Matching μ-Logic: Foundation of A Unifying Programming Language Framework

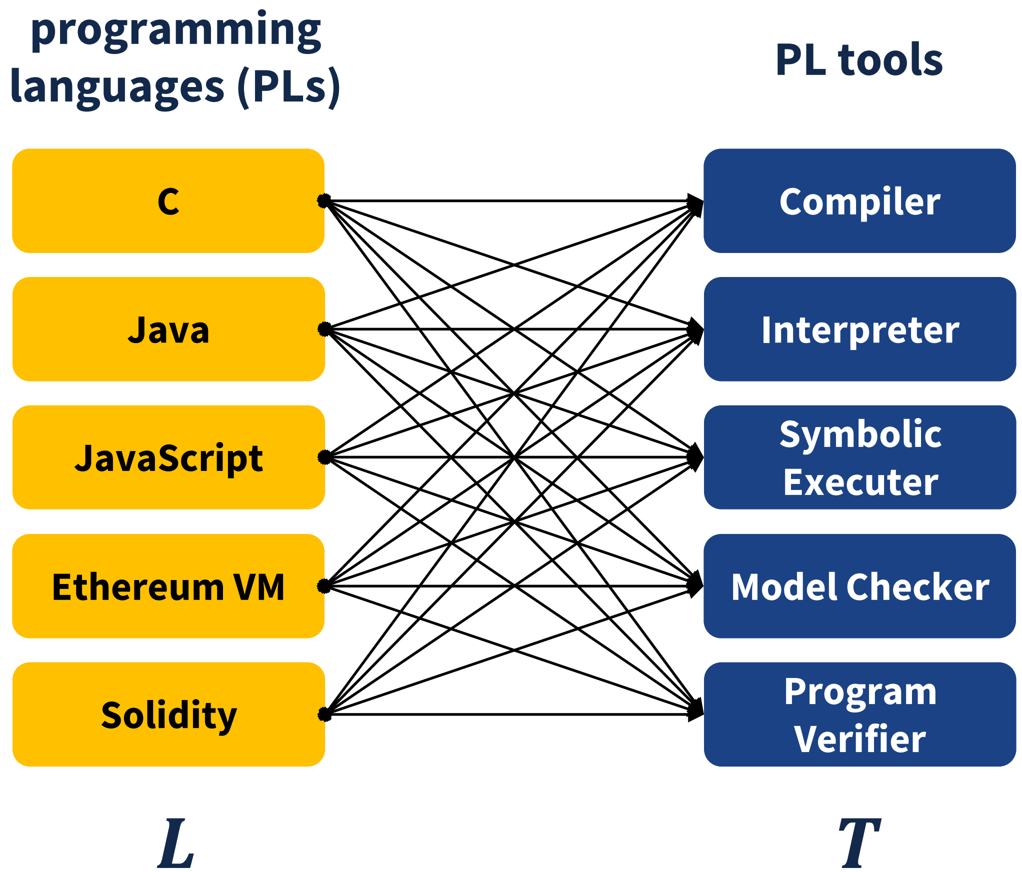
Xiaohong Chen
PhD Final Exam
University of Illinois Urbana-Champaign
Department of Computer Science

May 3, 2023

Overview

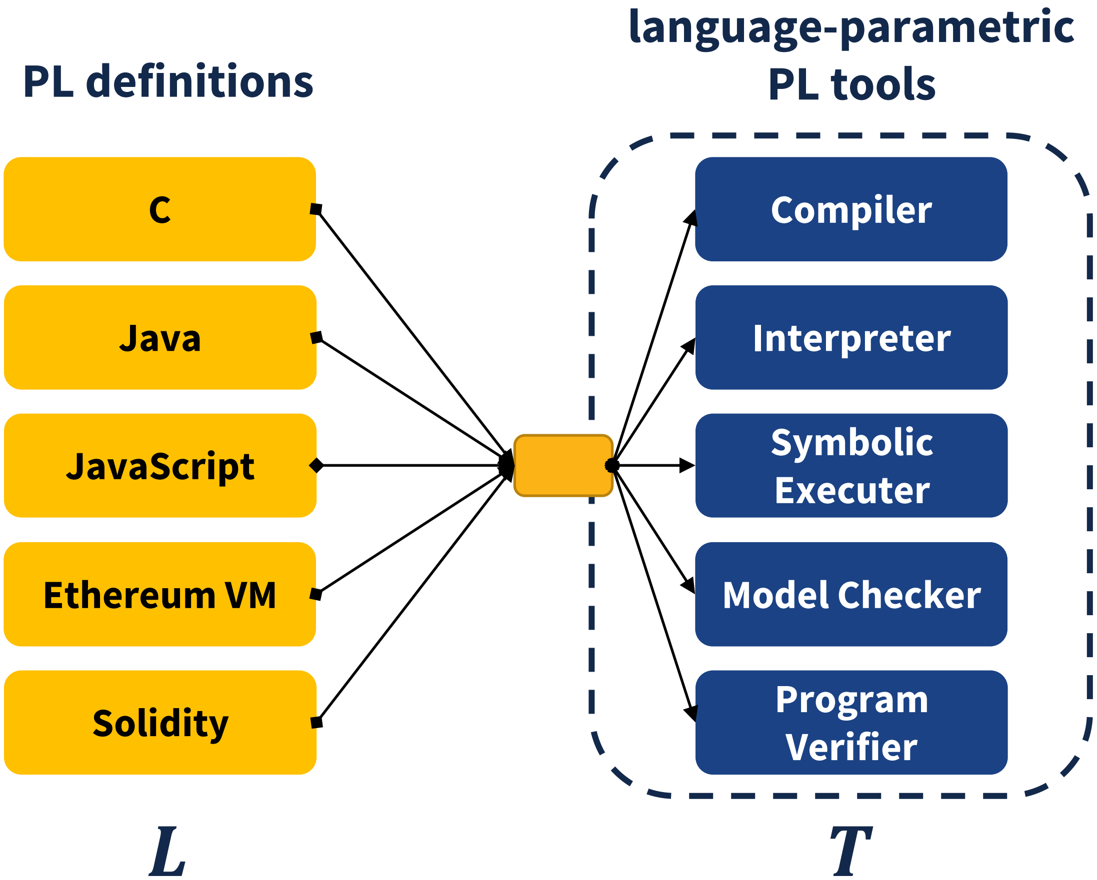
- **Introduction to a Unifying Programming Language Framework**
  - Motivating Example: The K Semantic Framework
  - Research Challenge: Proving the Correctness of K
- **Main Contribution: Matching μ-Logic**
  - Basic Definitions
  - Expressive Power
  - Proof System and Proof Checker
  - Automatic Theorem Prover
- **Using Matching μ-Logic to Prove the Correctness of K**
- **Concluding Remarks**

