Getting started with the $\mathbb{K}$ Framework

Precise Modeling and Analysis group
University of Oslo
Daniel Fava

(borrowing material from Grigore Roșu's slides)

February 24, 2017

What is it?

The $\mathbb{K}$ framework is

- an executable semantic framework
- based on rewriting logic
- used to define
  - programming languages
  - type systems and
  - formal analysis tools

Disclaimer: this is not an introduction to rewriting
Goal: to help you get to know the tool and get started

After defining a programming language $\mathcal{L}$ in $\mathbb{K}$, you get:

- A parser and compiler for the language
- An interpreter for programs in the language
- Facilities to perform model checking
  - E.g., Is state X reachable when running a non-deterministic program P of language $\mathcal{L}$?
- Facilities for exporting the language definition into Coq

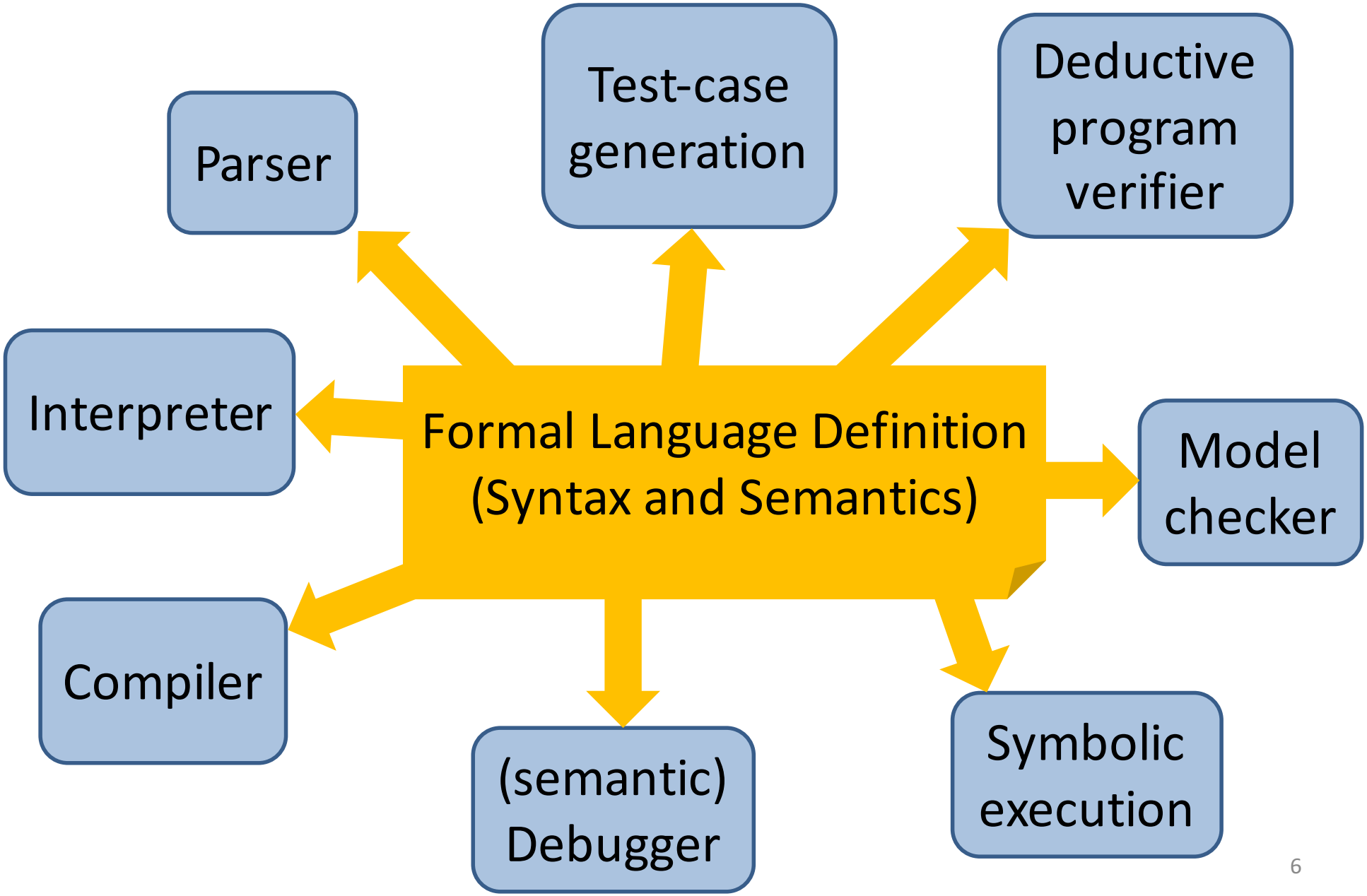
Motivation

Shortcomings of existing frameworks

- Hard to deal with control (except evaluation contexts)
  - halt, break/conInue, exceplons, callcc
- Nonmodular (except Modular SOS)
  - Adding new features require changing unrelated rules
- Lack of semantics for true concurrency (except CHAM)
  - BigStep 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 BigStep SOS)

[rosu-serbanuta-2010-jlap-slides-2011-01-14-Iasi]

Vision and objective of $\mathbb{K}$



[Roşu 2015-06-29-RTA]

Short history of $\mathbb{K}$

Grigore Roșu and José Meseguer, 2003/2004

- Project to define the semantics of a programming language as a rewrite theory
- Showed that most executable semantics approaches can be framed as rewrite logic semantics (Modular/SmallStep/BigStep SOS, evaluation contexts, continuation-based, etc.)

Evolved into the $\mathbb{K}$ tool currently being developed at

