Rewriting in Matching Logic

Outline

First Session (Matching Logic)

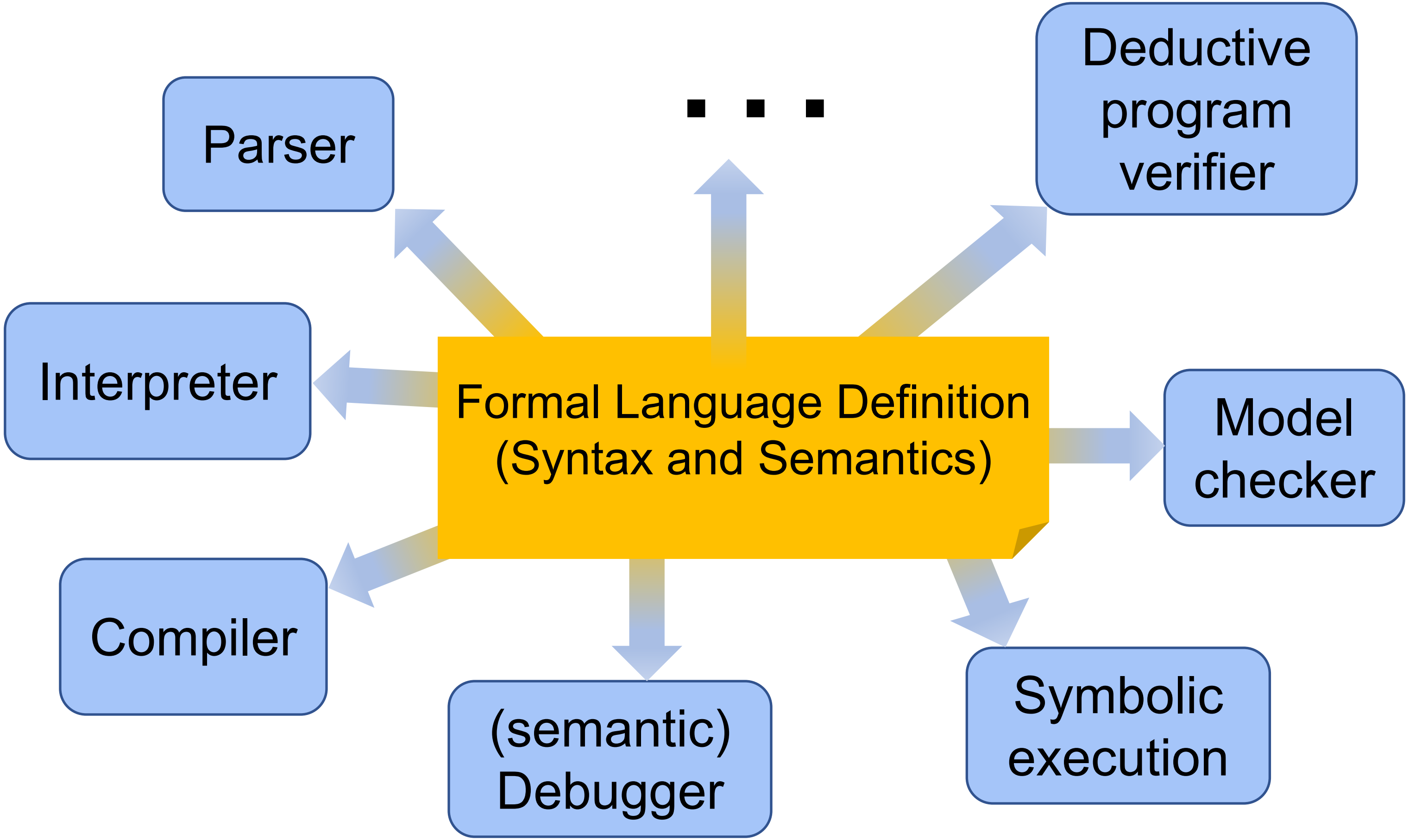
- Ideal Language Framework Vision
- Matching Logic Syntax and Semantics
- Basic Matching Logic Theories
- Matching Logic Theory of Transition Systems
- Matching Logic Proof System