Programming Language Design & Implementation: State-of-the-Art



$L \times T$ systems
to develop and maintain

A Unifying Programming Language Framework



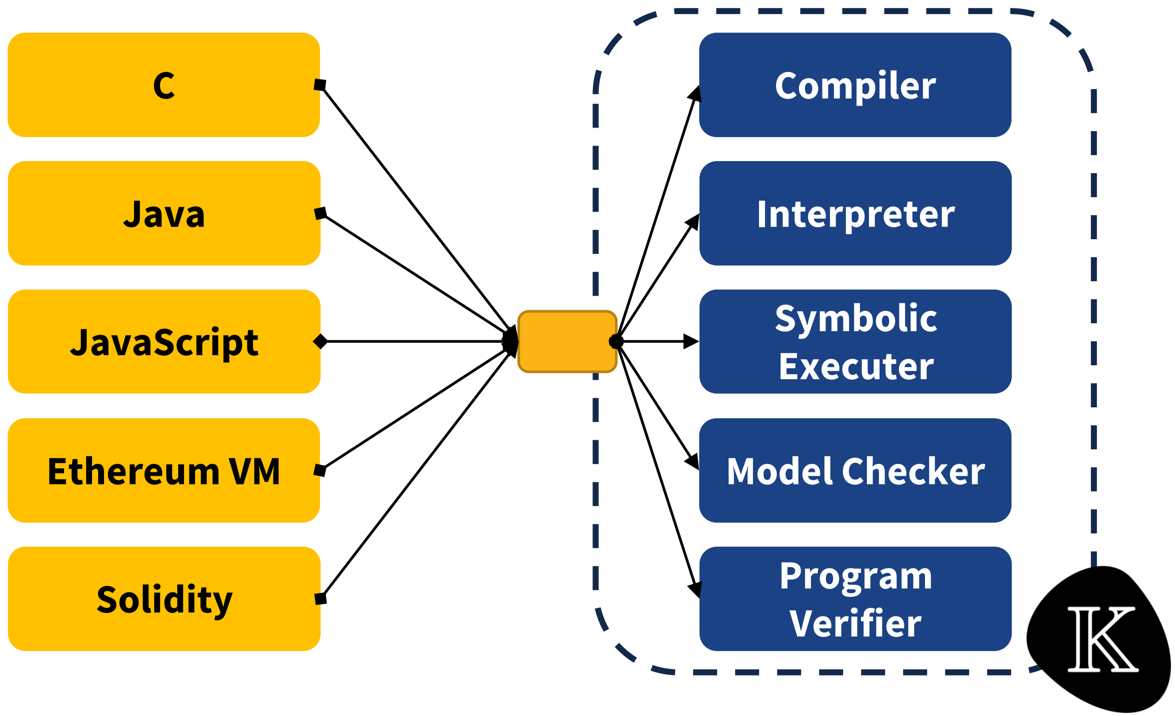
$L + T$ systems
to develop and maintain

K Semantic Framework https://kframework.org/

formal systems laboratory

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

runtime verification



K has wide applications

RV-Match

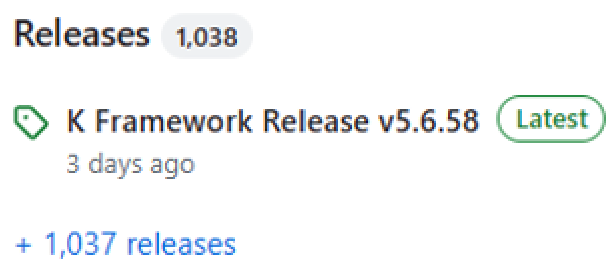
NASA

BOEING

ethereum

Research Challenge: Proving the Correctness of K

- **K has a large code base**
  - >500k LOC in 4 programming languages
  - complex data structures, algorithms, and optimizations
- **K is constantly evolving**
  - latest release: 3 days ago



- **It’s not practical to thoroughly verify the entire K.**

- **Main Idea: Translation Validation**

Main Idea: Translation Validation

K	**Matching μ-Logic: Foundation of K**
A PL definition **Ethereum VM**	A *logical theory* Γ^{EVM}
Any PL task • program execution **Interpreter** • formal verification **Program Verifier**	A *logical theorem* proved by a *proof system* • $\Gamma^{\text{EVM}} \vdash t_{\text{init}} \Rightarrow_{\text{exec}} t_{\text{final}}$ • $\Gamma^{\text{EVM}} \vdash \varphi_{\text{pre}} \rightsquigarrow \varphi_{\text{post}}$
Correctness of the task	Generating the proof and checking it using a *200-LOC proof checker*

correctness of any task done by any tool of any PL in K $\Longrightarrow$ **correctness of <u>1</u> task (proof checking) done by <u>1</u> program (proof checker)**

Why Matching μ-Logic?

- **We tried many logics/calculi/foundations**

  First-order logic; Second/higher-order logic; Least fixpoint logic; Modal logics; Temporal logics (LTL, CTL, CTL*, …), λ-calculus; Type systems (parametric, dependent, inductive, …); μ-calculus; Hoare logics; Separation logics; Dynamic logics; Rewriting logic; Reachability logic; Equational logic; Small-/big-step SOS; Evaluation contexts; Abstract machines (CC, CK, CEK, SECD, …); Chemical abstract machine; Axiomatic; Continuations; Denotational; Initial Algebras; …

- **… but each of the above had limitations**
  - Some only handle certain aspects of K (e.g., operational semantics)
  - Some are "design patterns" (e.g., Hoare logics)
  - Some are domain-specific (e.g., separation logic)
  - Some require complex encodings/translations

- **Matching μ-logic: Expressive and Small**
  - PLs defined as theories; PL tools specified by theorems
  - Logics defined as theories; logical proof rules proved as theorems
  - A 15-rule proof system and a 200-LOC proof checker: small trust base

Overview

- **Introduction to a Unifying Programming Language Framework**
  - Motivating Example: The K Semantic Framework
  - Research Challenge: Proving the Correctness of K
- **Main Contribution: Matching μ-Logic**
  - Basic Definitions
  - Expressive Power
  - Proof System and Proof Checker
  - Automatic Theorem Prover
- **Using Matching μ-Logic to Prove the Correctness of K**
- **Concluding Remarks**

Matching μ-Logic Syntax

Matching μ-logic formulas, called *patterns*:

$$\varphi ::= \underbrace{\boxed{x \mid \sigma(\varphi_1, \ldots, \varphi_n)}}_{\textbf{structures}} \mid \underbrace{\boxed{\varphi_1 \land \varphi_2 \mid \neg\varphi}}_{\textbf{logical constraints}} \mid \underbrace{\boxed{\exists x.\, \varphi}}_{\substack{\textbf{first-order}\\ \textbf{quantification}}} \mid \underbrace{\boxed{X \mid \mu X.\, \varphi}}_{\substack{\textbf{fixpoints}\\ \textbf{(in this talk)}}}$$

- X a *set variable*, ranging over sets
- $\mu X.\, \varphi$ the *least fixpoint* of φ, where X occurs positively in φ
- $\nu X.\, \varphi \equiv \neg \mu X.\, \neg\varphi[\neg X / X]$ the *greatest fixpoint* of φ

Matching μ-Logic Semantics

A matching μ-logic *model* has:

- a carrier set M
- a function $\sigma_M : M \times \cdots \times M \to \boldsymbol{\mathcal{P}}(\boldsymbol{M})$ for each symbol σ

