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

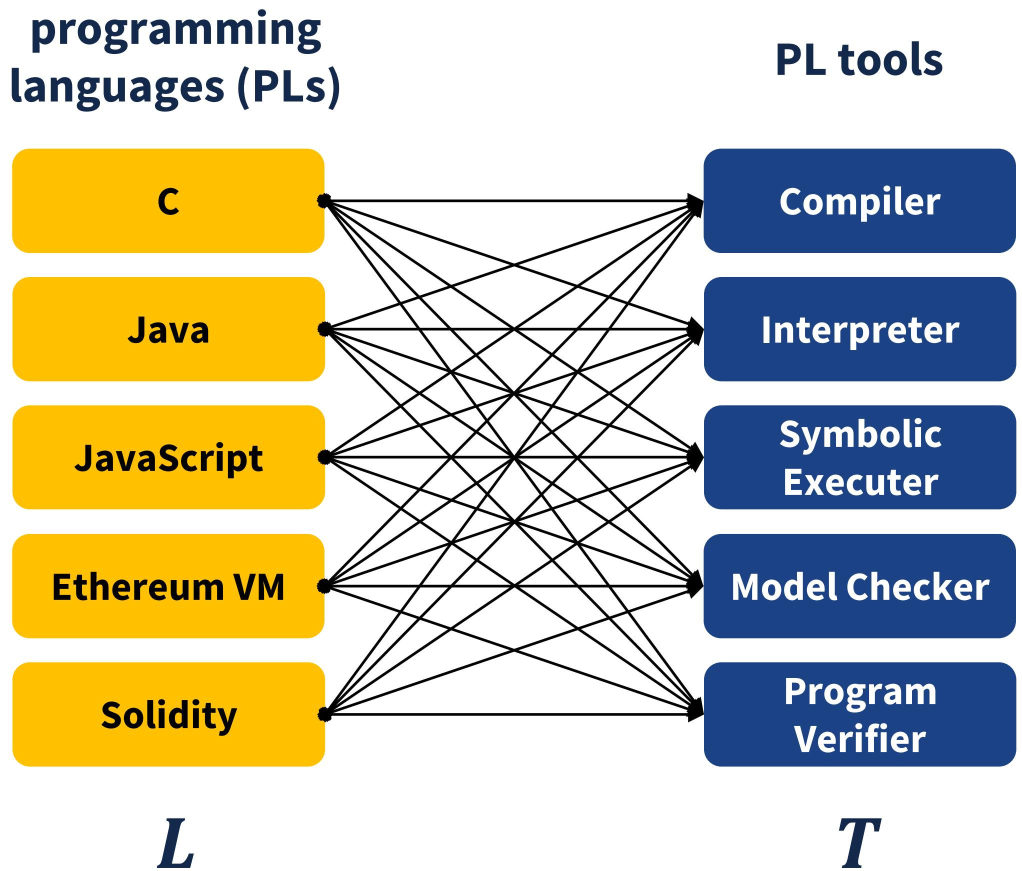
Overview

- **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**

Overview

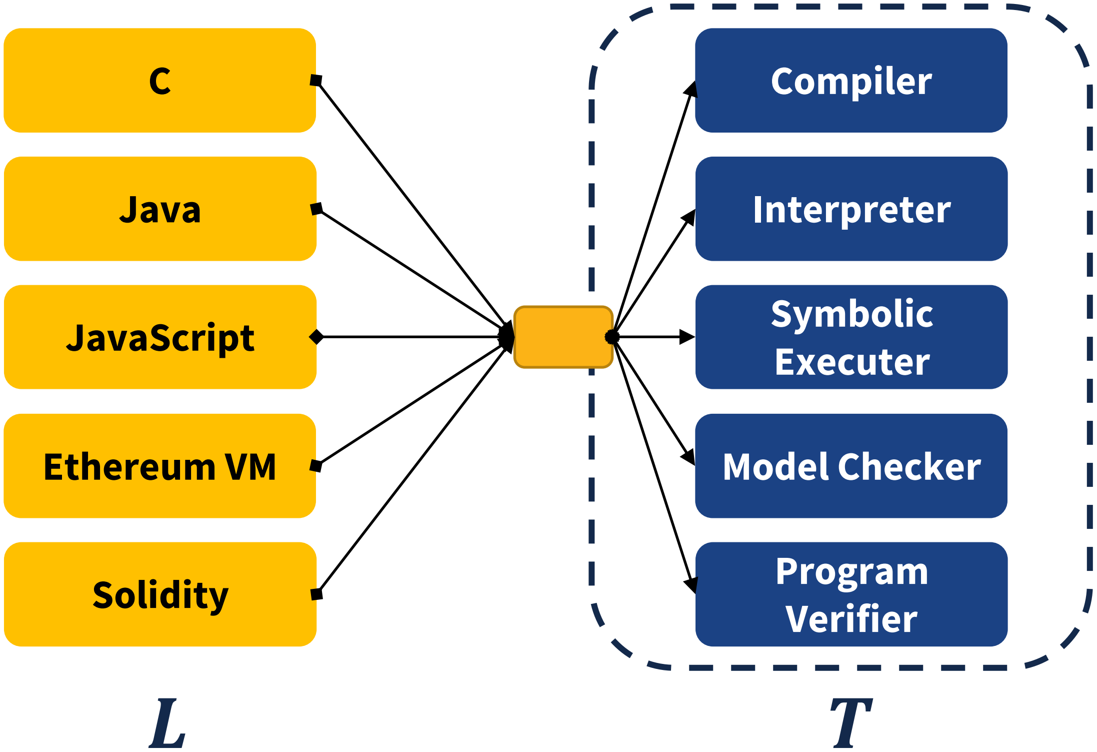
- **Introduction to a Unifying Programming Language Framework**
  - Motivating Example: The K Semantic Framework
  - Research Challenge: Ensure the Correctness of K
- **Main Contribution: Matching μ-Logic**
  - Basic Definitions
  - Expressive Power
  - Proof System and Proof Checker
  - Automatic Theorem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