Second Session (Rewriting in Matching Logic)

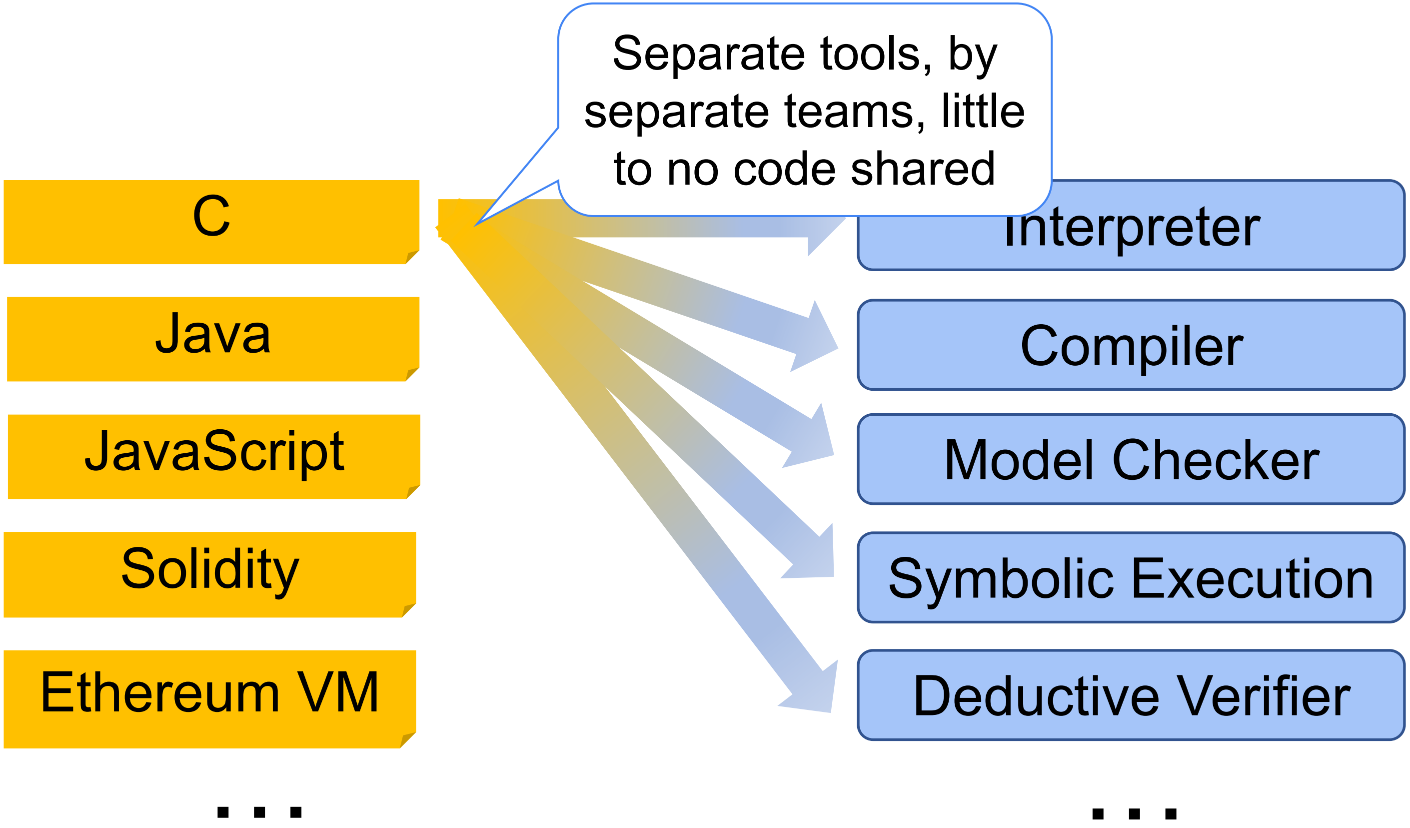
- First-Order Term Unification & Anti-Unification: A Review
- First-Order Term Unification & Anti-Unification in Matching Logic
- Rewriting and Narrowing in Matching Logic