Given a model M and a variable valuation ρ:

$$\boldsymbol{\varphi} \xRightarrow{\textbf{pattern matching}} |\boldsymbol{\varphi}|_{\boldsymbol{M},\boldsymbol{\rho}} \subseteq \boldsymbol{M}$$

- $|x|_{M,\rho} = \{\rho(x)\}$
- $|\sigma(\varphi_1, \ldots, \varphi_n)|_{M,\rho} = \bigcup\{\sigma_M(a_1, \ldots, a_n) \mid a_i \in |\varphi_i|_{M,\rho}\}$
- $|\varphi_1 \wedge \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cap |\varphi_2|_{M,\rho}$
- $|\neg\varphi|_{M,\rho} = M \setminus |\varphi|_{M,\rho}$
- $|\exists x. \varphi|_{M,\rho} = \bigcup\{|\varphi|_{M,\rho[a/x]} \mid a \in M\}$
- $|X|_{M,\rho} = \rho(X)$
- $|\mu X. \varphi|_{M,\rho} = \mathbf{lfp}\left(A \mapsto |\varphi|_{M,\rho[A/X]}\right)$

Examples of Fixpoint Patterns

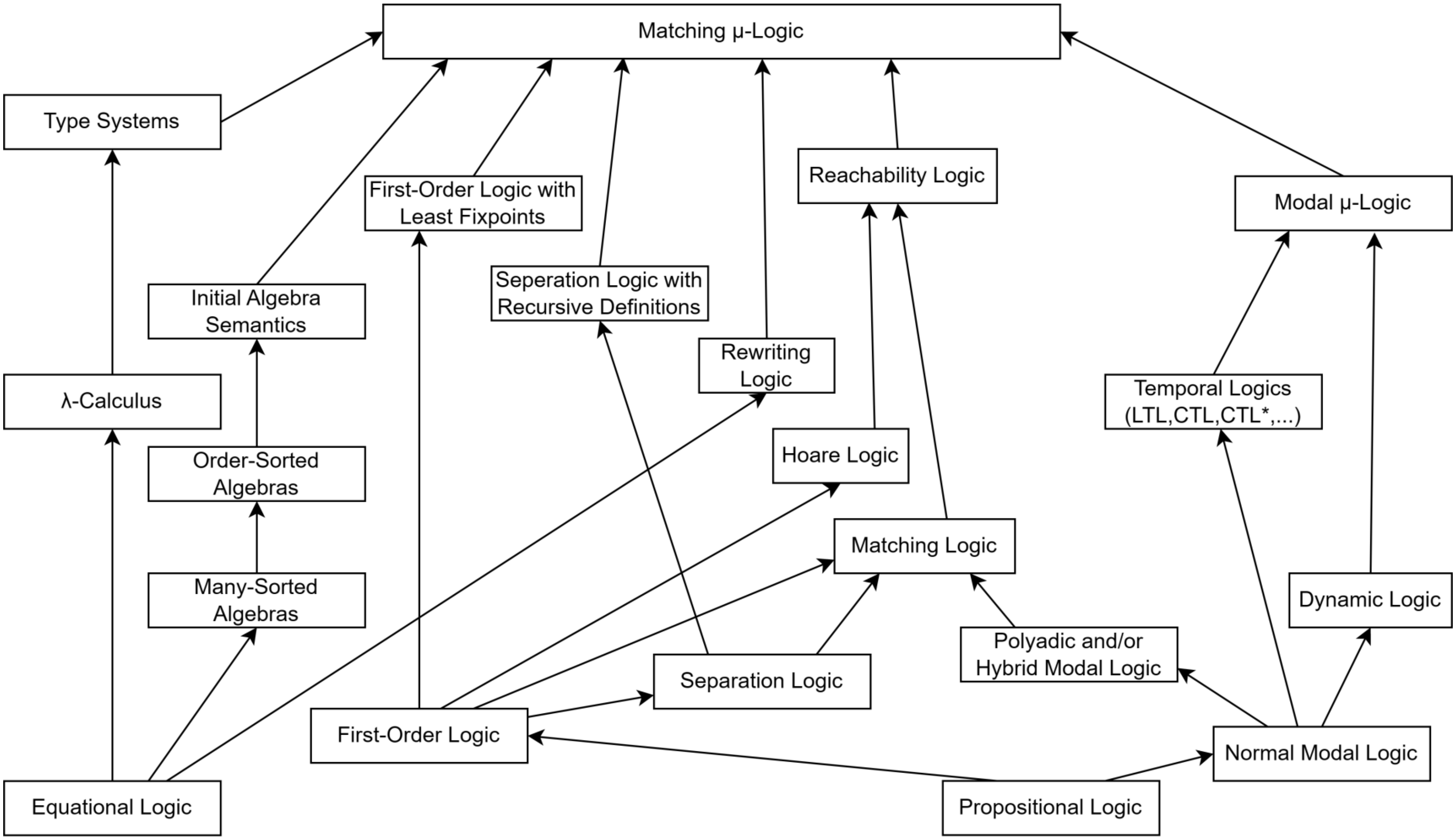
- **inductive datatypes** [JLAMP'21]
  - `type nat = Zero | Succ of nat`
  - $\top_{\mathbf{nat}} = \mu N.\, \mathbf{0} \vee \mathbf{Succ}(N)$
  - `type list = Nil | Cons of nat * list`
  - $\top_{\mathbf{list}} = \mu L.\, \mathbf{Nil} \vee \mathbf{Cons}(\top_{\mathbf{nat}}, L)$
- **program execution** [LICS'19, CAV'21]
  - $t_1 \Rightarrow_{\mathrm{exec}} t_2 \quad \equiv \quad t_1 \rightarrow \underbrace{\mathbf{eventually}\ t_2}_{\mu S.\, t_2 \vee (\mathbf{next}\ S)}$
- **formal verification** [LICS'19, OOPSLA'23]
  - $\varphi_{\mathrm{pre}} \rightsquigarrow \varphi_{\mathrm{post}} \quad \equiv \quad \varphi_{\mathrm{pre}} \rightarrow \underbrace{\mathbf{weak\text{-}eventually}\ \varphi_{\mathrm{post}}}_{\nu S.\, \varphi_{\mathrm{post}} \vee (\mathbf{next}\ S)}$

  (if φ_{pre} holds when P starts, then φ_{post} holds when P terminates)