Vision: A Unifying Programming Language Framework



$L + T$ systems
to develop and maintain

K Semantic Framework

https://kframework.org/

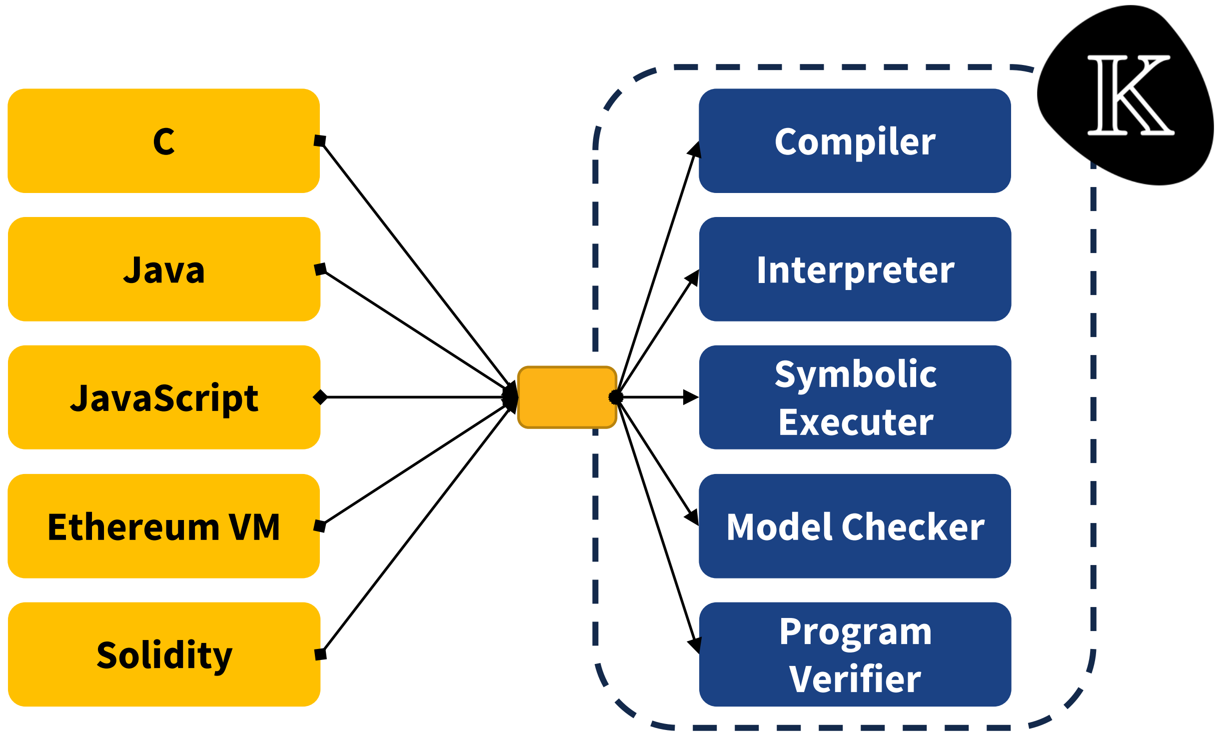
Ellison et al
[POPL'12, PLDI'15]

Bogdanas et al
[POPL'15]

Park et al
[PLDI'15]

Hildenbrandt et al
[CSF'18]

Filaretti et al
[Github Project 2019]

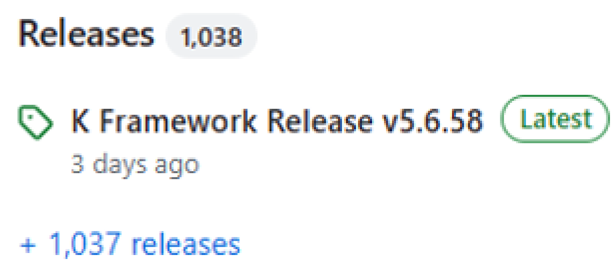


K has wide applications

RV-Match RV-Monitor NASA ethereum BOEING

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 fully verify K using traditional methods.**

Main Idea: Let's translate K to logic.

K	**Logical 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* • $\Gamma^{\mathrm{EVM}} \vdash t_{\mathrm{init}} \Rightarrow_{\mathrm{exec}} t_{\mathrm{final}}$ • $\Gamma^{\mathrm{EVM}} \vdash \varphi_{\mathrm{pre}} \Rightarrow_{\mathrm{verify}} \varphi_{\mathrm{post}}$
Correctness of the task	Generating the proof and checking it using a *proof checker*

correctness of any task done by any tool of any PL $\Longrightarrow$ **correctness of <u>1</u> task done by <u>1</u> program**

Which 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., only execution)
  - Some are "design patterns" (e.g., Hoare logics)
  - Modularity and heavy notation

- **Matching μ-logic: keep advantages and avoid limitations**
  - PLs defined as theories; PL tools specified by theorems
  - High expressive power (K and beyond)
  - 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: Ensure 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 101

Matching μ-logic formulas, called *patterns*:

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

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

Matching μ-Logic 101

A matching μ-logic *model* has:

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

Given a model M and a variable valuation ρ:

