longmapsto 7\ x \longmapsto 3\ y \longmapsto 4\ \mathit{y} \longmapsto 8$

Rules at Work

$I1 + I2 \rightharpoonup I1 +_{Int} I2$

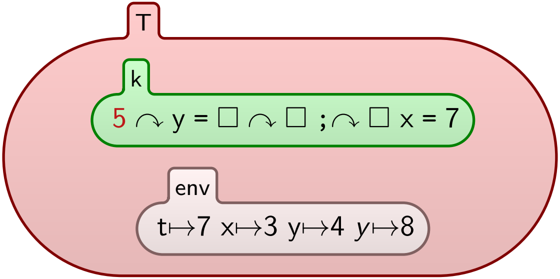


$3 + 2 \curvearrowright y = \square \curvearrowright \square\ ; \curvearrowright \square\ x = 7$

$\mathrm{t} \longmapsto 7\ \ \mathrm{x} \longmapsto 3\ \ \mathrm{y} \longmapsto 4\ \ y \longmapsto 8$

Rules at Work

$I1 + I2 \rightharpoonup I1 +_{Int} I2$



T

k

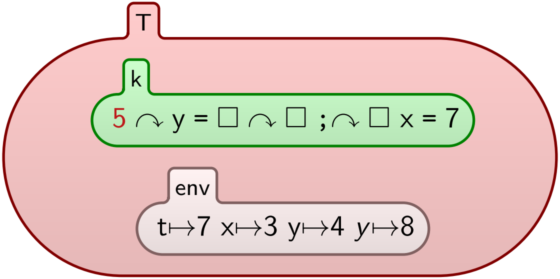
$\color{red}{5} \curvearrowright y = \square \curvearrowright \square \;; \curvearrowright \square\ x = 7$

env

$t \mapsto 7\ x \mapsto 3\ y \mapsto 4\ y \mapsto 8$

Rules at Work

$$E = ERed \quad \rightleftharpoons \quad ERed \curvearrowright E = \square$$



T

k

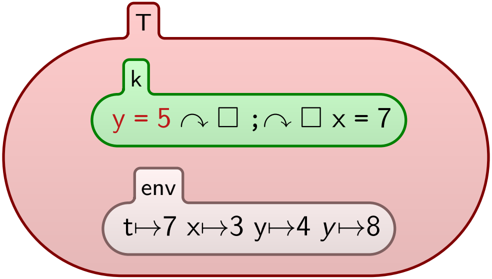
$\color{red}{5} \curvearrowright \mathrm{y} = \square \curvearrowright \square \; ; \curvearrowright \square \; \mathrm{x} = 7$

env

$\mathrm{t} \longmapsto 7 \;\; \mathrm{x} \longmapsto 3 \;\; \mathrm{y} \longmapsto 4 \;\; y \longmapsto 8$

Rules at Work

$$E = ERed \quad \rightleftharpoons \quad ERed \curvearrowright E = \square$$



T

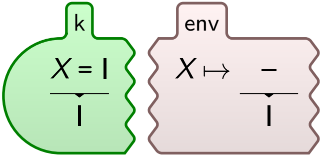
k

$\color{red}{y = 5} \curvearrowright \square \; ; \curvearrowright \square \; x = 7$

