A Matching Logic Theory of Contexts with Applications to $\mathbb{K}$ (Work in Progress)

Xiaohong Chen[1] and Horațiu Cheval[2] and Dorel Lucanu[1,3] and Grigore Roșu[1,4]

[1]Pi Squared Inc [2]University of Bucharest [3]Alexandru Ioan Cuza University of Iași
[4]University of Illinois at Urbana-Champaign

FROM, September 17, 2025

Plan

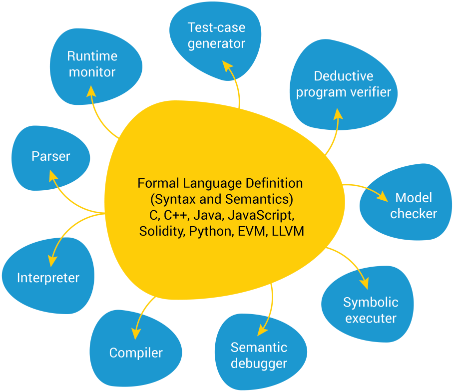
A Brief History of $\mathbb{K}$ Framework

- 2003, Grigore Roșu at UIUC: motivated mainly by teaching programming languages and noticing that the existing semantic frameworks have limitations
- 2010-2013: 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
- since 2014: joint work between FSL and Runtimeverification - a start-up founded by Grigore Roșu
- since 2024: joint work with Pi Squared Inc - a second start-up founded by Grigore Roșu

K Framework: The Main Idea

$\mathbb{K}$ (`https://kframework.org/`) is a framework where

- programming languages can be formally defined, and
- tools can be soundly derived from the formal language definition



Example: IMP (Partial)

```
module IMP-SYNTAX
  imports DOMAINS-SYNTAX
  syntax AExp  ::= Int | Id
                 > AExp "+" AExp              [left, strict]
                 ...
  syntax Stmt  ::= Block
                 | Id "=" AExp ";"            [strict(2)]
                 ...
  syntax Pgm ::= "int" Ids ";" Stmt
endmodule
module IMP-CONFIG
  imports IMP-SYNTAX
  imports DOMAINS
  configuration <T color="yellow">
                  <k> $PGM:Pgm </k>
                  <state> .Map </state>
                </T>
endmodule
module IMP
  imports IMP-CONFIG
  imports VERIFICATION

  syntax KResult ::= Int | Bool
  ..
  rule I1 + I2 => I1 +Int I2
  ...
  rule <k> X = I:Int; => .K ...</k> <state>... X |-> (_ => I) ...</state>
  ...
endmodule
```

Example of IMP Program

sum.imp

```
// This program calculates in sum
// the sum of numbers from 1 to n.

int n, sum;
n = 100;
sum = 0;
while (!(n <= 0)) {
  sum = sum + n;
  n = n + -1;
}
```

Runing sum.imp

```
% cd imp/
% kompile imp.k
% krun sum.imp
% krun sum.imp
<T>
  <k>
    .K
  </k>
  <state>
    n |-> 0
    sum |-> 5050
  </state>
```

Example: IMP with Threads

```
module IMP-SYNTAX
  // the same
endmodule

module IMP-CONFIG
  imports IMP-SYNTAX
  imports DOMAINS
  configuration
    <T color="yellow">
      <threads>
        <thread multiplicity="*" type="Map" initial="">
          <id> 0 </id>
          <k> $PGM:K </k>
        </thread>
      </threads>
      <state> .Map </state>
      <next-id> 1 </next-id>
    </T>
endmodule

module IMP
  // the same
endmodule
```

Example of IMP Program with Threads

sum.imp

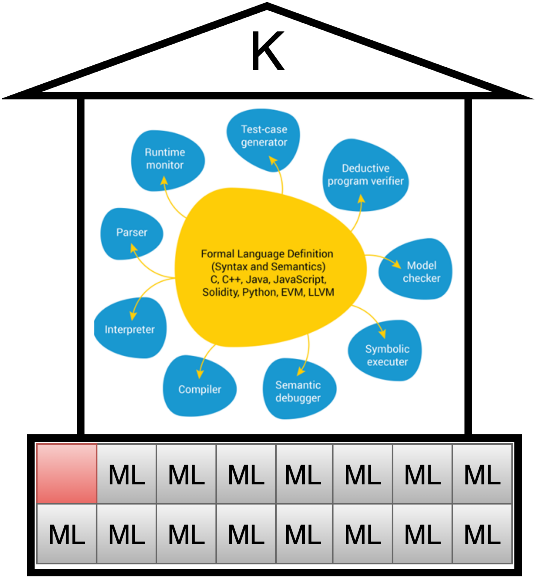
```
// This program calculates in sum
// the sum of numbers from 1 to n.

int n, sum;
n = 100;
sum = 0;
while (!(n <= 0)) {
  sum = sum + n;
  n = n + -1;
}
```