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

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

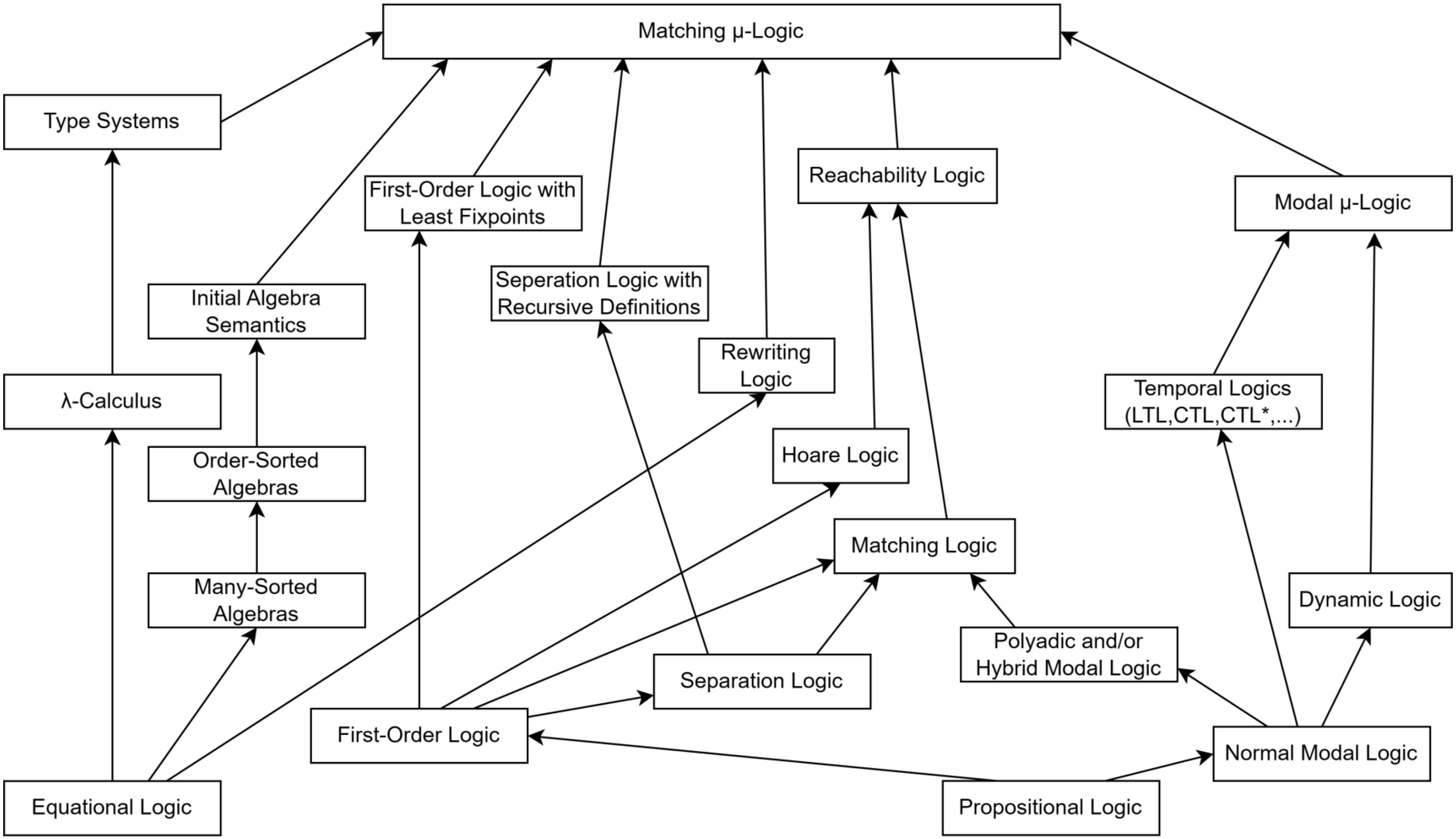
Examples of Fixpoint Patterns

- **inductive datatypes** [JLAMP'21]
  - `type list = Nil | Cons of int * list`
  - $\mu L.\, \mathbf{Nil} \vee \exists x.\, \mathbf{Cons}(x, L)$
  - $L =_{\mathrm{lfp}} \{\mathbf{Nil}\} \cup \{\mathbf{Cons}(x, l) \mid x \in \mathbf{int}, l \in L\}$
- **program execution** [LICS'19, CAV'21]
  - an execution trace from t_{init} to t_{final}
  - $t_{\mathrm{init}} \Rightarrow_{\mathrm{exec}} t_{\mathrm{final}} \quad \equiv \quad t_{\mathrm{init}} \to \underbrace{\mathbf{eventually}\ t_{\mathrm{final}}}_{\mu S.\, t_{\mathrm{final}} \vee (\mathbf{next}\ S)}$
- **formal verification** [LICS'19, OOPSLA'23]
  - if φ_{pre} holds when P starts, then φ_{post} holds when P terminates
  - $\varphi_{\mathrm{pre}} \Rightarrow_{\mathrm{verify}} \varphi_{\mathrm{post}} \quad \equiv \quad \varphi_{\mathrm{pre}} \to \underbrace{\mathbf{weak\text{-}eventually}\ \varphi_{\mathrm{post}}}_{\nu S.\, \varphi_{\mathrm{post}} \vee (\mathbf{next}\ S)}$

Various forms/instances of fixpoints are definable by patterns.