Session 1: Matching Logic

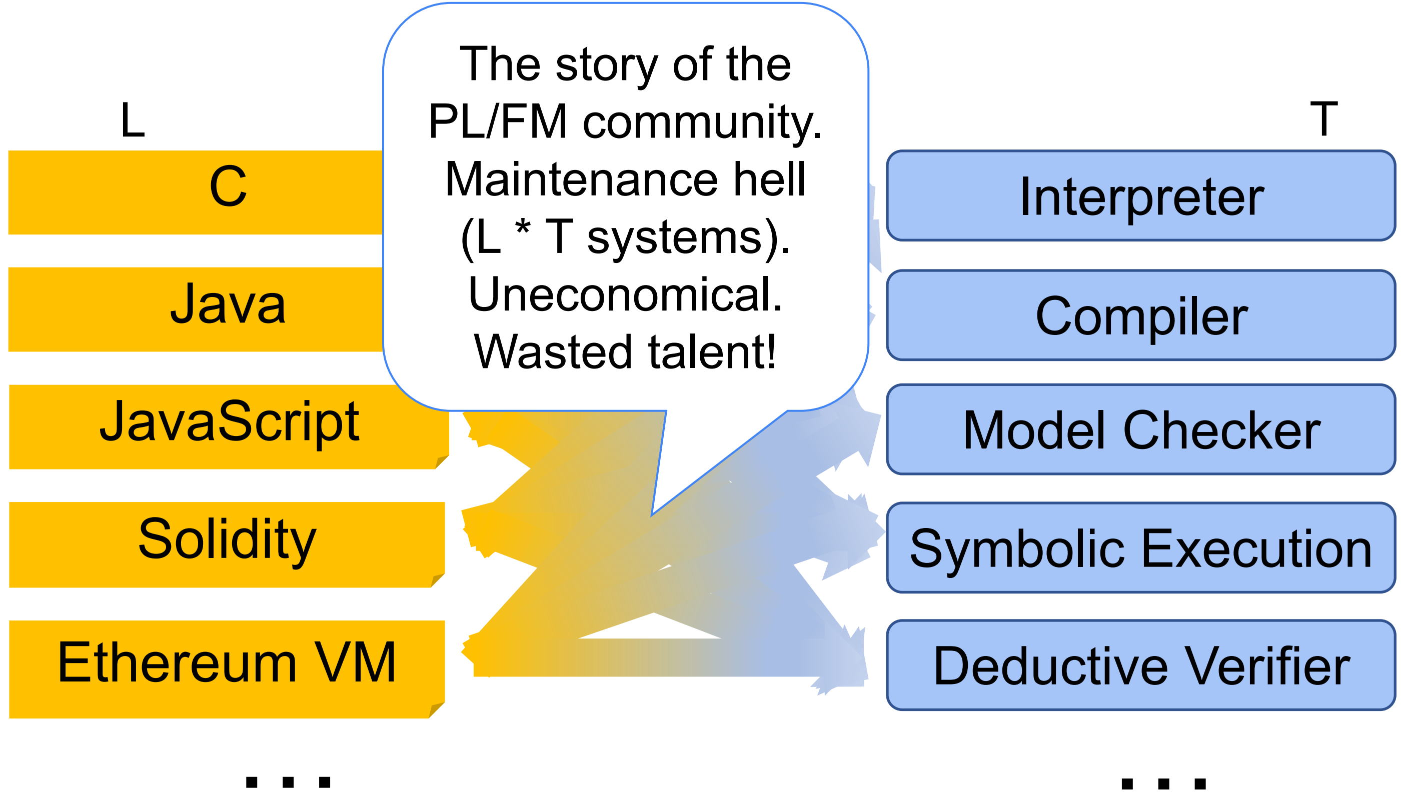
Ideal Language Framework Vision