env

$t \mapsto 7 \;\; x \mapsto 3 \;\; y \mapsto 4 \;\; y \mapsto 8$

Rules at Work

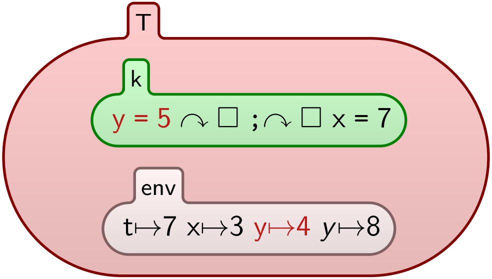


k

$$\frac{X = I}{I}$$

env

$$X \mapsto \frac{\_}{I}$$



T

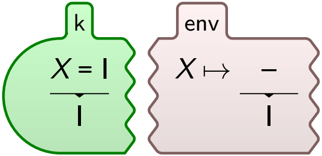
k

$y = 5 \curvearrowright \square \;;\curvearrowright \square\ x = 7$

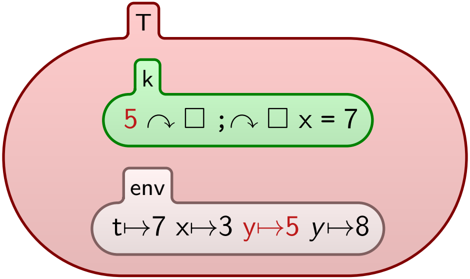
env

$t \mapsto 7\ \ x \mapsto 3\ \ y \mapsto 4\ \ y \mapsto 8$

Rules at Work



$$\frac{X = I}{I} \qquad X \mapsto \frac{\_}{I}$$



$$5 \curvearrowright \square \,; \curvearrowright \square\ x = 7$$

$$t \mapsto 7 \ \ x \mapsto 3 \ \ y \mapsto 5 \ \ y \mapsto 8$$

Program Logic in 𝕂

An ideal PL Framework should serve as a program logic
We can prove program correctnes in 𝕂 using Matching Logic (ML)
Matching Logic = K + FOL [G. Roșu, W. Schulte, C. Ellyson, A, Ștefănescu]

- Formulae = FOL over configurations, called patterns
  (Configurations are allowed to contain variables)
- Models = Ground configurations
- Satisfaction = Matching for configurations, plus FOL for the rest

MatchC = Matching Logic for a (kernel of) C

MatchC: Example of Annotated Program

```
struct listNode* reverse(struct listNode *x)
/*@ rule <k> $ => return p1; </k>
         <heap_> list(x)(A) => list(p1)(rev(A)) <_/heap> */
{
  struct listNode *p;
  p = 0;
  /*@ inv <heap_> list(p)(?B), list(x)(?C) <_/heap>
      ∧ A = rev(?B) @ ?C*/
  while(x) {
    struct listNode *y;
    y = x->next;
    x->next = p;
    p = x;
    x = y;
  }
  return p;
}
```

MatchC: Example of Annotated Program

```
void readWrite(int n)
/*@ rule <k> $ => return; </k>
         <in> A => epsilon <_/in>
         <out_> epsilon => A </out>
    if n = len(A) */
{
  /*@ inv <in> ?B <_/in> <out_> ?A </out>
          ∧ n >= 0 ∧ len(?B) = n ∧ A = ?A @ ?B */
  while (n) {
    int t;

    scanf("%d", &t);
    printf("%d ", t);
    n -= 1;
  }
}
```

Plan

1. Introduction
2. 𝕂 Framework
   - Motivation
   - K Framework Solution
   - Matching Logic
3. 𝕂ontributions
4. Conclusion

UAIC 𝕂ontributions 1/2

- Parsing (Radu Mereuță)
  a K definition uses both K syntax and PL syntax, hence a lot of ambiguities
- Contextual Transformers (Andrei Arusoaie)
  mapping rules on configurations according to Locality Principle
- Abstract Model Checking (Irina M. Asăvoae)
  collecting semantics, symbolic execution, model checking
  model-checking the object creation -based properties (joint work with F. de Boer, M. Bonsangue, J. Rot – CWI, LIACS)

UAIC 𝕂ontributions 2/2

- Worst Case Execution Time Analysis for embedded systems (Mihai Asăvoae)

  𝕂 semantics for hardware languages (SSRISC), abstraction and symbolic execution
- Modeling and Metamodeling Languages (joint work with Vlad Rusu, INRIA Lille)
- Automatic Instantiation of the Patterns from Annotations (Elena naum)

Plan

Conclusion

Formal semantics is useful and practical!

One can use an executable semantics of a language as is also for program verification

Giving a formal semantics is not necessarily painful, it can be fun if one uses the right tools

$\mathbb{K}$ Framework is scalable (C, Scheme, Verilog, Java etc)