- University of Illinois Urbana-Champaign
- Runtime Verification Inc.
- Alexandru Ioan Cuza University (UAIC), Romania

Theoretical underpinnings: Matching Logic [RTA'15]

Successes

Many programming languages's semantics have been defined in $\mathbb{K}$

- Java, the Java Virtual Machine, LLVM, OCaml, Python, etc

"The most complete formal C semantics"

Tested on thousands of C programs (several benchmarks, including the gcc torture test, code from the obfuscated C competition, etc.)

- Passed 99.2% so far
- GCC 4.1.2 passes 99%, ICC 99.4%, Clang 98.3% (no opt.)

[POPL12, PLDI15]

```
1  module EXP-SYNTAX

3    //@ Arithmetic Syntax
4    syntax Exp ::= Int
5                 | "(" Exp ")" [bracket]
6                 | Exp "+" Exp [seqstrict] //addition
7                 | Exp "*" Exp [seqstrict] //multiplication
8                 | Exp "/" Exp [seqstrict] //division
9                 | Exp "?" Exp ":" Exp [strict(1)]
10                | Exp ";" Exp [seqstrict]

12   //@ Input / Output Syntax

14   syntax Exp ::= "read"
15                | "print" "(" Exp ")" [strict]


18   //@ Concurrency features
19   syntax Exp ::= "spawn" Exp
20                | "rendezvous" Exp [strict]
21 end module

23 module EXP
24   imports EXP-SYNTAX
25   syntax KResult ::= Int
26   configuration
27     <k color="green" multiplicity="*"> $PGM:K </k>
28     <streams>
29       <in color="magenta" stream="stdin"> .List </in>
30       <out color="Fuchsia" stream="stdout"> .List </out>
31     </streams>

33  //@ Arithmetic Semantics

35   rule I1:Int + I2:Int
36     => I1 +Int I2

38   rule I1:Int * I2:Int
39     => I1 *Int I2

41   rule I1:Int / I2:Int => I1 /Int I2
42                                 requires I2 =/=Int 0
```