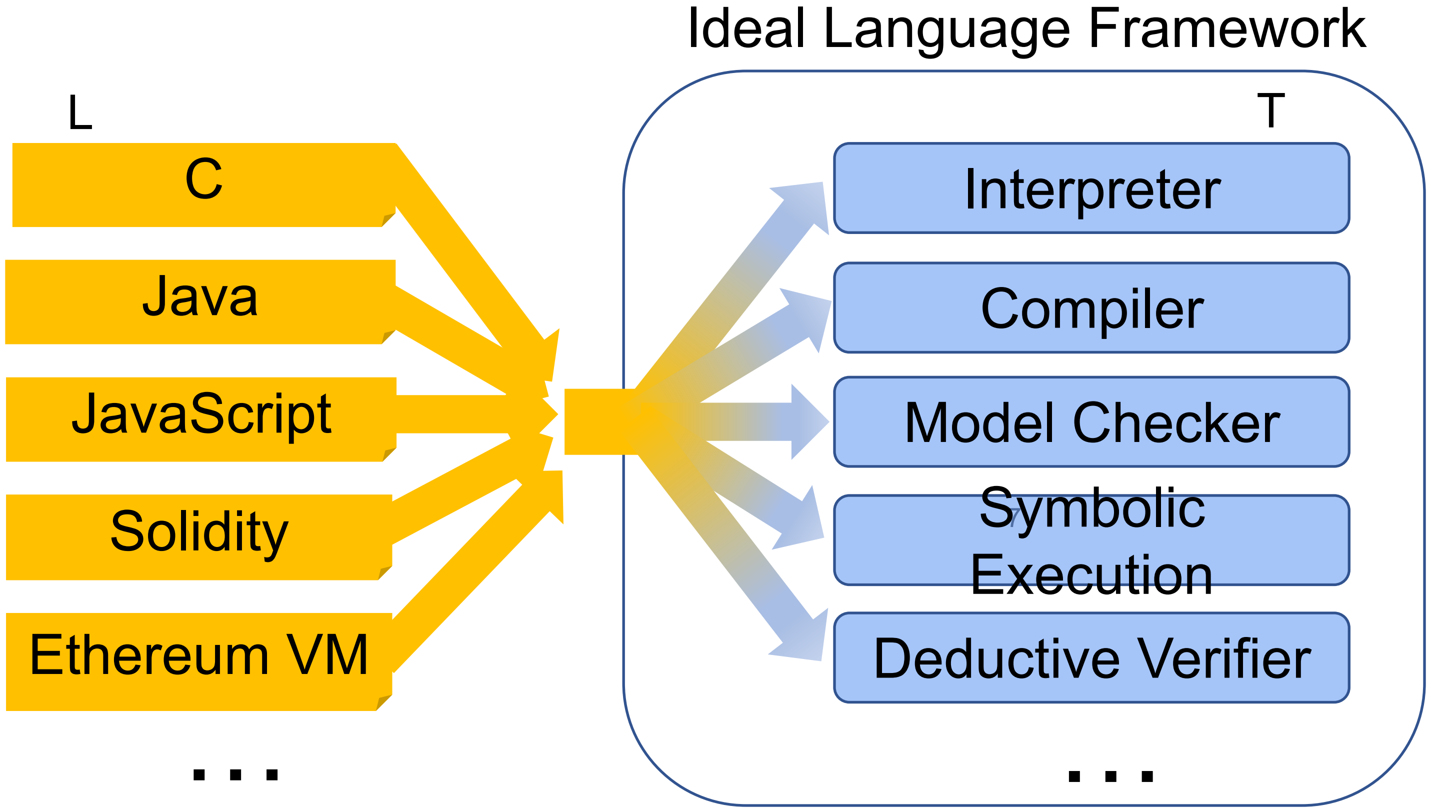
Current State-of-the-Art
- Sharp Contrast to Ideal Vision -