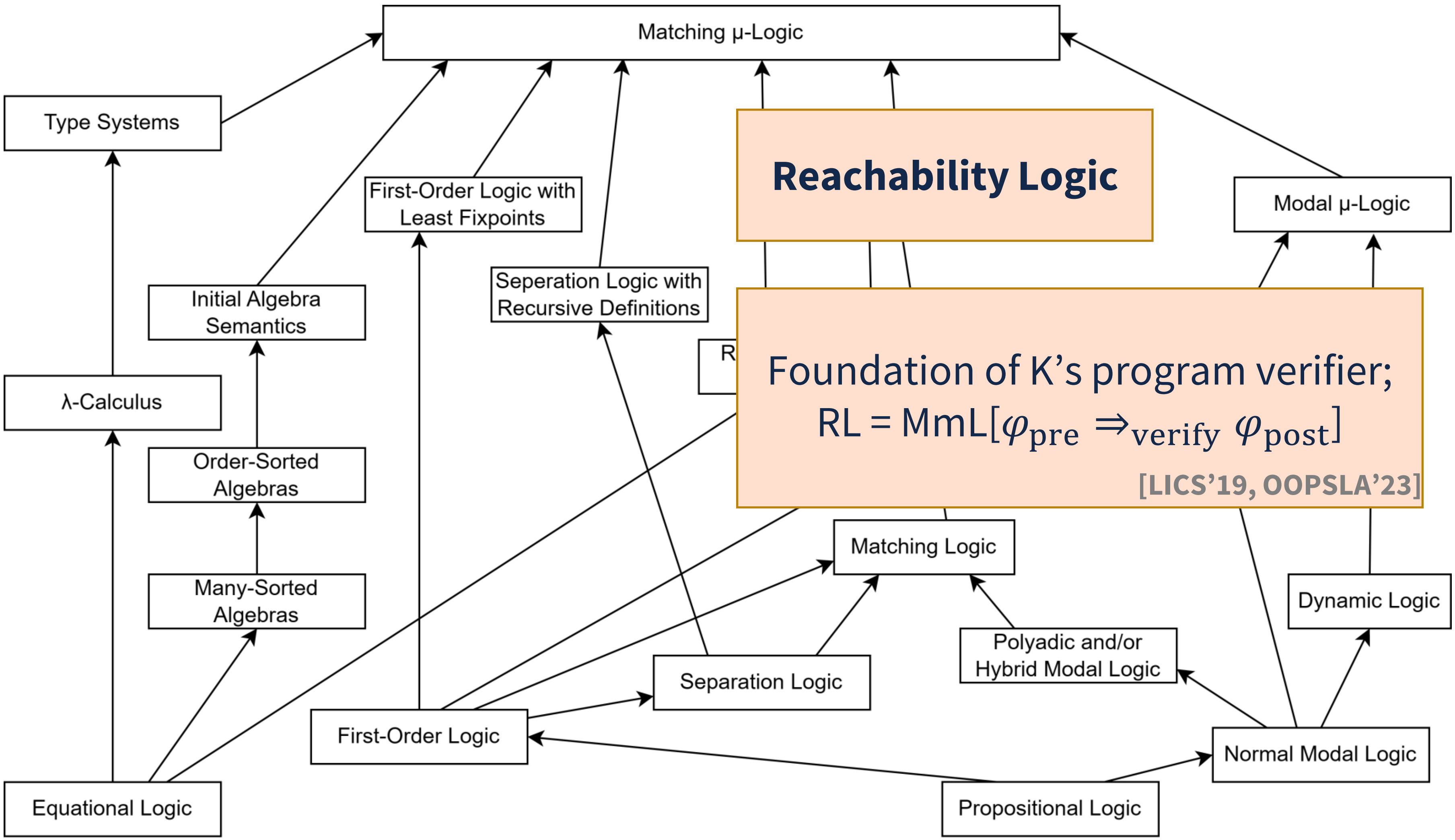
Matching μ-Logic (MmL) Expressive Power

[LICS'19, OOPSLA'20, ICFP'20, CAV'21, JLAMP'21, JLAMP'22, OOPSLA'23]



Matching μ-Logic (MmL) Expressive Power

[LICS'19, OOPSLA'20, ICFP'20, CAV'21, JLAMP'21, JLAMP'22, OOPSLA'23]



