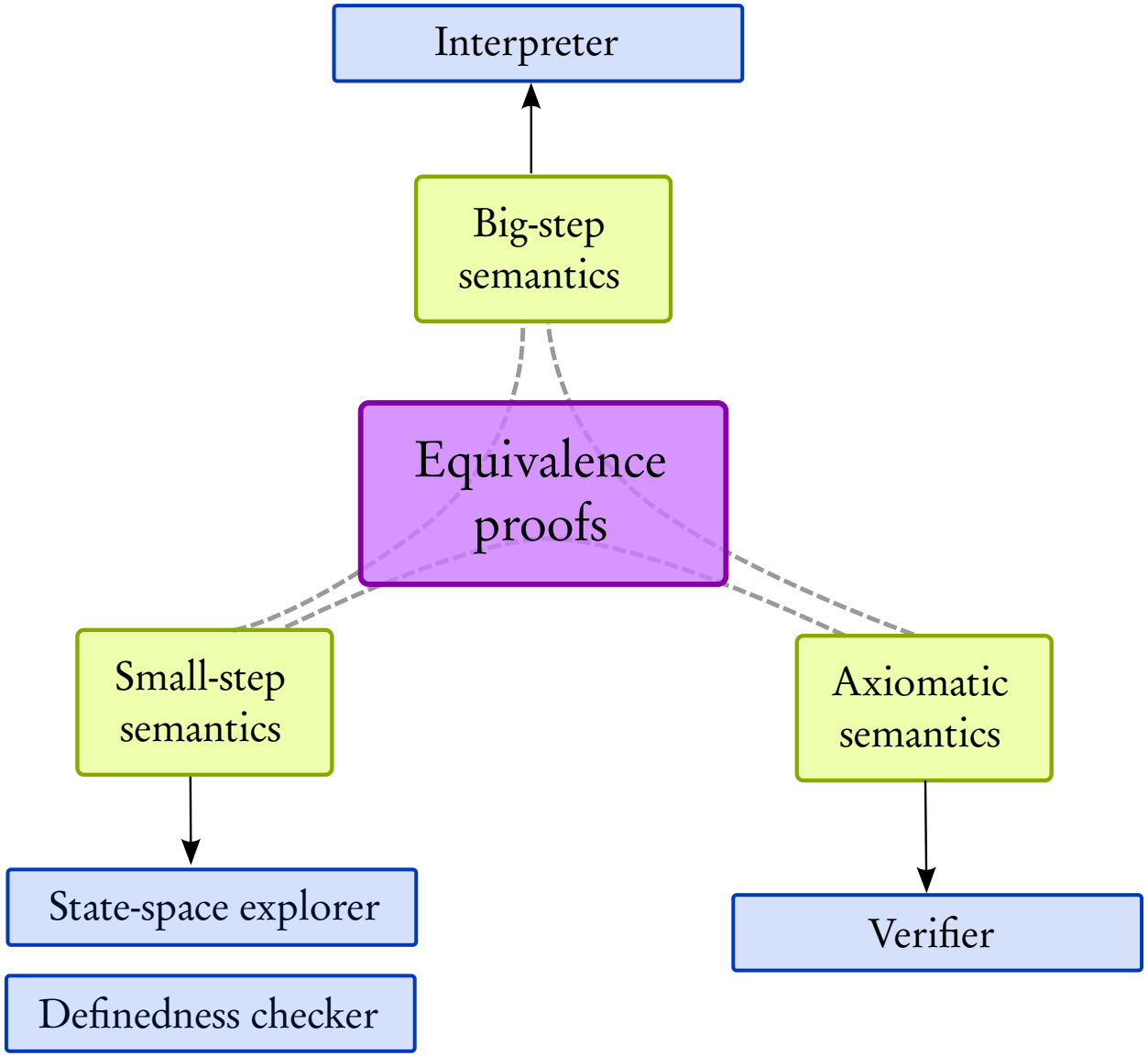
Executing Formal Semantics with the $\mathbb{K}$ Tool

David Lazar[1] Andrei Arusoaie[2] Traian Șerbănuță[1,2]
Chucky Ellison[1] Radu Mereuta[2] Dorel Lucanu[2]
Grigore Roșu[1,2]