Current State-of-the-Art
- Sharp Contrast to Ideal Vision -



How It Should Be



K Framework
http://kframework.org

- We tried various semantic styles, for >17y and >100 top-tier conference and journal papers:
  - Small-/big-step SOS; Evaluation contexts; Abstract machines (CC, CK, CEK, SECD, …); Chemical abstract machine; Axiomatic; Continuations; Denotational;…
- But each of the above had limitations
  - Especially related to modularity, notation, verification
- K framework initially *engineered*: keep advantages and avoid limitations of various semantic styles
- Then theory was developed: **matching logic**

Matching Logic is the Logical Foundation of K

K	Matching Logic
PL formal definitions (`java.k`)	Logical theories (Γ^{Java})
• PL syntax	• Constructors and terms
• PL semantics and K rewrite rules	• Rewrite axioms
Formal properties of programs	Logical formulas
A language task conducted by K	A proof obligation $\Gamma^{\mathrm{Java}} \vdash \varphi_{\mathrm{task}}$
K does it right!	$\Gamma^{\mathrm{Java}} \vdash \varphi_{\mathrm{task}}$ has a proof object that can be quickly checked by a proof checker (245 LOC)

Outline

- Ideal Language Framework and K
- Matching Logic Syntax
- Matching Logic Semantics
- Basic Matching Logic Theories
- Transition Systems Defined in Matching Logic
- Matching Logic Proof System
- Reading Materials

Matching Logic Syntax

Definition (Patterns)

Let Σ be a set of (constant) symbols, called *signature*. The set of Σ-*patterns* is defined by the following **8** constructs:

$$\varphi ::= x \mid X \mid \sigma \mid \varphi_1\ \varphi_2 \mid \bot \mid \varphi_1 \to \varphi_2 \mid \exists x.\, \varphi \mid \mu X.\, \varphi$$

- $x, y, z, \ldots$ denote *element variables*
- $X, Y, Z, \ldots$ denote *set variables*
- σ is a symbol in Σ
- $\varphi_1\ \varphi_2$ is a binary *application* operation

Matching Logic Syntax

Definition (Patterns)

Let Σ be a set of (constant) symbols, called *signature*. The set of Σ*-patterns* is defined by the following **8** constructs:

$$\varphi ::= x \mid X \mid \sigma \mid \varphi_1\ \varphi_2 \mid \bot \mid \varphi_1 \to \varphi_2 \mid \exists x.\, \varphi \mid \mu X.\, \varphi$$

- $\bot$ and $\varphi_1 \to \varphi_2$ are propositional connectives
- $\exists x.\, \varphi$ is existential quantification
- $\mu X.\, \varphi$, called *least fixpoint pattern*, requires that φ has no negative occurrences of X

Matching Logic Syntax

Common Notation

The following derived constructs are defined as notations:

- $\neg\varphi \equiv \varphi \to \bot$
- $\top \equiv \neg\bot$
- $\varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \to \varphi_2$
- $\varphi_1 \wedge \varphi_2 \equiv \neg\varphi_1 \wedge \neg\varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$
- $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$
- $\nu X.\varphi \equiv \neg\mu X.\neg\varphi[\neg X/X]$

// standard definition of greatest fixpoints

Matching Logic Syntax

Definition (Free Variables)

In matching logic, $\exists x$ and μX are *binders*. Therefore:

- $FreeVars(x) = \{x\}$
- $FreeVars(X) = \{X\}$
- $FreeVars(\sigma) = \emptyset$
- $FreeVars(\varphi_1\ \varphi_2) = FreeVars(\varphi_1) \cup FreeVars(\varphi_2)$
- $FreeVars(\bot) = \emptyset$
- $FreeVars(\varphi_1 \to \varphi_2) = FreeVars(\varphi_1) \cup FreeVars(\varphi_2)$
- $FreeVars(\exists x. \varphi) = FreeVars(\varphi) \setminus \{x\}$
- $FreeVars(\mu X. \varphi) = FreeVars(\varphi) \setminus \{X\}$

Matching Logic Syntax

Example (Variable Capture)

This is wrong:

$$\big(\exists y.\,(x \to y)\big)[y/x] \equiv \exists y.\,\big((x \to y)[y/x]\big) \equiv \exists y.\,(y \to y)$$

This is correct:

$$\big(\exists y.\,(x \to y)\big)[y/x] \equiv \big(\exists z.\,(x \to z)\big)[y/x] \equiv \exists z.\,(y \to z)$$

It is called *capture-avoiding substitution*, where bound variables are renamed to prevent variable capture.