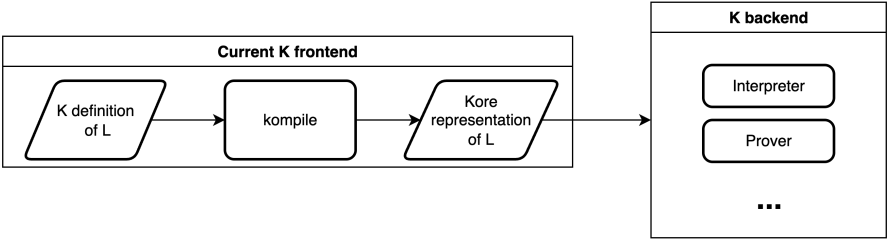
Runing sum.imp

```
cd ../imp++
% kompile imp.k
% krun sum.imp
<T>
  <threads>
    <thread>
      <id>
        0
      </id>
      <k>
        .K
      </k>
    </thread>
  </threads>
  <state>
    n |-> 0
    sum |-> 5050
  </state>
  <next-id>
    1
  </next-id>
</T>
```

The Foudation of $\mathbb{K}$ is Matching Logic ($\mathbb{ML}$)



$\mathbb{K}$ Framework: Frontend and Backend



Kore for rule `I1 + I2 => I1 +Int I2` in IMP

```
axiom{} \rewrites{TopCell{}} (
  \and{TopCell{}} (
    <Top>(<T>(<k>(kseq{}(inj{AExp{}, KItem{}}(
      _+_(inj{Int{}, AExp{}}(VarI1:Int{}),inj{Int{}, AExp{}}(VarI2:Int{}))),
      DotVar2:K{})),
      DotVar1:StateCell{}),
      DotVar0:GeneratedCounterCell{}),
    \top{TopCell{}}()),
  \and{TopCell{}} (
    <Top>(<T>(<k>(kseq{}(inj{Int{}, KItem{}}(_
        +Int_(VarI1:Int{},VarI2:Int{})),DotVar2:K{})),
      DotVar1:StateCell{}),
      DotVar0:GeneratedCounterCell{}),
    \top{TopCell{}}()
  )
)
```

Kore for rule `I1 + I2 => I1 +Int I2` in IMP with Threads

```
axiom{} \rewrites{TopCell{}} (
  \and{TopCell{}} (
    <Top>(<T>(<treads>(
        ThreadCellMap{}(ThreadCellMapItem{}(DotVar3:IdCell{},
        <thread>(DotVar3:IdCell{},
          <k>(kseq{}(inj{AExp{}, KItem{}}(_+_(inj{Int{}, AExp{}}(VarI1:Int{}),
            inj{Int{}, AExp{}}(VarI2:Int{}))),DotVar4:K{})))),
        DotVar2:ThreadCellMap{})),
        Gen0:StateCell{},Gen1:NextIdCell{}),
      DotVar0:GeneratedCounterCell{}),
    \top{TopCell{}}()),
  \and{TopCell{}} (
    <Top>(<T>(<treads>(
        ThreadCellMap{}(ThreadCellMapItem{}(DotVar3:IdCell{},
          <thread>(DotVar3:IdCell{},
            <k>(kseq{}(inj{Int{}, KItem{}}(_+Int_(VarI1:Int{},VarI2:Int{})),
              DotVar4:K{})))),
          DotVar2:ThreadCellMap{})),
          Gen0:StateCell{},Gen1:NextIdCell{}),
        DotVar0:GeneratedCounterCell{}),
    \top{TopCell{}}()
  )
)
```

Main Questions

Q1 What is the $\mathbb{ML}$ denotation of rules like

```
rule I1 + I2 => I1 +Int I2
```

```

rule <k> X = I:Int; => .K ...</k> <state>... X |-> (_ => I) ...</state>
```

Q2 How such a simple rule, like the first one, can handle any $E_1 + E_2$ expression?