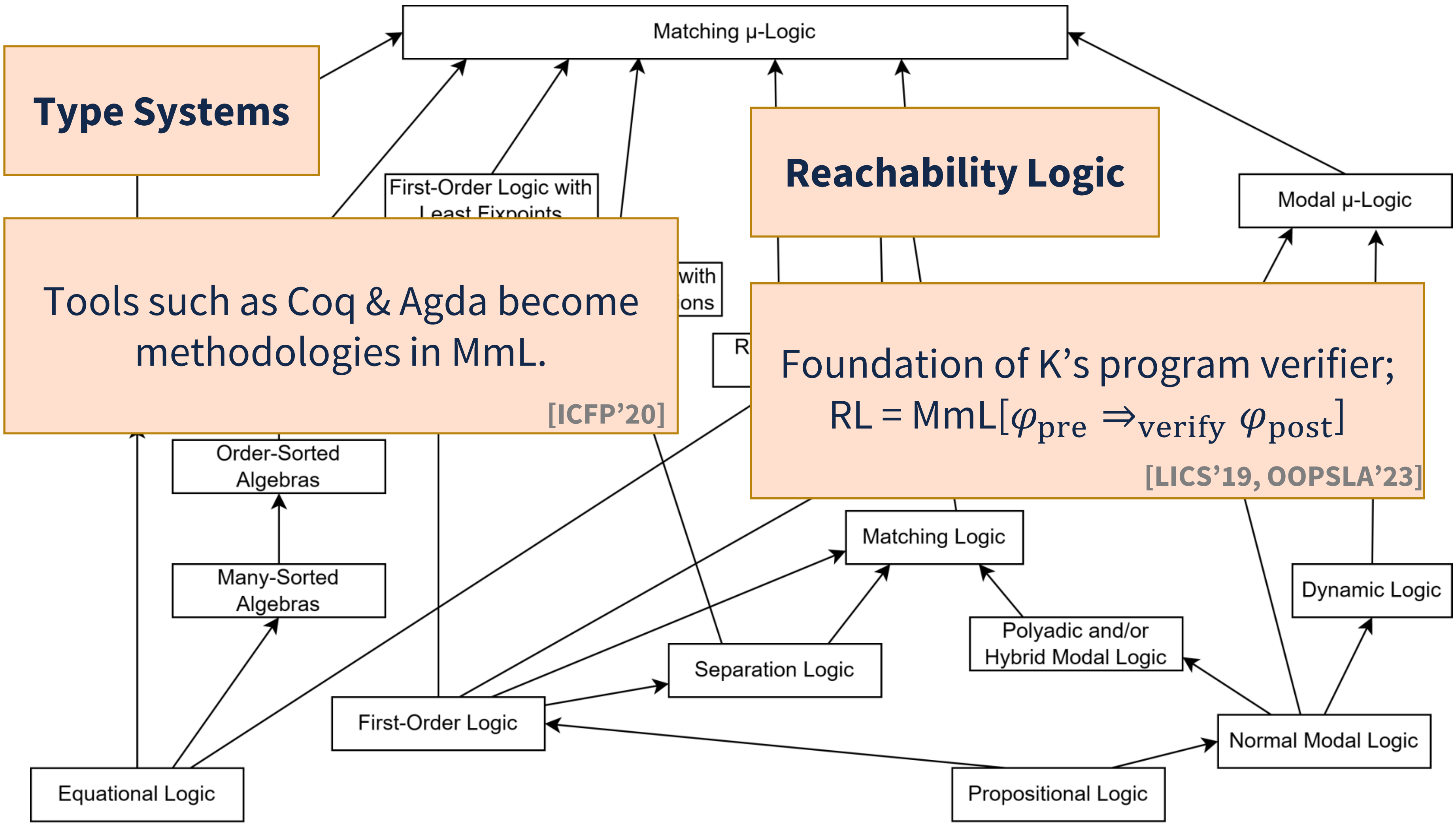
Reachability Logic

Foundation of K's program verifier;
RL = MmL[$\varphi_{\text{pre}} \Rightarrow_{\text{verify}} \varphi_{\text{post}}$]

[LICS'19, OOPSLA'23]

Matching μ-Logic (MmL) Expressive Power

[LICS’19, OOPSLA’20, ICFP’20, CAV’21, JLAMP’21, JLAMP’22, OOPSLA’23]



Type Systems

Reachability Logic

Tools such as Coq & Agda become methodologies in MmL.

[ICFP’20]

Foundation of K’s program verifier;
RL = MmL[$\varphi_{\text{pre}} \Rightarrow_{\text{verify}} \varphi_{\text{post}}$]

[LICS’19, OOPSLA’23]

Matching μ-Logic Proof System

(only 15 proof rules)

FOL 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)	$\dfrac{\varphi \to \psi}{(\exists x.\, \varphi) \to \psi}\ x \notin FV(\psi)$
Frame Rules	(Propagation$_\bot$)	$C[\bot] \to \bot$
	(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)	$\dfrac{\varphi \to \psi}{C[\varphi] \to C[\psi]}$
Fixpoint Rules	(Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
	(Prefixpoint)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
	(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{(\mu X.\, \varphi) \to \psi}$
Technical Rules	(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

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

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

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 $\mu N. 0 \vee \mathbf{succ}(N)$ captures all natural numbers.
Step 2. Set the proof goal $\vdash \left(\mu N. 0 \vee \mathbf{succ}(N)\right) \to \psi_P$
Step 3. Apply (**Knaster Tarski**) and get two sub-goals:

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 System

FOL 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)	$\dfrac{\varphi \to \psi}{(\exists x.\, \varphi) \to \psi}\ x \notin FV(\psi)$
Frame Rules	(Propagation$_\bot$)	$C[\bot] \to \bot$
	(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)	$\dfrac{\varphi \to \psi}{C[\varphi] \to C[\psi]}$
Fixpoint Rules	(Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
	(Prefixpoint)	$\varphi[(\mu X.\, \varphi)/X] \to \mu X.\, \varphi$
	(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{(\mu X.\, \varphi) \to \psi}$
Technical Rules	(Existence)	$\exists x.\, x$
	(Singleton)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

(Modus Ponens)

$$\frac{\varphi_1 \qquad \varphi_1 \to \varphi_2}{\varphi_2}$$

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

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

$\ldots$

$\ldots$

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

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

$\ldots$

$\ldots$

#999. $\boldsymbol{\varphi}$

a proof object

Matching μ-Logic Proof Checker

- We use Metamath [Megill & Wheeler] http://metamath.org
  - to encode proof objects &
  - check them automatically
- 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 (checked by Metamath)

Checking proof objects is fast and trustworthy.

Where do Proof Objects Come From?

Q1: Is there always a proof object for a true statement?

- Completeness of matching μ-logic (briefly)

Q2: Can we find proof objects automatically?

- Automatic theorem prover for matching μ-logic (briefly)

Q3: Can we generate proof objects from K?

- Proving the correctness of K

Where do Proof Objects Come From?

Q1: Is there always a proof object for a true statement?

- Completeness of matching μ-logic (briefly)

Q2: Can we find proof objects automatically?

- Automatic theorem prover for matching μ-logic (briefly)

Q3: Can we generate proof objects from K?

- Proving the correctness of K

Completeness of Matching μ-Logic

- Matching μ-logic is incomplete (because of $\exists$ and μ)
- What if there is no μ?
  - (**Local Completeness**)
    $\emptyset \vDash \varphi \quad \Longrightarrow \quad \emptyset \vdash \varphi$
  - (**Definedness Completeness**)
    $\Gamma \vDash \varphi \quad \Longrightarrow \quad \Gamma \vdash \varphi$, if Γ includes definedness/equality.
  - The rest is open problem.
- What if there is no $\exists$?
  - Open problem.