MODULE EXP-SYNTAX

Arithmetic Syntax

SYNTAX $Exp ::= Int$
$\mid (Exp)$ [bracket]
$\mid Exp + Exp$ [strict]
$\mid Exp * Exp$ [strict]
$\mid Exp / Exp$ [strict]
$\mid Exp \texttt{ ? } Exp \texttt{ : } Exp$ [strict(1)]
$\mid Exp \texttt{ ; } Exp$ [seqstrict]

Input / Output Syntax

SYNTAX $Exp ::= \texttt{read}$
$\mid \texttt{print } (Exp)$ [strict]

Concurrency features

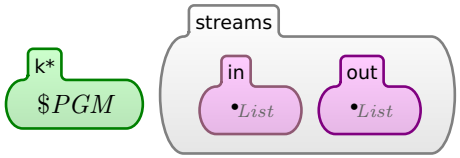
SYNTAX $Exp ::= \texttt{spawn } Exp$
$\mid \texttt{rendezvous } Exp$ [strict]

END MODULE

MODULE EXP

SYNTAX $KResult ::= Int$

CONFIGURATION:



Arithmetic Semantics

RULE $\dfrac{I1 + I2}{I1 +_{Int} I2}$

RULE $\dfrac{I1 * I2}{I1 *_{Int} I2}$

RULE $\dfrac{I1 / I2}{I1 /_{Int} I2}$ requires $I2 =/=_{Int} 0$

Some features

- Built-in types
  - Bool, Int, Float, String, Id, List, Set, Map, ...
- Support for literate programming
  - Embed latex annotations into language definition
- Support for generating documentation
  - .tex, .pdf

$\mathbb{K}$ in a nutshell

$\mathbb{K}$omputations

- Sequences of tasks
- Capture the sequential fragment of programming languages

$\mathbb{K}$onfigurations

- Bags of nested cells (XML/HTML like syntax)
- Modularity

$\mathbb{K}$ rules

- Precisely identify changes
- More concise and concurrent than regular rewrite rules

[rosu-serbanuta-2010-jlap-slides-2011-01-14-Iasi]

Demo

Getting started, Tutorial

Tutorial videos (around 6 minutes each)
with written transcript and code
`http://www.kframework.org/index.php/K_Tutorial`

- Extremely useful and (relatively) easy to follow
- Shows how to implement a store, input/output, closures, etc

- Define Lambda, a call-by-value variant of lambda calculus
- Define IMP, a C-like imperative language
- Define Lambda++, adds call/cc to Lambda
- etc

Getting started, Other help

- Papers and Tutorials
  `http://fsl.cs.illinois.edu/index.php/FSL_Publications`
  - The K primer (version 3.3)
- Mailing list
  `https://lists.cs.illinois.edu/lists/arc/k-user`
  - Frequently read and answered by the developers
- Semantics of
  - C, LLVM, Java, JVM, OCaml, Python, etc

  Code available on the git repo

Getting started, Versions

- Tutorial videos based on $\mathbb{K}$ version 3.4,
  but v3.4 is no longer supported
- Latest release is 4.0, but missing some features
  such as the latex back-end and pretty printing

- Can get pre-compiled binaries or build from source
  `https://github.com/kframework`

- It is ok to start with the precompiled binaries
  But I suggest cloning the git repo and build from source

Getting started, Installation from source

Instructions from `https://github.com/kframework/k`

1. Apache Maven
   - Check if it is installed by typing `mvn -v` on the command line
   - If not installed, can use a package manager
   - I use brew on the Mac [Can install brew on IFI machines even without sudo]
2. Java JDK (version 8u45 or higher)
3. `git clone https://github.com/kframework/k`
4. Add an environment variable and edit `$PATH`
   - Add to `$PATH` <PATH_TO_K>`/k/k-distribution/target/release/k/bin`
   - Add environment variable `MAVEN_OPTS` and set it to `-XX:+TieredCompilation`
5. cd into the k directory and run the following commands
   - `mvn package` (can take about 5 1/2 minutes)
   - `mvn verify` (can take about half hour)

Thank you