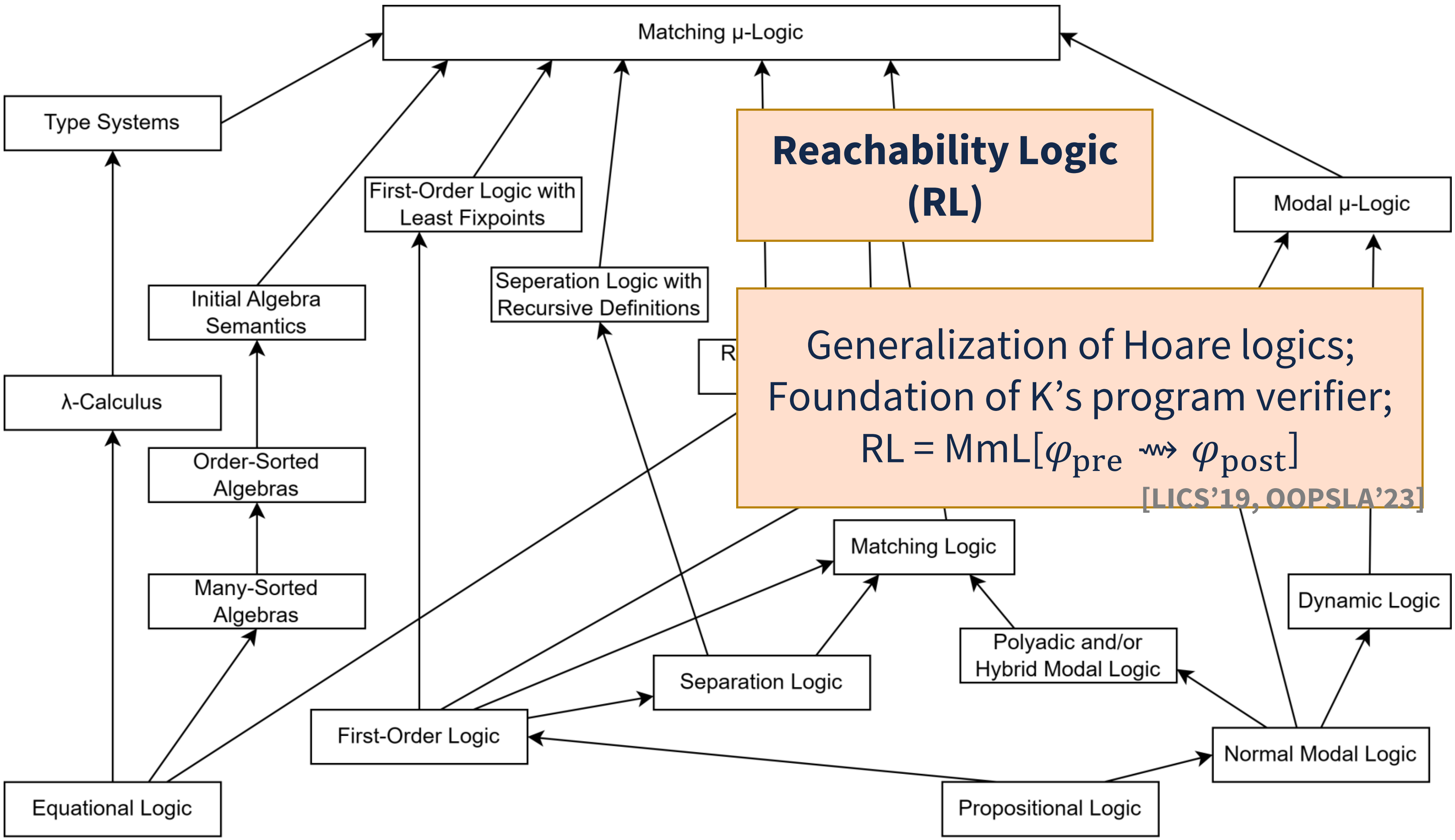
Various forms/instances of fixpoints are definable by patterns.

Matching μ-Logic (MmL) Expressive Power

[Chap 5 of Thesis, also in LICS'19, OOPSLA'20, ICFP'20, CAV'21, JLAMP'21, JLAMP'22, OOPSLA'23]



Matching μ-Logic (MmL) Expressive Power

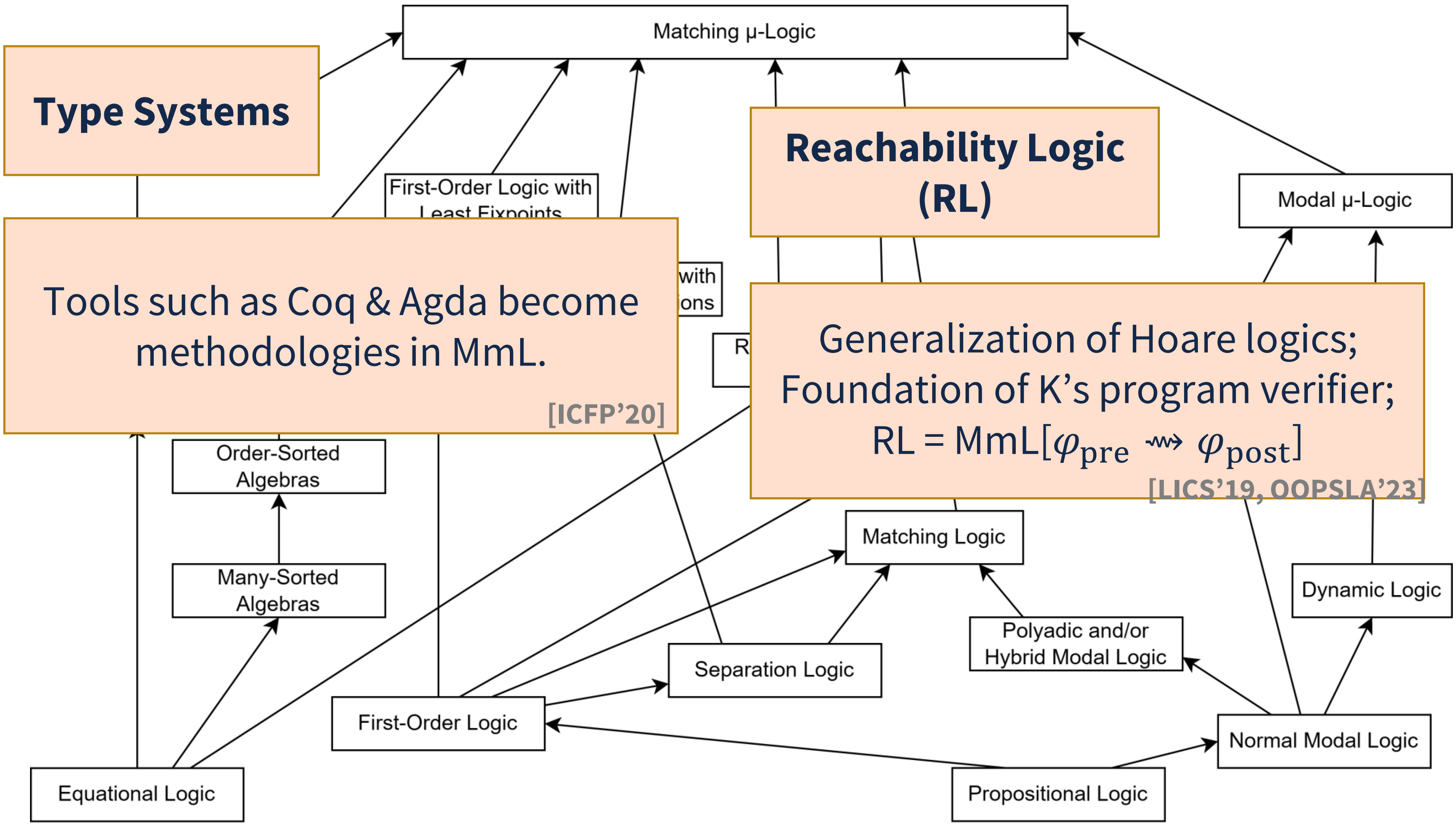


Reachability Logic (RL)

Generalization of Hoare logics;
Foundation of K's program verifier;
RL = MmL[$\varphi_{\mathrm{pre}} \rightsquigarrow \varphi_{\mathrm{post}}$]

[LICS'19, OOPSLA'23]

Matching μ-Logic (MmL) Expressive Power



Type Systems

Tools such as Coq & Agda become methodologies in MmL.