Where do Proof Objects Come From?

Q1: Is there always a proof object for a true statement?

- Completeness of matching μ-logic (briefly)

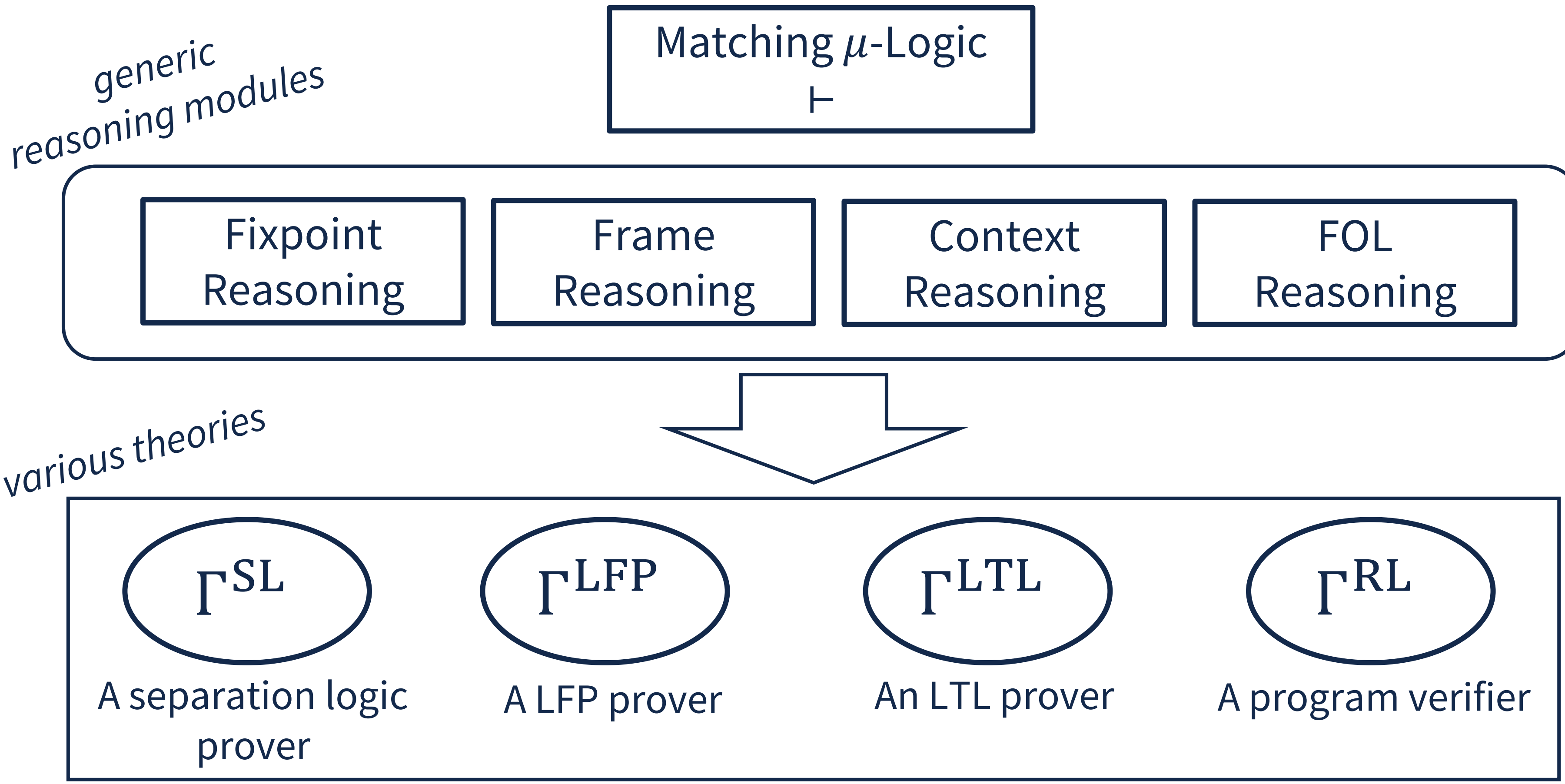
Q2: Can we find proof objects automatically?

- Automatic theorem prover for matching μ-logic (briefly)

Q3: Can we generate proof objects from K?

- Proving the correctness of K

Automatic Theorem Prover for Matching μ-Logic



- Separation logic: Proved 265/280 benchmark tests in SL-COMP'19
- LTL: Proved all the axioms in the complete LTL proof system
- LTP & RL: Proved the correctness of the SUM program

Where do Proof Objects Come From?

Q1: Is there always a proof object for a true statement?

- Completeness of matching μ-logic (briefly)

Q2: Can we find proof objects automatically?

- Automatic theorem prover for matching μ-logic (briefly)

Q3: Can we generate proof objects from K?