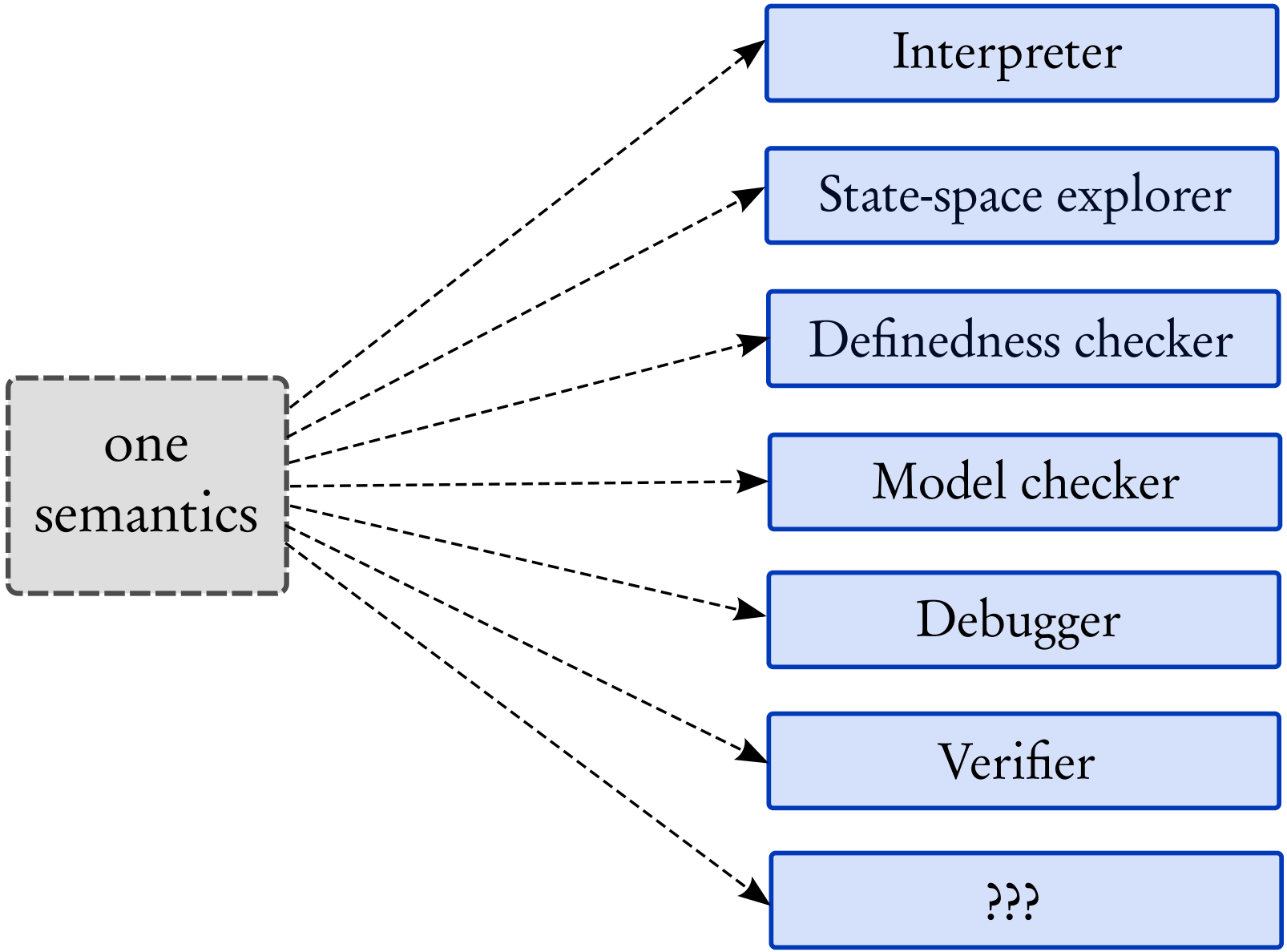
[1]University of Illinois at Urbana-Champaign
[2]University Alexandru Ioan Cuza of Iași