[ICFP'20]

Reachability Logic (RL)

Generalization of Hoare logics;
Foundation of K's program verifier;
RL = MmL[$\varphi_{\text{pre}} \rightsquigarrow \varphi_{\text{post}}$]

[LICS'19, OOPSLA'23]

Matching μ-Logic Proof System

(only 14 proof rules)

(Propositional 1)	$\varphi \to (\psi \to \varphi)$
(Propositional 2)	$(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$
(Propositional 3)	$((\varphi \to \bot) \to \bot) \to \varphi$
(Modus Ponens)	$\dfrac{\varphi \quad \varphi \to \psi}{\psi}$
($\exists$-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$
($\exists$-Generalization)	$\dfrac{\varphi \to \psi}{(\exists x.\, \varphi) \to \psi}\ x \notin FV(\psi)$
(Propagation$_\vee$)	$C[\varphi \vee \psi] \to C[\varphi] \vee C[\psi]$
(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ with $x \notin FV(C)$
(Framing)	$\dfrac{\varphi \to \psi}{C[\varphi] \to C[\psi]}$
(Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
(Prefixpoint)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{(\mu X.\, \varphi) \to \psi}$
(Existence)	$\exists x.\, x$
(Singleton)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

Defines provability relation

$$\Gamma \vdash \varphi$$

theory theorem

$$\varphi[(\mu X.\, \varphi)/X] \leftrightarrow \mu X.\, \varphi$$

(Knaster-Tarski) $\dfrac{\varphi[\psi/X] \leftrightarrow \psi}{(\mu X.\, \varphi) \to \psi}$

proof rules for fixpoints

Deriving Mathematical Induction in Matching μ-Logic

Mathematical Induction: To show a property P holds for all naturals, prove:
(**basis**). The number 0 satisfies P
(**step**). If n satisfies P then $n + 1$ also satisfies P.

Step 1. Note that $\top_{\mathbf{nat}} = \mu N.\, 0 \vee \mathbf{succ}(N)$ captures all natural numbers.
Step 2. Set the proof goal $\vdash \big(\mu N.\, 0 \vee \mathbf{succ}(N)\big) \to \psi_P$
Step 3. Apply (**Knaster Tarski**) and get

$$\vdash \big(0 \vee \mathbf{succ}(\psi_P)\big) \to \psi_P$$

i.e., Sub-Goal-1 $0 \to \psi_P$ .... (**basis**)
Sub-Goal-2 $\mathbf{succ}(\psi_P) \to \psi_P$ .... (**step**)

(**Knaster Tarski**)

$$\frac{\varphi[\psi/X] \to \psi}{\mu X.\, \varphi \to \psi}$$

Various forms/instances of fixpoints reasoning are supported by (Knaster Tarski)

Matching μ-Logic Proof Object

(Propositional 1)	$\varphi \to (\psi \to \varphi)$
(Propositional 2)	$(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$
(Propositional 3)	$((\varphi \to \bot) \to \bot) \to \varphi$
(Modus Ponens)	$\dfrac{\varphi \quad \varphi \to \psi}{\psi}$
($\exists$-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$
($\exists$-Generalization)	$\dfrac{\varphi \to \psi}{(\exists x.\, \varphi) \to \psi}\ x \notin FV(\psi)$
(Propagation$_\vee$)	$C[\varphi \vee \psi] \to C[\varphi] \vee C[\psi]$
(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ with $x \notin FV(C)$
(Framing)	$\dfrac{\varphi \to \psi}{C[\varphi] \to C[\psi]}$
(Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
(Prefixpoint)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{(\mu X.\, \varphi) \to \psi}$
(Existence)	$\exists x.\, x$
(Singleton)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

$$\underset{\text{theory}}{\boldsymbol{\Gamma}} \vdash \underset{\text{theorem}}{\boldsymbol{\varphi}}$$

$$\left.\begin{array}{l}\#1.\ \psi_1 \\ \#2.\ \psi_2 \\ \ldots \\ \#100.\ \psi_{100}\end{array}\right\} \boldsymbol{\Gamma}$$

…

…

#246. $\psi_2 \to \varphi'$

#247. φ' // by (Modus Ponens)
// on #2 and #246

…

…

#999. $\boldsymbol{\varphi}$

a proof object;
very easy & fast to check;
embarrassingly parallelable

Matching μ-Logic Proof Checker

- We use Metamath [Megill & Wheeler] http://metamath.org
  - to encode proof objects &
  - check them automatically
  - embarrassingly parallelable
- Very small trust base
  - Matching μ-logic: 200 LOC
  - Metamath itself:
    - 350 LOC in Python
    - 400 LOC in Haskell
    - 550 LOC in C#
    - …

```
$c \imp ( ) #Pattern |- $.

$v ph1 ph2 ph3 $.
ph1-is-pattern $f #Pattern ph1 $.
ph2-is-pattern $f #Pattern ph2 $.
ph3-is-pattern $f #Pattern ph3 $.
imp-is-pattern
  $a #Pattern ( \imp ph1 ph2 ) $.

axiom-1
  $a |- ( \imp ph1 ( \imp ph2 ph1 ) ) $.

axiom-2
  $a |- ( \imp ( \imp ph1 ( \imp ph2 ph3 ) )
        ( \imp ( \imp ph1 ph2 )
               ( \imp ph1 ph3 ) ) ) $.

${
  rule-mp.0 $e |- ( \imp ph1 ph2 ) $.
  rule-mp.1 $e |- ph1 $.
  rule-mp   $a |- ph2 $.
$}                                    ...
```