Plan

1. Introduction
2. Matching Logic (ML)
3. Contexts
4. Application to IMP
5. Conclusion

A Brief History of ML

- An alternative to Hoare/Floyd Logic (Roșu, Ellison, Schulte, AMAST 2010)
- Reachability Logic (A. Stefanescu, St. Ciobaca, B. M. Moore, T.-F. Serbanuta, R. Mereuta, G. Rosu, LICS 2013, RTA-TLCA 2014, LMCS 2029)
- (Many-sorted) Matching Logic (Roșu, LMCS 2017)
- Matching mu-Logic (Chen, Roșu, LICS 2019)
- Applicative Matching Logic (Chen, Roșu, TR 2019; Chen, Roșu, Lucanu, JLAMP 2021)

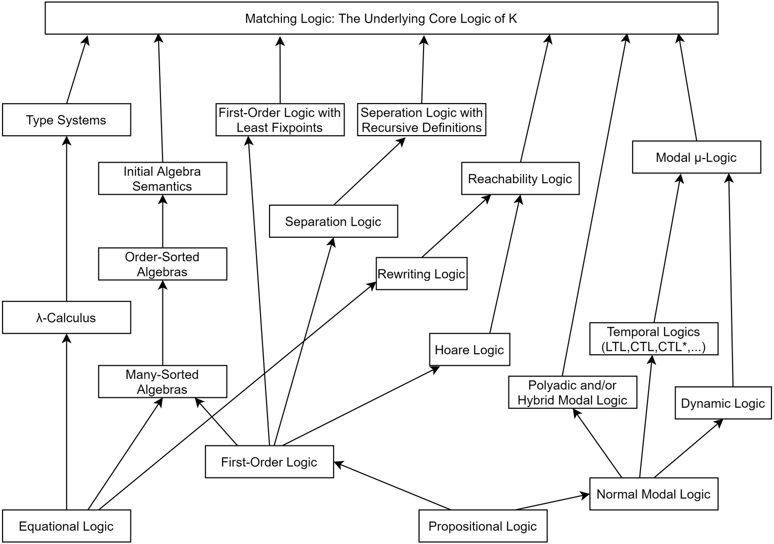
ML: Rationale Behind

A minimal logic where

- definition of programming languages and
- behavioral properties of their programs

can uniformly specified.

ML is Expressive[1]



[1] Source: http://www.matching-logic.org/

$\mathbb{ML}$: Syntax

Signature: (Σ, EV, SV),
where Σ is a set of *constant symbols*, EV a set of *element variables*, and SV. a set of *set variables*

Formulas (Patterns):

$\varphi ::= x$	elementary variable ($x \in EV$)
$\mid X$	set variabile ($X \in SV$)
$\mid \sigma$	symbol ($\sigma \in \Sigma$)
$\mid \varphi_1\ \varphi_2$	application
$\mid \bot$	bottom
$\mid \varphi_1 \to \varphi_2$	implication
$\mid \exists x.\varphi$	existential binder
$\mid \mu X.\varphi$ *if* φ *is positive in* X	least fixpoint binder

$\mathbb{ML}$: Semantics - Models

$(M, \_\bullet\_, \{M_\sigma\}_{\sigma \in \Sigma})$, where

- M is a carrier set, required to be nonempty;
- $\_\bullet\_ : M \times M \to \mathcal{P}(M)$ is a function, called the *interpretation of application*; here, $\mathcal{P}(M)$ is the powerset of M;
- $M_\sigma \subseteq M$ is a subset of M, called the *interpretation of* σ *in* M for each $\sigma \in \Sigma$.

$\mathbb{ML}$: Semantics - Pattern Interpretation

M-valuation: $\rho\colon (EV \cup SV) \to (M \cup \mathcal{P}(M))$ s.t.
$\rho(x) \in M$ for all $x \in EV$ and $\rho(X) \subseteq M$ for all $X \in SV$.

pattern interpretation: $|\_|_\rho\colon \textsc{Pattern} \to \mathcal{P}(M)$

$$
\begin{aligned}
|x|_\rho &= \{\rho(x)\} \\
|X|_\rho &= \rho(X) \\
|\sigma|_\rho &= M_\sigma \\
|\bot|_\rho &= \emptyset \\
|\varphi_1\, \varphi_2|_\rho &= |\varphi_1|_\rho \bullet |\varphi_2|_\rho \\
|\varphi_1 \to \varphi_2|_\rho &= M \setminus (|\varphi_1|_\rho \setminus |\varphi_2|_\rho) \\
|\exists x.\, \varphi|_\rho &= \bigcup_{a \in M} |\varphi|_{\rho[a/x]} \\
|\mu X.\, \varphi|_\rho &= \mu \mathcal{F}^\rho_{X,\varphi}
\end{aligned}
$$

$\mathbb{ML}$: Theory of Sorts (Over Theory of Equality)

If $s \in \Sigma$ represents a sort name, then the pattern (inh s) represents all its inhabitants, where inh $\in \Sigma$.