FM 2012

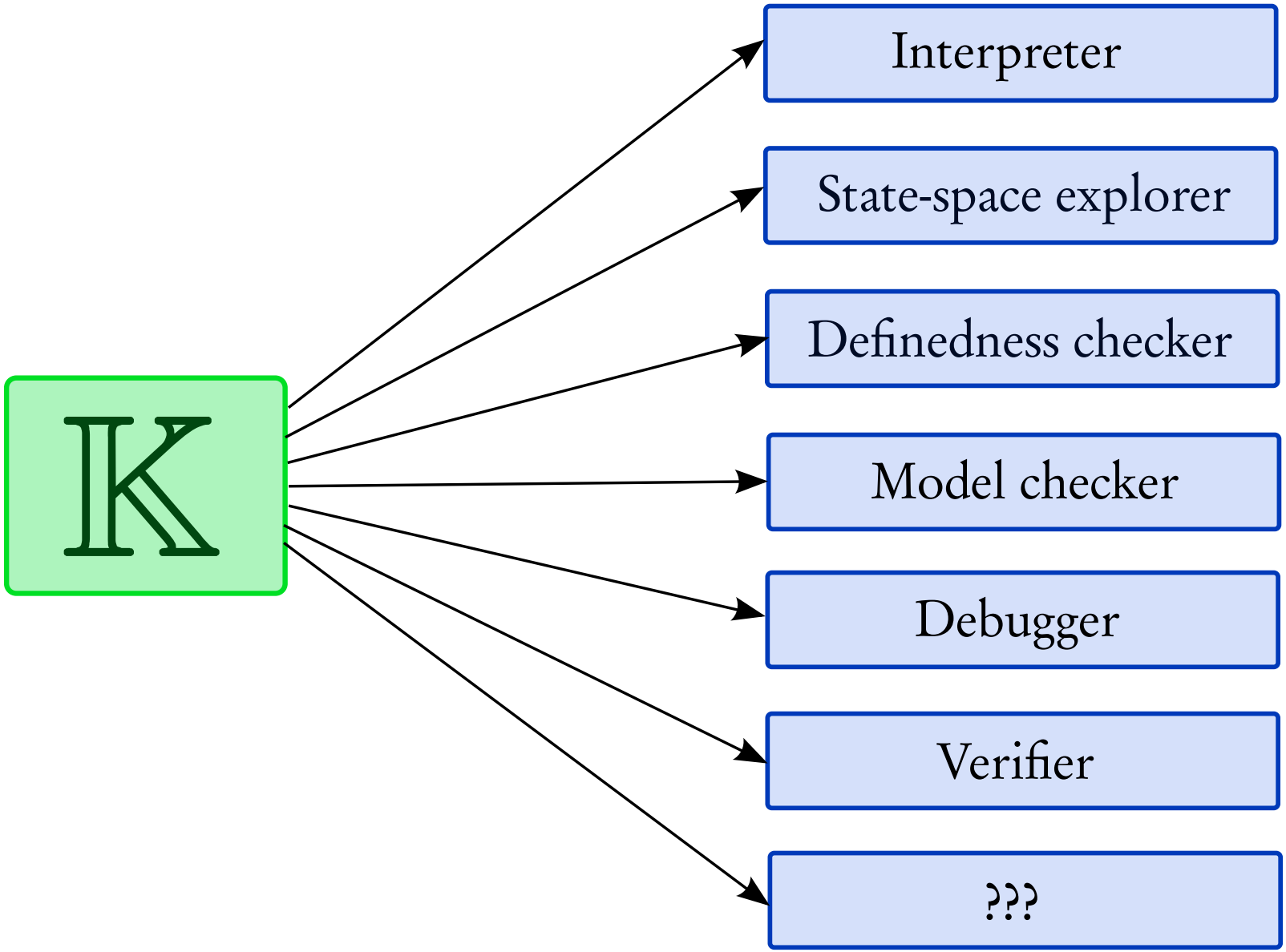
Semantics-based Tools



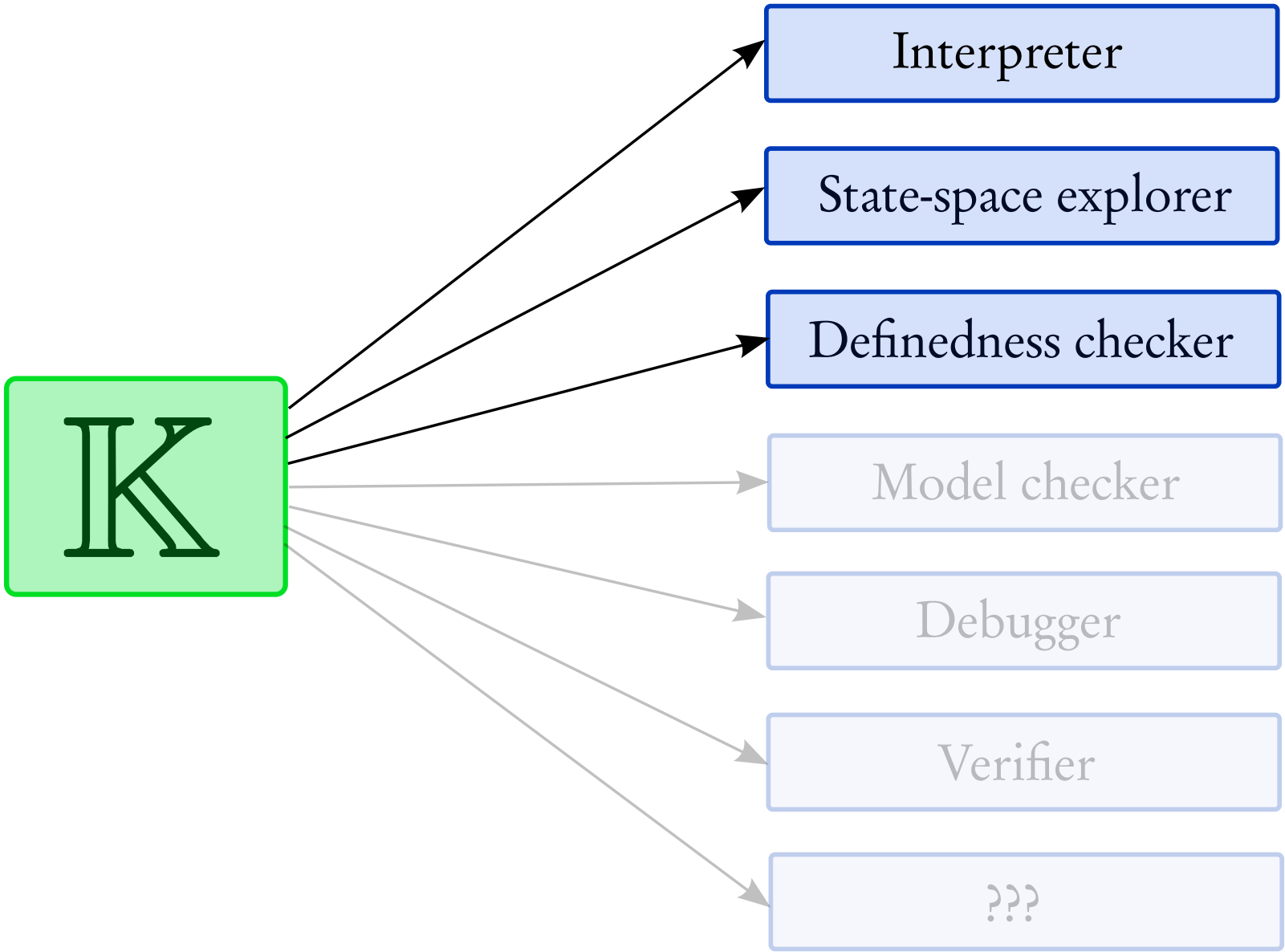
The Goal: Many Tools, One Semantics



A Solution: The $\mathbb{K}$ Framework



We Will Focus on ...



The **EXP** Language

Integer Arithmetic

```
5 + 3/2
```

Variables

```
x + y
```

for simplicity, variable lookup only

Reading from STDIN

```
read
```

Writing to STDOUT

```
print(x)
```

The **EXP** Language

Integer Arithmetic

```
5 + 3/2
```

Variables

```
x + y
```

for simplicity, variable lookup only

Reading from STDIN

```
read
```

Writing to STDOUT

```
print(x)
```

The $\mathbb{K}$ Definition of **EXP**

5 rules, one for each construct above

MODULE EXP

CONFIGURATION

$$\langle\ \$PGM\ \rangle_{\mathsf{k}}\ \langle\ \$STATE\ \rangle_{\mathsf{state}}\quad \langle\ \langle\ \cdot\ \rangle_{\mathsf{in}}\ \langle\ \cdot\ \rangle_{\mathsf{out}}\ \rangle_{\mathsf{streams}}$$

SYNTAX $KResult ::= Int$

SYNTAX $K ::= K + K\ [\text{strict}] \mid K\ /\ K\ [\text{strict}]$

RULE $I_1 + I_2 \Rightarrow I_1 +_{Int} I_2$

RULE $I_1\ /\ I_2 \Rightarrow I_1 \div_{Int} I_2 \quad \text{when} \quad I_2 \neq_{Int} 0$