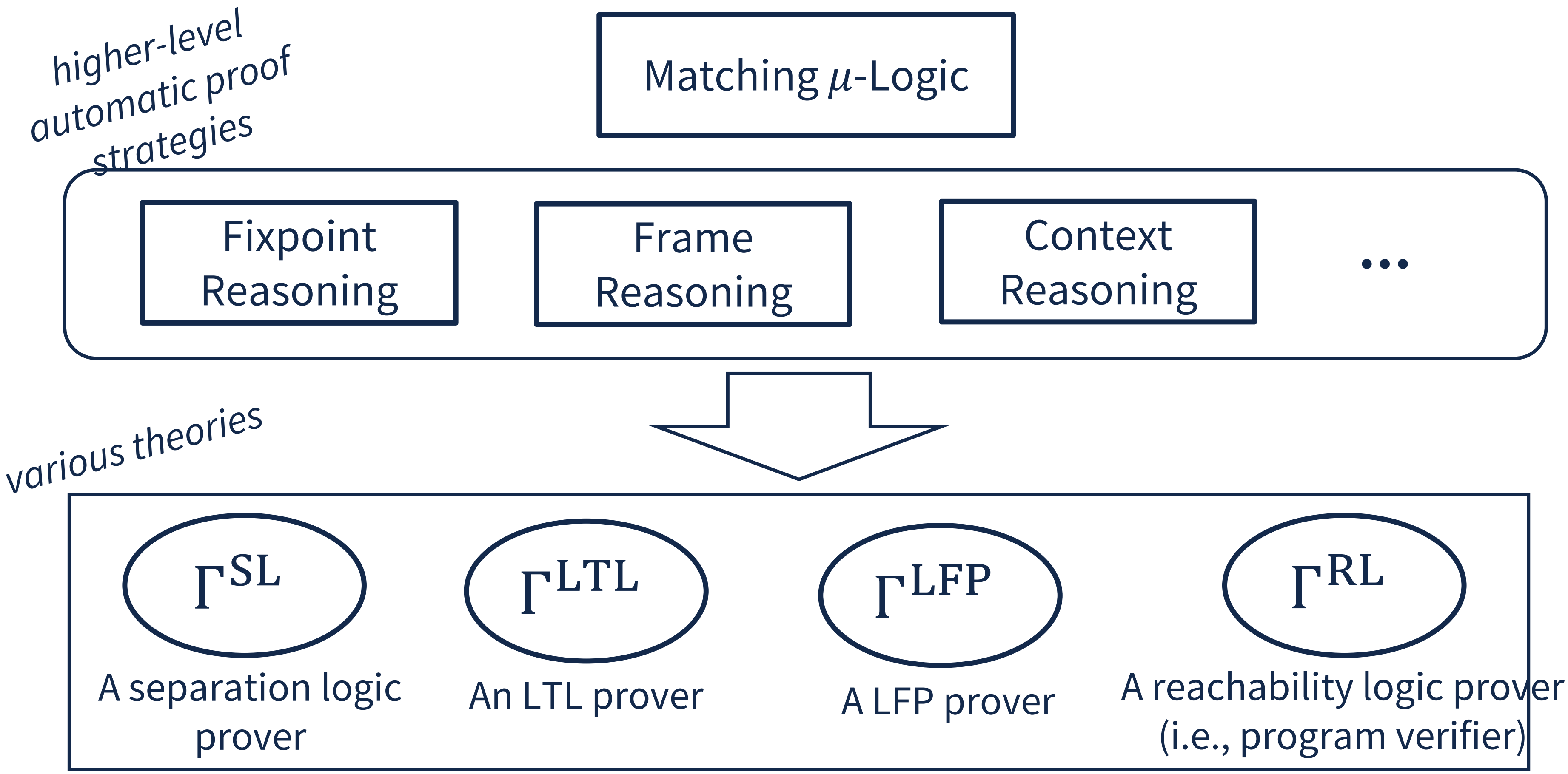
Matching μ-logic syntax & proof rules; Defined in 200 LOC

```
imp-refl $p |- ( \imp ph1 ph1 )
$=
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  imp-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  ph1-is-pattern imp-is-pattern
  imp-is-pattern ph1-is-pattern
  ph1-is-pattern ph1-is-pattern
  imp-is-pattern imp-is-pattern
  ph1-is-pattern ph1-is-pattern
  imp-is-pattern imp-is-pattern
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  ph1-is-pattern axiom-2
  ph1-is-pattern ph1-is-pattern
  ph1-is-pattern imp-is-pattern
  axiom-1 rule-mp ph1-is-pattern
  ph1-is-pattern axiom-1 rule-mp
$.
```

Proof objects (automatically checked)

Checking proof objects is fast and trustworthy.

Automatic Theorem Prover for Matching μ-Logic



- Separation logic: Proved 265/280 benchmark tests in SL-COMP'19
  - (latest WIP even reached 280/280!)