- New formulas (patterns):

$$\varphi ::= \top_s \mid \forall x{:}s.\, \varphi \mid \exists x{:}s.\, \varphi \mid \varphi{:}s \mid \forall x_1, \ldots, x_n{:}s.\, \varphi \mid \exists x_1, \ldots, x_n{:}s.\, \varphi$$

- Axioms:

$$\begin{gathered} \mathsf{Sort} \in \top_{\mathsf{Sort}} \\ \forall s. s \in \top_{\mathsf{Sort}} \to \lceil \top_s \rceil \end{gathered}$$

- Notations:

$$\begin{aligned}
\top_s &:\leftrightarrow \mathit{inh}\ s && \text{/* inhabitants of } s \text{ */} \\
\forall x{:}s.\, \varphi &:\leftrightarrow \forall x.\, x \in \top_s \to \varphi && \text{/* } \forall \text{ within sort } s \text{ */} \\
\exists x{:}s.\, \varphi &:\leftrightarrow \exists x.\, x \in \top_s \wedge \varphi && \text{/* } \exists \text{ within sort } s \text{ */} \\
\forall x_1, \ldots, x_n{:}s.\, \varphi &:\leftrightarrow \forall x_1{:}s.\, \ldots \forall x_n{:}s.\, \varphi && \text{/* nested } \forall \text{ within sort } s \text{ */} \\
\exists x_1, \ldots, x_n{:}s.\, \varphi &:\leftrightarrow \exists x_1{:}s.\, \ldots \exists x_n{:}s.\, \varphi && \text{/* nested } \exists \text{ within sort } s \text{ */}
\end{aligned}$$

Power Sorts

Given a sort s ($s \in \top_{\mathsf{Sort}}$), its *power sort* is specified by

- a sort 2^s

$$2^s \in \top_{\mathsf{Sort}}$$

- two constant symbols, extension and intension in Σ, together with the following axioms:

$$\forall \alpha{:}2^s.\text{extension } \alpha \subseteq \top_s$$
$$X \subseteq \top_s \to \exists \alpha{:}2^s.\text{extension } \alpha = X$$
$$\forall \alpha{:}2^s.\forall \beta{:}2^s.\text{extension } \alpha = \text{extension } \beta \to \alpha = \beta$$
$$\text{intension } \varphi :\leftrightarrow \exists \alpha{:}2^s.\alpha \wedge (\text{extension } \alpha = \varphi)$$

Remark

1. The product sort $s_1 \otimes s_2$ can also be specified.
2. The function sort $[s_1 \to s_2]$ can be specified a subsort of $2^{s_1 \otimes s_2}$.

Plan

Contexts, Intuitively

A context-based processing has two steps:

split: a term t is split into two components: a subterm t_0 and a *context* $C[\,]$, including a special variable $\square$ usually named *hole*, such that $t = C[t_0/\square]$;

plug: given a context $C[\,]$ and a term $\bar{t}_0$, returns $C[\bar{t}_0] = C[\bar{t}_0/\square]$.

Contexts: Definition 1/2

- given the sorts s_1 and s_2 ($s_1, s_2 \in \top_{\mathsf{Sort}}$), consider a new sort $\mathsf{Context}_{s_1}^{s_2}$

$$\mathsf{Context}_{s_1}^{s_2} \in \top_{\mathsf{Sort}}$$

- a constant symbol gamma in Σ, used for abstraction;
- a constant symbol plug in Σ, used for plugging operation;
- the following notations:

$$\gamma\square{:}s_1.\varphi :\leftrightarrow \mathsf{gamma}([\square{:}s_1]\varphi) \qquad /*\ \text{abstraction}\ */ \qquad \text{(Ntn.1)}$$

$$C[x] :\leftrightarrow \mathsf{plug}(C, x) \qquad /*\ \text{plugging}\ */ \qquad \text{(Ntn.2)}$$

where $C{:}\mathsf{Context}_{s_1}^{s_2}$, $x{:}s_1$;