- Proving the correctness of K

Overview

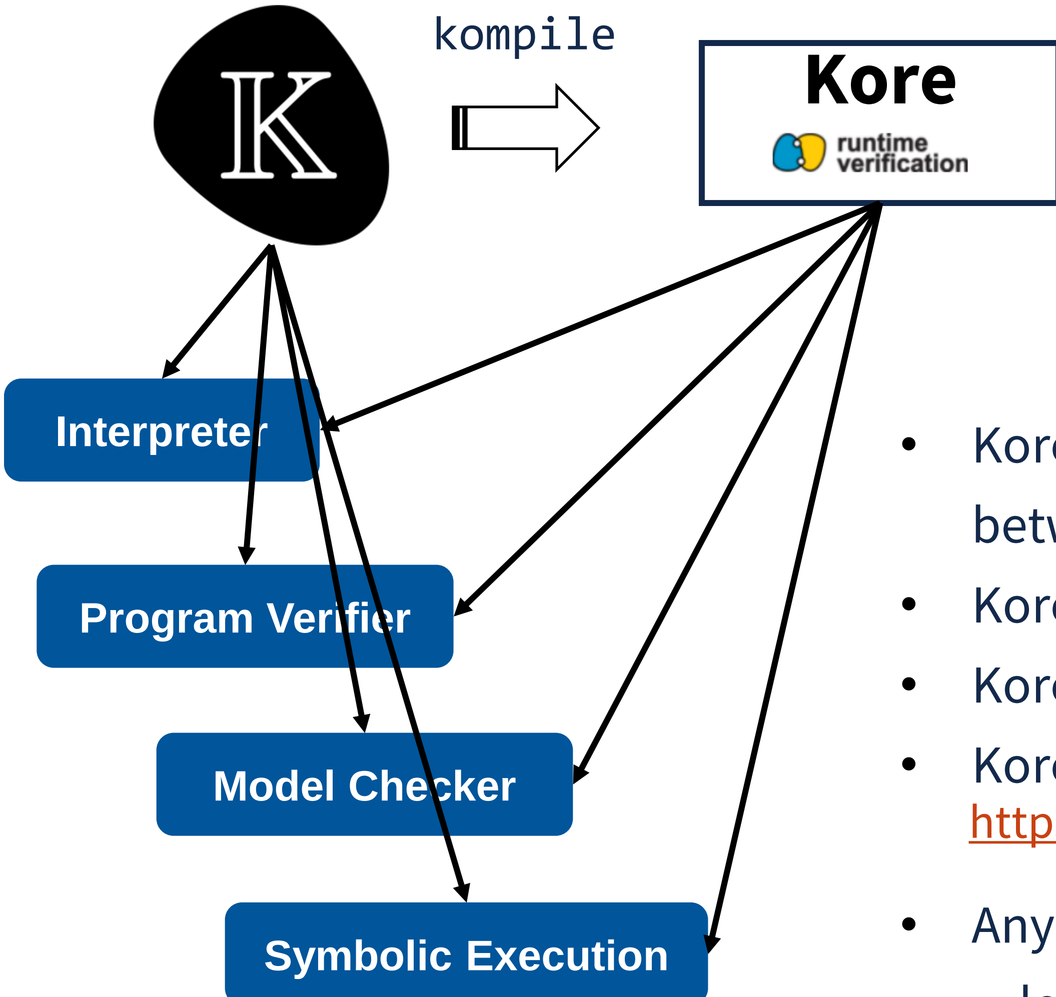
- **Introduction to a Unifying Programming Language Framework**
  - Motivating Example: The K Semantic Framework
  - Research Challenge: Ensure the Correctness of K
- **Main Contribution: Matching μ-Logic**
  - Basic Definitions
  - Expressive Power
  - Proof System and Proof Checker
  - Automatic Theorem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

Translating K to Matching μ-Logic

K	Matching μ-Logic
A PL definition **Ethereum VM**	A *logical theory* Γ^{EVM}
Any PL task • program execution **Interpreter** • formal verification **Program Verifier**	A *theorem* proved by the 15-rule *proof system* • $\Gamma^{\mathrm{EVM}} \vdash t_{\mathrm{init}} \Rightarrow_{\mathrm{exec}} t_{\mathrm{final}}$ • $\Gamma^{\mathrm{EVM}} \vdash \varphi_{\mathrm{pre}} \Rightarrow_{\mathrm{verify}} \varphi_{\mathrm{post}}$
Correctness of the task	Generating the proof and checking it using the 200-LOC *proof checker*

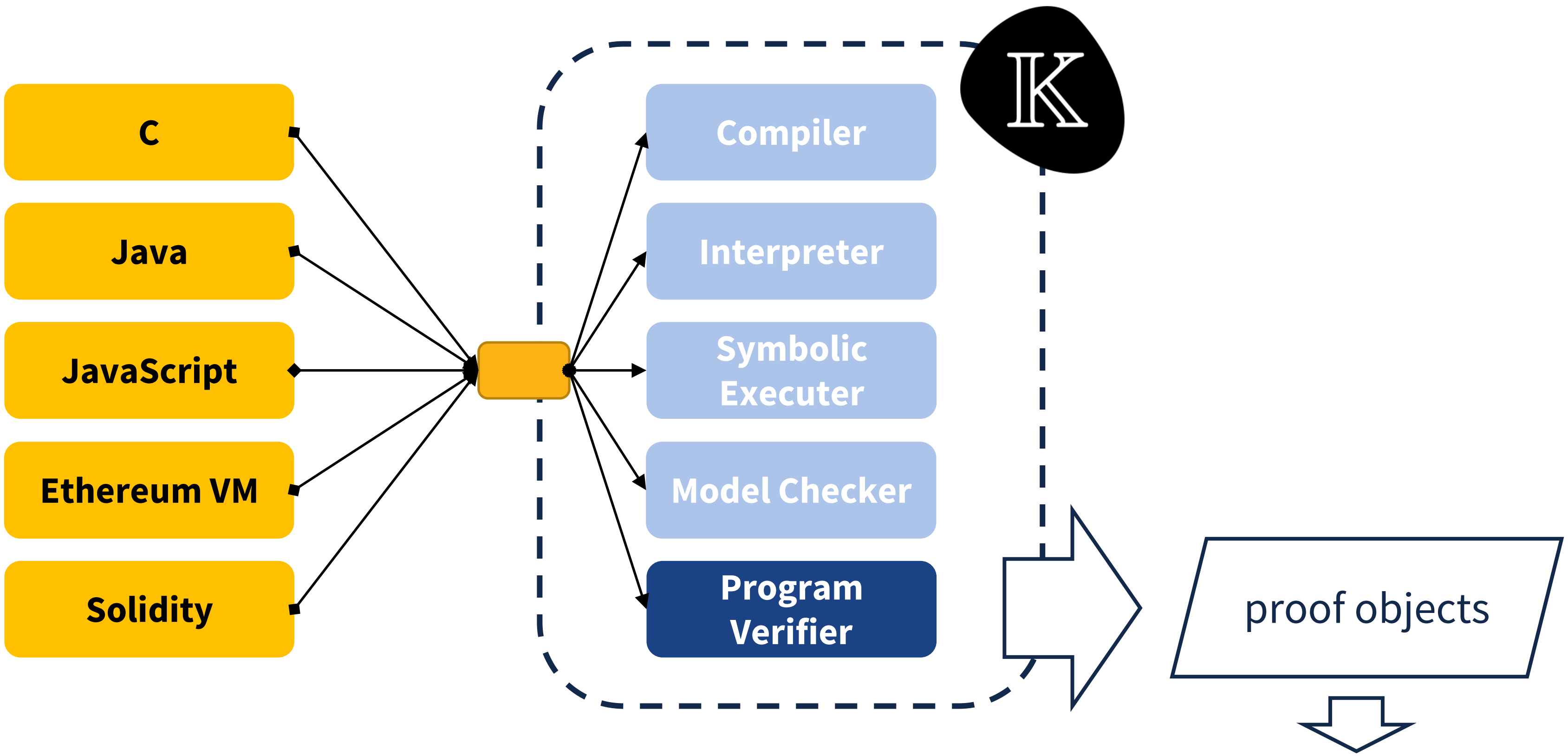
- **Task 1**: Generating the logical theory (e.g., Γ^{EVM})
- **Task 2**: Generating the proof for a given PL task (e.g., verifying a program)