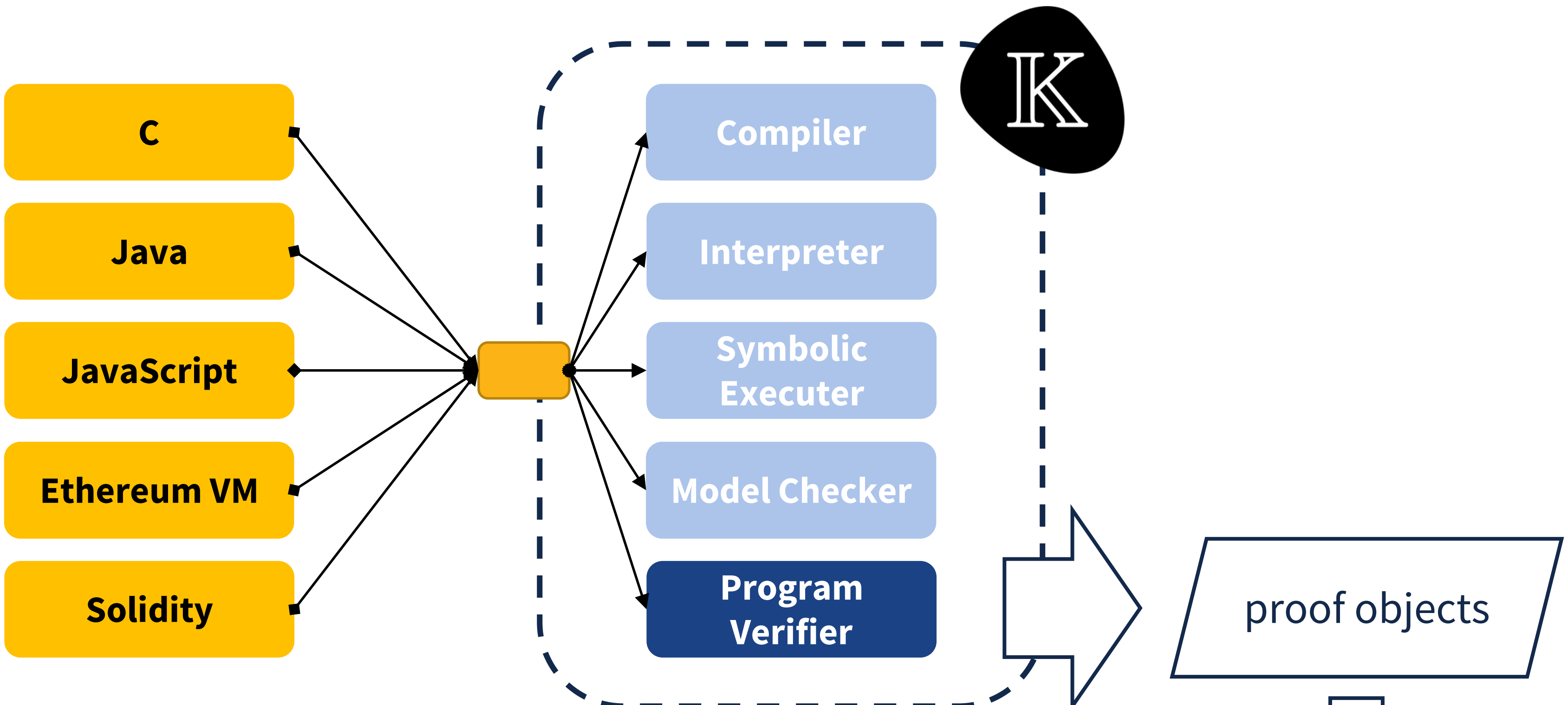
Overview

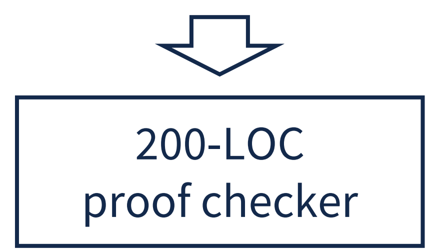
- **Introduction to a Unifying Programming Language Framework**
  - Motivating Example: The K Semantic Framework
  - Research Challenge: Proving the Correctness of K
- **Main Contribution: Matching μ-Logic**
  - Basic Definitions
  - Expressive Power
  - Proof System and Proof Checker
  - Automatic Theorem Prover
- **Using Matching μ-Logic to Prove the Correctness of K (in the translation validation style)**
  - Translating PL Definitions in K to Matching μ-Logic Theories
  - Generating Proof Objects for K's Program Verifier
- **Concluding Remarks**

Translating K to Matching μ-Logic

- Kore as an intermediate language between K and matching μ-logic
- Kore = K's "bytecode"
- Kore = theories of rewrites, equality, …
- Kore is now at core of K
  https://github.com/kframework/kore
- Any PL definition in K is now a matching μ-logic theory

Proving the Correctness of K's Program Verifier

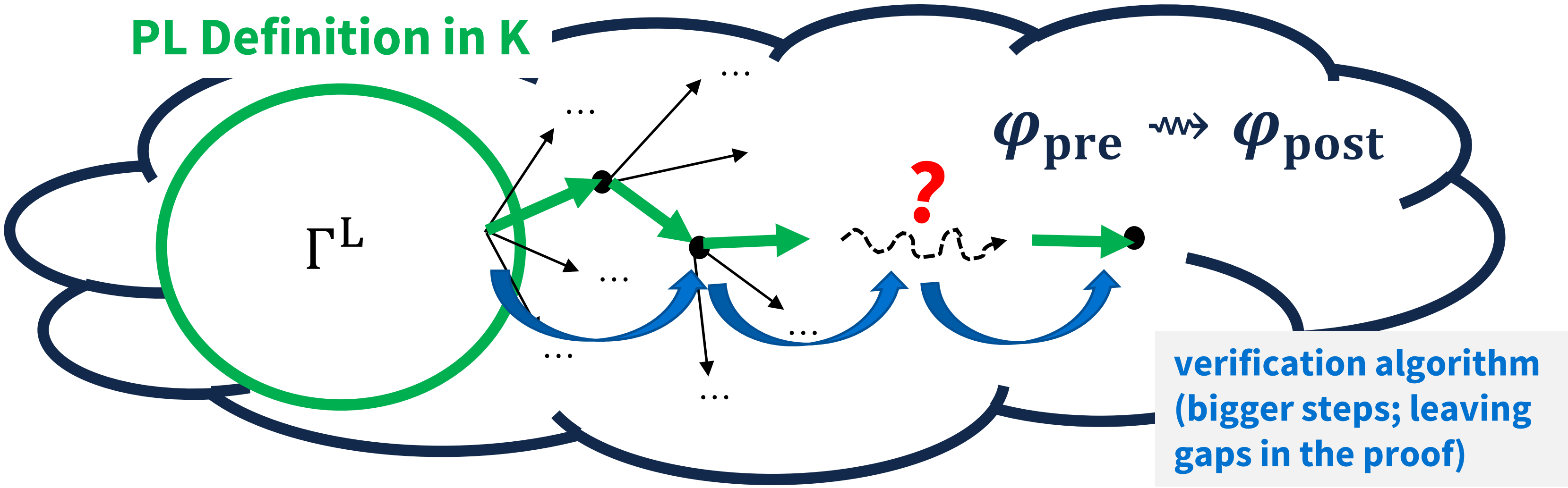




Challenge: Large code base; >120k lines of Haskell

Solution: Translation validation;
Generating proof objects and
checking them automatically

Program Verification is Actually Proof Search



A program verifier is a specialized, optimized, proof searcher.

Proof Generation for Program Verification

The K program verifier checks that P satisfies the pre/post-conditions φ_{pre} and φ_{post} in L

proof generation

fill in the "gaps" in the verification ("proof search")

$$\Gamma^L \vdash \varphi_{\text{pre}} \rightsquigarrow \varphi_{\text{post}}$$

a proof object

$$\left.\begin{array}{l}\text{#1. } \psi_1 \\ \text{#2. } \psi_2 \\ \ldots \\ \text{#100. } \psi_{100}\end{array}\right\} \Gamma^L$$

…

…

#247. $\psi_2 \rightarrow \varphi$

#247. φ // by (Modus Ponens)
// on #2 and #246

…

…

#99999. $\varphi_{\text{pre}} \rightsquigarrow \varphi_{\text{post}}$

Proof Generation for Program Verification

The K program verifier checks that P satisfies the pre/post-conditions φ_{pre} and φ_{post} in L

proof generation

filling the "gaps" in the verification ("proof search")

$$\Gamma^L \vdash \varphi_{\mathrm{pre}} \rightsquigarrow \varphi_{\mathrm{post}}$$

a proof object

proof checking

200-LOC matching μ-logic proof checker

P satisfies the spec.; proof available

something is wrong (verifier, proof generator, PL definitions, etc.)

Proof Generation: Complicated …

top-level proof goal $\Gamma^L \vdash \varphi_{\text{pre}} \rightsquigarrow \varphi_{\text{post}}$

$$\bigwedge_{(\psi_1 \Rightarrow \psi_2) \in A} \square \left(\forall FV(\psi_1, \psi_2).\, \psi_1 \Rightarrow^{+}_{reach} \psi_2\right) \wedge \bigwedge_{(\psi_1 \Rightarrow \psi_2) \in C} \circ\square \left(\forall FV(\psi_1, \psi_2).\, \psi_1 \Rightarrow^{+}_{reach} \psi_2\right) \to \left(\varphi \Rightarrow^{\triangle}_{reach} \psi\right)$$

$$\left(t_j^{\text{hint}} \wedge p_j^{\text{hint}}\right) \Rightarrow_{exec} \left(t_{j,1}^{\text{hint}} \wedge p_{j,1}^{\text{hint}}\right) \vee \ldots \vee \left(t_{j,l_j}^{\text{hint}} \wedge p_{j,l_j}^{\text{hint}}\right) \vee \left(t_j^{\text{rem}} \wedge p_j^{\text{rem}}\right)$$

sub-goal A

$$\left(t_j^{\text{hint}} \wedge p_{j,l}^{\text{hint}}\right) \to \left(lhs_{k_{j,l}}\theta_{k_{j,l}} \wedge q_{k_{j,l}}\theta_{k_{j,l}}\right)$$

$$\left(rhs_{k_{j,l}}\theta_{k_{j,l}} \wedge q_{k_{j,l}}\theta_{k_{j,l}}\right) \to \left(t_{j,l}^{\text{hint}} \wedge p_{j,l}^{\text{hint}}\right)$$

sub-goal B

$$\square\left(\forall FV(\varphi, \psi).\, \varphi \Rightarrow_{reach} \psi\right) \to \varphi' \Rightarrow_{reach} \varphi''$$

sub-goal C

…

…

…

… but none of the above needs to be trusted.