Remark

$$\begin{aligned}\gamma\square{:}s_1.\varphi &:\leftrightarrow \mathsf{gamma}([\square{:}s_1]\varphi)\\ &:\leftrightarrow \mathsf{gamma}(\mathsf{intension}\ (\exists\square{:}s_1.\langle\square, \varphi\rangle))\end{aligned}$$

Contexts: Definition 2/2 (Axioms)

// unique name for gamma

$$\exists x.\mathsf{gamma} = x \tag{Ax.1}$$

// gamma as a function $2^{s_1 \otimes s_2} \to \mathsf{Context}_{s_1}^{s_2}$

$$\forall s_1, s_2{:}\mathsf{Sort}.\forall \alpha{:}2^{s_1 \otimes s_2}.\exists C{:}\mathsf{Context}_{s_1}^{s_2}.\mathsf{gamma}\ \alpha = C \tag{Ax.2}$$

// gamma is injective

$$\forall s_1, s_2{:}\mathsf{Sort}.\forall \alpha_1, \alpha_2{:}2^{s_1 \otimes s_2}.(\mathsf{gamma}\ \alpha_1 = \mathsf{gamma}\ \alpha_2) \to (\alpha_1 = \alpha_2) \tag{Ax.3}$$

// carrier set for $\mathsf{Context}_{s_1}^{s_2}$

$$\forall s_1, s_2{:}\mathsf{Sort}.\top_{\mathsf{Context}_{s_1}^{s_2}} = \exists \alpha{:}[s_1 \to s_2].\mathsf{gamma}\ \alpha \tag{Ax.4}$$

// unique name for plug

$$\exists x.\mathsf{plug} = x \tag{Ax.5}$$

// plug definition

$$\begin{aligned}&\forall s_1, s_2{:}\mathsf{Sort}.\forall \alpha{:}2^{s_1 \otimes s_2}.\forall x{:}s_1.\\ &\qquad C[x] = \exists y{:}s_2.y \wedge (C = \mathsf{gamma}\ \alpha \wedge \langle x, y \rangle \in \mathsf{extension}\ \alpha)\end{aligned} \tag{Ax.6}$$

// extensionality

$$\forall s_1, s_2{:}\mathsf{Sort}.\forall C_1, C_2{:}\mathsf{Context}_{s_1}^{s_2}.C_1 = C_2 \leftrightarrow \forall x{:}s_1.C_1[x] = C_2[x] \tag{Ax.7}$$

Plugging is substitution

$$(\gamma x{:}s_1.\varphi)[\psi] = \varphi[\psi/x]$$

Extension to multi-holes:

$$(\gamma\Box_1{:}s_1.\ldots.\gamma\Box_n{:}s_n.\varphi)[\psi_1]\ldots[\psi_n] = \varphi[\psi_1/\Box_1]\ldots[\psi_n/\Box_n]$$

Context Composition

$$C \odot \langle C_1, \ldots, C_n \rangle :\leftrightarrow \gamma\square_1{:}s'_1 \ldots \gamma\square_n{:}s'_n.\, C[C_1[\square_1], \ldots, C_n[\square_n]] \quad \text{(Ntn.3)}$$

$$(C \odot \langle C_1, \ldots, C_n \rangle)[\psi_1, \ldots, \psi_n] = C[C_1[\psi_1], \ldots, C_n[\psi_n]].$$

Plan

$\mathbb{K}$ contextual Rule

$\mathbb{K}$

```
1 rule I1 + I2 => I1 +Int I2
```

$\mathbb{ML}$:

$$\forall C{:}\mathrm{Context}^{\mathrm{Cell}\langle \mathrm{T}\rangle}_{\mathrm{Cell}\langle \mathrm{k}\rangle}.\forall \kappa{:}\mathrm{K}.\forall i_1, i_2{:}\mathrm{Int}.$$
$$(C \circ C_\kappa)[\mathrm{plus}(i_1, i_2)] \to \bullet(C \circ C_\kappa)[i_1 +\mathrm{Int}\ \ i_2] \quad (\mathrm{Ax.8})$$

where $C_\kappa :\leftrightarrow \gamma\square{:}\mathrm{KItem}.\mathrm{Cell}\langle \mathrm{k}\rangle(\square \curvearrowright \kappa)$.