Matching Logic Syntax

Definition (Capture-Avoiding Substitution)

Let $\varphi[\psi/x]$ be the result of substituting ψ for x in φ:

- $x[\psi/x] \equiv \psi$
- $y[\psi/x] \equiv y$ if y distinct from x
- $\sigma[\psi/x] \equiv \sigma$
- $(\varphi_1\varphi_2)[\psi/x] \equiv (\varphi_1[\psi/x])(\varphi_2[\psi/x])$
- $\perp [\psi/x] \equiv \perp$
- $(\varphi_1 \to \varphi_2)[\psi/x] \equiv (\varphi_1[\psi/x]) \to (\varphi_2[\psi/x])$
- $(\exists x. \varphi)[\psi/x] \equiv \exists x. \varphi$
- $(\exists y. \varphi)[\psi/x] \equiv \exists z. (\varphi[z/y][\psi/x])$ where z is fresh
- $(\mu Y. \varphi)[\psi/x] \equiv \mu Z. (\varphi[Z/Y][\psi/x])$ where Z is fresh

Matching Logic Syntax

Definition (Capture-Avoiding Substitution)
Similarly, $\varphi[\psi/X]$ is the result of substituting ψ for X in φ.

Matching Logic Syntax

Summary
- Syntax of patterns (**very simple!**)
- Common notations ($\neg\varphi$, $\varphi_1 \land \varphi_2$, etc.)
- Free variables and capture-avoiding substitution

Next
- Models & semantics of patterns

Questions?

Matching Logic Semantics

Definition (Models)

Let Σ be a signature. A Σ*-model* M is a tuple $\left(M, @_M, \{\sigma_M\}_{\{\sigma\in\Sigma\}}\right)$

- a nonempty *carrier set* M
- an *application function* $@_M : M \times M \to \mathcal{P}(M)$
- a *symbol interpretation* $\sigma_M \subseteq M$ for every $\sigma \in \Sigma$

Unlike FOL, matching logic adopts a *powerset interpretation*. E.g.,

- In FOL: $@_M : M \times M \to M$
- In matching logic: $@_M : M \times M \to \mathcal{P}(M)$

Matching Logic Semantics

Example (Applicative Structures)

An *applicative structure* A is a pair $(A, @_A)$

- a nonempty carrier set A
- an application function $@_A: A \times A \to A$

We can regard A as a matching logic model.

- Let signature $\Sigma = \emptyset$
- Let carrier set $M = A$
- Let $a \ @_M \ b = \{a \ @_A \ b\}$ for all $a, b \in A$

Matching Logic Semantics

Example (Combinatory Algebras)

A *combinatory algebra* A is a tuple $(A, @_A, k_A, s_A)$

- $(A, @_A)$ is an applicative structure
- $k_A, s_A \in A$
- $k_A \, @_A \, a \, @_A \, b = a$ for all $a, b \in A$
- $s_A \, @_A \, a \, @_A \, b \, @_A \, c = (a \, @_A \, c) \, @_A \, (b \, @_A \, c)$ for all $a, b, c \in A$

We can regard A as a matching logic model.

- Let signature $\Sigma = \{k, s\}$ and carrier set $M = A$
- Let symbol interpretations $k_M = \{k_A\}$ and $s_M = \{s_A\}$
- Let $a \, @_M \, b = \{a \, @_A \, b\}$ for all $a, b \in A$

Matching Logic Semantics

- Matching logic adopts a *powerset interpretation*
- FOL adopts a *functional interpretation*
  - which is a special case
- One-to-one correspondence between a and $\{a\}$

Pattern Matching Semantics of Matching Logic

- A pattern φ is evaluated to a *set*
  - a set that includes the elements that *match* it

Matching Logic Semantics

Definition (Variable Valuations)

Given a model M, a variable valuation ρ is a mapping

- $\rho(x) \in M$ for all element variables x
- $\rho(X) \subseteq M$ for all set variables X

Definition (Semantics)

Given M and ρ, a pattern φ is evaluated to a set $|\varphi|_{M,\rho} \subseteq M$.