Evaluation

We tested on 3 PL paradigms:

- imperative
- register-based
- functional

Reduced K trust base
(~120k lines of Haskell)

Found issues in K
(missing axioms etc.)

Future work

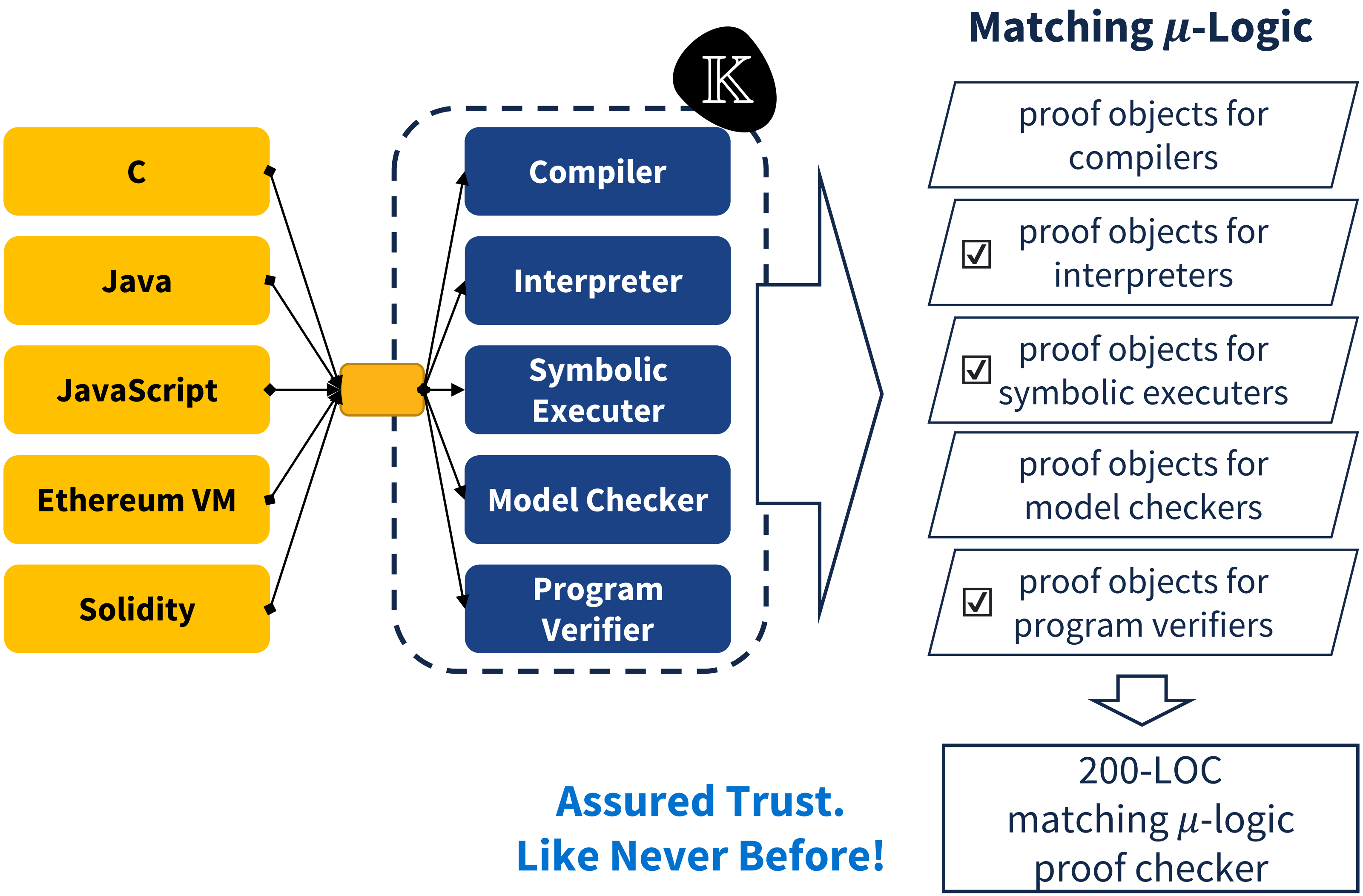
- Apply it to more PLs

proof generation time

proof checking time

Task	Spec. LOC	Steps	Hint Size	Proof Size	Time (seconds)		
					𝕂 Verifier	Gen.	Check
`sum.imp`	40	42	0.58 MB	37/1.6 MB	4.2	105	1.8
`sum.reg`	46	108	2.24 MB	111/3.6 MB	9.1	259	5.4
`sum.pcf`	18	22	0.29 MB	38/1.5 MB	2.9	119	2.4
`exp.imp`	27	31	0.5 MB	37/1.5 MB	3.7	108	2.0
`exp.reg`	27	43	0.96 MB	70/2.3 MB	4.7	177	3.1
`exp.pcf`	20	29	0.5 MB	65/2.3 MB	3.8	199	3.1
`collatz.imp`	25	55	1.14 MB	49/1.7 MB	4.8	138	2.6
`collatz.reg`	37	100	3.66 MB	209/4.7 MB	9.3	414	5.5
`collatz.pcf`	26	39	1.51 MB	110/2.2 MB	5.3	247	5.2
`product.imp`	44	42	0.62 MB	44/1.8 MB	3.9	124	2.4
`product.reg`	24	42	0.81 MB	65/2.3 MB	4.3	164	4.0
`product.pcf`	21	48	0.82 MB	80/2.8 MB	5.3	234	4.9
`gcd.imp`	51	93	1.9 MB	74/2.3 MB	22.9	237	2.7
`gcd.reg`	27	73	1.92 MB	124/3.3 MB	18.6	306	3.6
`gcd.pcf`	22	38	1.35 MB	150/3.2 MB	12.8	367	5.2
`ln/count-by-1`	44	25	0.24 MB	28/1.3 MB	2.7	81	1.6
`ln/count-by-2`	44	25	0.26 MB	28/1.3 MB	9.0	88	1.4
`ln/gauss-sum`	51	39	0.53 MB	38/1.6 MB	4.6	107	2.0
`ln/half`	62	65	1.3 MB	63/2.2 MB	13.1	173	3.0
`ln/nested-1`	92	84	1.88 MB	104/3.4 MB	7.5	231	5.9

Conclusion: Matching μ-Logic as A Unifying Foundation for Programming



Assured Trust.
Like Never Before!

Thank you

Xiaohong Chen
https://xchen.page
xc3@illinois.edu