Local Rule (Multi-Context)

$\mathbb{K}$

```
rule <k> X = I:Int; => .K ... </k> <state>... X |-> (_ => I)
      ...</state>
```

$\mathbb{ML}$

$$\forall C{:}\mathrm{Context}_{\mathrm{Cell}\langle \mathrm{k}\rangle, \mathrm{Cell}\langle \mathrm{state}\rangle}.\forall x{:}\mathrm{Id}.\forall i, v{:}\mathrm{Int}.\forall m_1, m_2{:}\mathrm{Map}.\forall \kappa{:}\mathrm{K}. \quad (C \odot \langle C_1, C_2\rangle)[\mathrm{assign}(x, i)][v] \to \bullet(C \odot \langle C_1, C_2\rangle)[\mathrm{dotK}, i] \tag{Ax.9}$$

where $C_1 :\leftrightarrow \gamma\square{:}\mathrm{K}.\langle\!|\mathrm{k}|\!\rangle(\square \curvearrowright \kappa)$ and
$C_2 :\leftrightarrow \gamma\square{:}\mathrm{Int}.\langle\!|\mathrm{state}|\!\rangle(m_1\_\mathrm{Map}\_x \mapsto \square\_\mathrm{Map}\_m_2)$.

Attribute `strict`

$\mathbb{K}$

```
syntax Stmt   ::= Id "=" AExp ";"                [strict(2)]
```

$\mathbb{ML}$

$$\forall C{:}\mathrm{Context}^{\mathrm{Cell}\langle\mathrm{T}\rangle}_{\mathrm{Cell}\langle\mathrm{k}\rangle}.\forall\kappa{:}\mathrm{K}.\forall x_1{:}\mathrm{Id}.\forall x_2{:}\mathrm{AExp}. \tag{Ax.10}$$

$$(C \circ C_\kappa)[\mathrm{assign}(x_1, x_2)] \wedge \neg\mathrm{KResult}(x_2) \rightarrow \bullet(C \circ C_\kappa)[x_2 \curvearrowright C_{\mathrm{assign},2}] \tag{Ax.11}$$

$$\forall C{:}\mathrm{Context}^{\mathrm{Cell}\langle\mathrm{T}\rangle}_{\mathrm{Cell}\langle\mathrm{k}\rangle}.\forall\kappa{:}\mathrm{K}.\forall x_1{:}\mathrm{Id}.\forall x_2{:}\mathrm{AExp}. \tag{Ax.12}$$

$$(C \circ C_\kappa)[x_2 \curvearrowright C_{\mathrm{assign},2}] \wedge \mathrm{KResult}(x_2) \rightarrow \bullet(C \circ C_\kappa)[C_{\mathrm{assign},2}[x_2]] \tag{Ax.13}$$

where $C_\kappa = \gamma\Box{:}\mathrm{KItem}.\mathrm{Cell}\langle\mathrm{k}\rangle(\Box \curvearrowright \kappa)$, and
$C_{\mathrm{assign},2} = \gamma\Box{:}\mathrm{AExp}.\mathrm{assign}(x_1, \Box)$

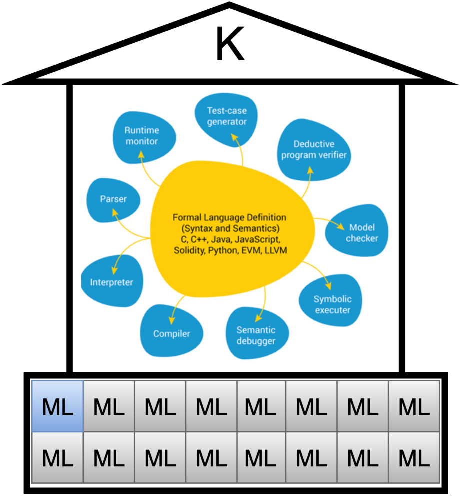
Plan

Concluding remarks

- problem addressed: the challenge of encoding in $\mathbb{ML}$ the $\mathbb{K}$'s abstract rewrites rules
- proposed solution: theory of contexts in ML for uniformly axiomatizing these kinds of rules
- demonstrate its application using the K definition of the IMP language as an example.

Future Work

- formally bridging the gap between $\mathbb{K}$'s abstract rewrite rules and their $\mathbb{ML}$ denotations



Questions?

Thanks!