Matching Logic Semantics

Definition (Semantics)

Given M and ρ, a pattern φ is evaluated to a set $|\varphi|_{M,\rho} \subseteq M$.

- $|x|_{M,\rho} = \{\rho(x)\}$
- $|X|_{M,\rho} = \rho(X)$
- $|\sigma|_{M,\rho} = \sigma_M$
- $|\varphi_1\ \varphi_2|_{M,\rho} = \bigcup_{a_1 \in |\varphi_1|_{M,\rho},\, a_2 \in |\varphi_2|_{M,\rho}} a_1 \mathbin{@_M} a_2$
- $|\perp|_{M,\rho} = \emptyset$
- $|\varphi_1 \to \varphi_2|_{M,\rho} = M \setminus \left(|\varphi_1|_{M,\rho} \setminus |\varphi_2|_{M,\rho}\right)$
- $|\exists x. \varphi|_{M,\rho} = \bigcup_{a \in M} |\varphi|_{M,\rho[a/x]}$
- $|\mu X. \varphi|_{M,\rho} = \mathbf{lfp}\left(A \mapsto |\varphi|_{M,\rho[A/X]}\right)$

Matching Logic Semantics

Semantics of Application $\varphi_1\ \varphi_2$

- $|\varphi_1\ \varphi_2|_{M,\rho} = \bigcup_{a_1 \in |\varphi_1|_{M,\rho},\, a_2 \in |\varphi_2|_{M,\rho}} a_1 \mathbin{@_M} a_2$

Pointwisely extend $@_M : M \times M \to \mathcal{P}(M)$ from elements to sets

- $\overline{@_M} : \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M)$
- $A_1 \mathbin{\overline{@_M}} A_2 = \bigcup_{a_1 \in A_1,\, a_2 \in A_2} a_1 \mathbin{@_M} a_2$ for all $A_1, A_2 \subseteq M$

Simplified: $|\varphi_1\ \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \mathbin{\overline{@_M}} |\varphi_2|_{M,\rho}$

Matching Logic Semantics

Semantics of Propositional connectives $\bot$ and $\varphi_1 \to \varphi_2$

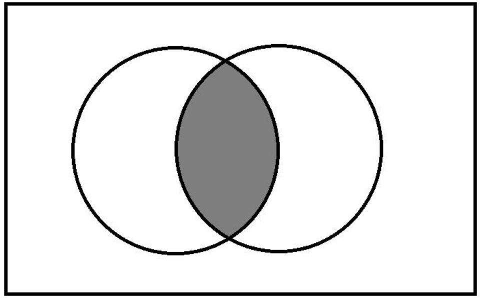
- $|\bot|_{M,\rho} = \emptyset$
- $|\varphi_1 \to \varphi_2|_{M,\rho} = M \setminus \left(|\varphi_1|_{M,\rho} \setminus |\varphi_2|_{M,\rho}\right)$

Propositional Connectives = Set Operations

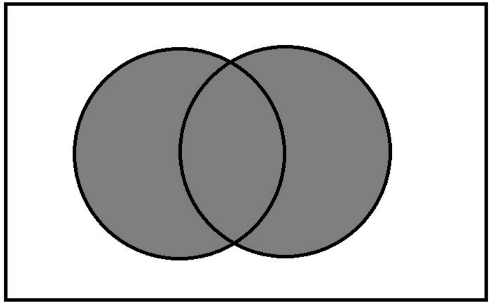
- $|\top|_{M,\rho} = M$
- $|\neg\varphi|_{M,\rho} = M \setminus |\varphi|_{M,\rho}$
- $|\varphi_1 \wedge \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cap |\varphi_2|_{M,\rho}$
- $|\varphi_1 \vee \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cup |\varphi_2|_{M,\rho}$
- $|\varphi_1 \leftrightarrow \varphi_2|_{M,\rho} = M \setminus \left(|\varphi_1|_{M,\rho} \,\Delta\, |\varphi_2|_{M,\rho}\right)$