Translating PL Semantics to Matching μ-Logic Theories



- 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

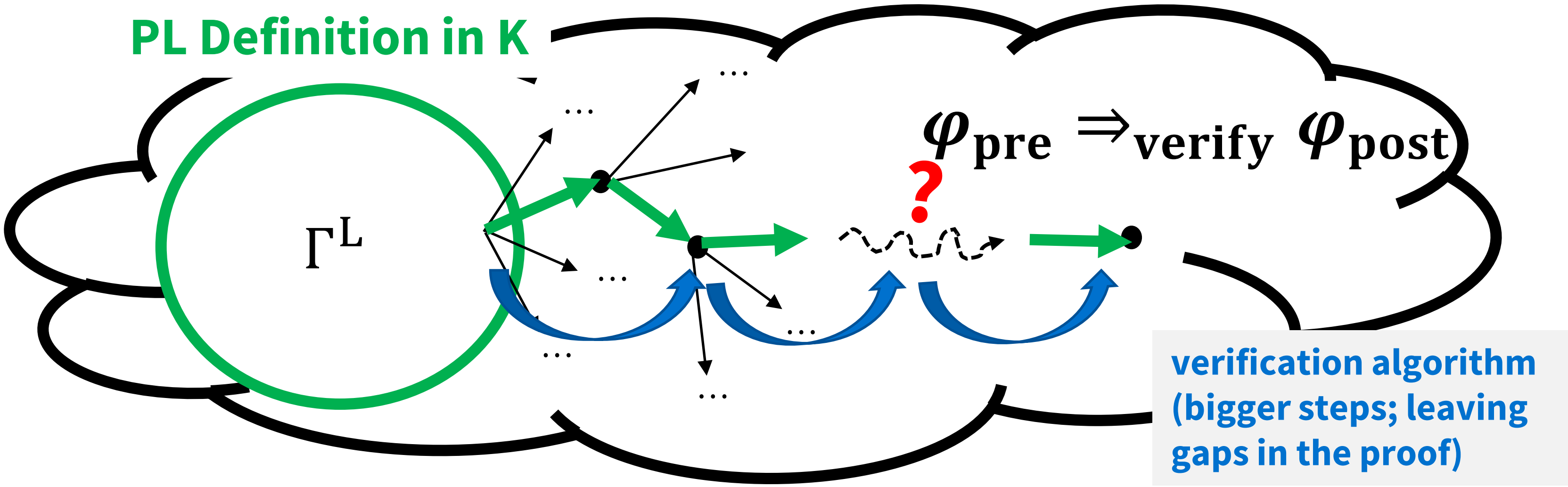
Proving the Correctness of the K Program Verifier



Pain Point: Large code base
>120k lines of Haskell to trust

Solution: Generating and checking proof objects

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

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

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

a proof object

#1. ψ_1
#2. ψ_2
…
#100. ψ_{100}

$\Big\} \; \Gamma^L$

…
…
#247. $\psi_2 \to \varphi$
#247. φ // by (Modus Ponens)
// on #2 and #246
…
…
#99999. $\varphi_{\text{pre}} \Rightarrow_{\text{verify}} \varphi_{\text{post}}$

Proof Generation for Program Verification

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

indeed,
P satisfies φ_P;
check it yourself!

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

proof generation

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

proof checking

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

a proof object

200-LOC matching μ-logic proof checker

Proof Generation: Complicated …

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

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

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

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

…

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

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
					Time (seconds)		
Task	Spec. LOC	Steps	Hint Size	Proof Size	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

Overview

- **Introduction to a Unifying Programming Language Framework**
  - Motivating Example: K Framework
- **Main Contribution: Matching μ-Logic**
  - Expressive Power
  - Proof System and Proof Checker
  - Automatic Proof Framework
- **Using Matching μ-Logic to Prove the Correctness of K**
- **Concluding Remarks**

Matching μ-Logic: A Unifying Foundation for Programming