SYNTAX $K ::= Id$

RULE $\langle\ \underbrace{X}_{I}\ \cdots\rangle_{\mathsf{k}}\ \langle\cdots\ X \mapsto I\ \cdots\rangle_{\mathsf{state}}$

SYNTAX $K ::= \texttt{read} \mid \texttt{print}\ K\ [\text{strict}]$

RULE $\langle\underbrace{\texttt{read}}_{I}\ \cdots\rangle_{\mathsf{k}}\ \langle\ \underbrace{I}_{\cdot}\ \cdots\rangle_{\mathsf{in}}$

RULE $\langle\underbrace{\texttt{print}\ I}_{I}\ \cdots\rangle_{\mathsf{k}}\ \langle\cdots\ \underbrace{\cdot}_{I}\ \rangle_{\mathsf{out}}$

END MODULE

Interpreter

average.exp

```
print((read + read + read) / 3)
```

Interpreter

average.exp

```
print((read + read + read) / 3)
```

Running the Program

```
$ echo "3 14 15" | krun average.exp
10
```

Definedness Checker

`div.exp`

```
print(42 / read)
```

Definedness Checker

div.exp

```
print(42 / read)
```

Defined Execution

```
$ echo "2" | krun div.exp
21
```

Definedness Checker

div.exp

```
print(42 / read)
```

Defined Execution

```
$ echo "2" | krun div.exp
21
```

Undefined Execution

```
$ echo "0" | krun div.exp
<k>
  42 / 0 ~> print □
</k>
```

State-space Explorer

`div-nondet.exp`

```
print(read / read)
```

State-space Explorer

`div-nondet.exp`

```
print(read / read)
```

Note

Evaluation order of / is nondeterministic!

State-space Explorer

div-nondet.exp

```
print(read / read)
```

Run it Normally

```
$ echo "7 0" | krun div-nondet.exp
0
```

Right-to-left evaluation order picked arbitrarily!

State-space Explorer

div-nondet.exp

```
print(read / read)
```

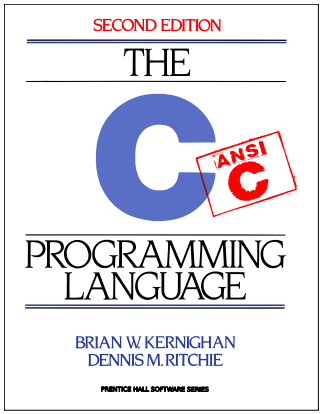
Search for all Possibilities

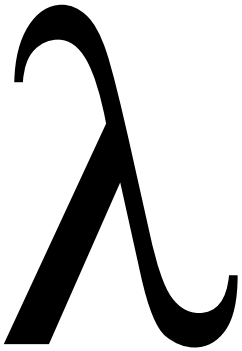
```
$ echo "7 0" | krun div-nondet.exp --search
Search results:
```

```
Solution 1, state 2:
<k>
  0
</k>
```

```
Solution 2, state 3:
<k>
  7 / 0 ~> print □
</k>
```

C, Scheme, LLVM, JavaScript, OCaml, Python, Haskell, ...





The $\mathbb{K}$ Definition of C

- 1200 rules
- `kcc`, similar to `krun` but feels like `gcc`
- `http://c-semantics.googlecode.com`

Tiny C Program

eval_order.c

```
int denominator = 5;

int setDenominator(int d) {
    return denominator = d;
}

int main(void) {
    return setDenominator(0) + (7 / denominator);
}
```

Bugs are Looming

```
$ clang -O0 eval_order.c && ./a.out
Floating point exception

$ clang -O2 eval_order.c && ./a.out
$
```

Find Bugs using Search

```
$ kcc eval_order.c
$ SEARCH=1 ./a.out
```

Find Bugs using Search

```
$ kcc eval_order.c
$ SEARCH=1 ./a.out

2 solutions found
------------------------------------------------------------
Solution 1
Program got stuck
File: eval_order.c
Line: 8
Description: Division by 0.
------------------------------------------------------------
Solution 2
Program completed successfully
Return value: 1
```

expressive

modular

concurrent

The $\mathbb{K}$ Framework

easy

practical

`http://k-framework.org`

executable

scalable

analyzable