// set symmetric difference

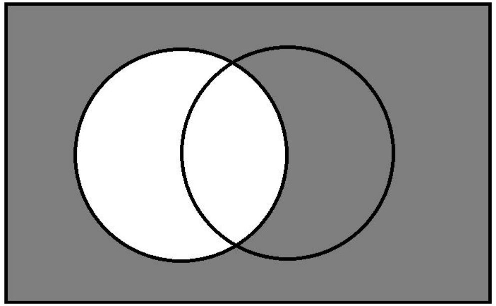
Matching Logic Semantics



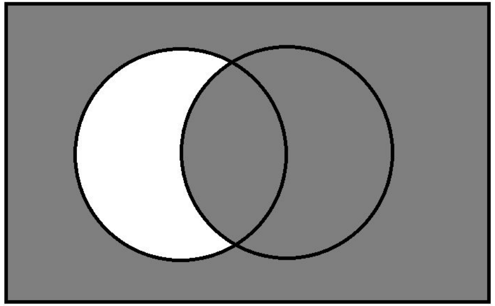
$\varphi_1 \land \varphi_2$



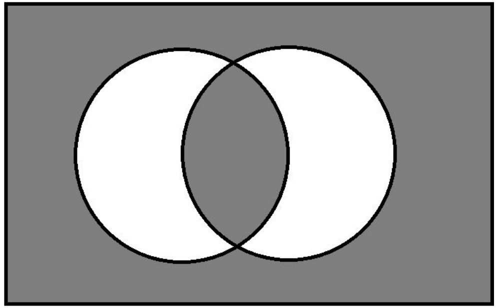
$\varphi_1 \lor \varphi_2$



$\neg\varphi_1$



$\varphi_1 \rightarrow \varphi_2$



$\varphi_1 \leftrightarrow \varphi_2$

Matching Logic Semantics

Semantics of $\exists x.\varphi$ and $\forall x.\varphi$

- $|\exists x.\varphi|_{M,\rho} = \bigcup_{a \in M} |\varphi|_{M,\rho[a/x]}$
- $|\forall x.\varphi|_{M,\rho} = \bigcap_{a \in M} |\varphi|_{M,\rho[a/x]}$ // $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$

Intuitively

- $\exists x.\varphi$ means $\varphi[a_1/x] \vee \varphi[a_2/x] \vee \varphi[a_3/x] \vee \cdots$
- $\forall x.\varphi$ means $\varphi[a_1/x] \wedge \varphi[a_2/x] \wedge \varphi[a_3/x] \wedge \cdots$

Example

- $|\exists x.x|_{M,\rho} = \bigcup_{a \in M} |x|_{M,\rho[a/x]} = \bigcup_{a \in M}\{a\} = M$
- $|\forall x.x|_{M,\rho} = \bigcap_{a \in M} |x|_{M,\rho[a/x]} = \bigcap_{a \in M}\{a\} = \emptyset$ or M

Matching Logic Semantics

Semantics of $\mu X.\varphi$ and $\nu X.\varphi$

- $|\mu X.\varphi|_{M,\rho} = \mathbf{lfp}\left(A \mapsto |\varphi|_{M,\rho[A/X]}\right)$
- $|\nu X.\varphi|_{M,\rho} = \mathbf{gfp}\left(A \mapsto |\varphi|_{M,\rho[A/X]}\right)$

Theorem (Knaster-Tarski)

Let $\mathcal{F}: \mathcal{P}(M) \to \mathcal{P}(M)$ be a *monotone function* (w.r.t. "⊆"). Then $\mathcal{F}$ has the least/greatest fixpoints, given as follows:

- $\mathbf{lfp}\,\mathcal{F} = \bigcap\{\, A \subseteq M \mid \mathcal{F}(A) \subseteq A \,\}$
- $\mathbf{gfp}\,\mathcal{F} = \bigcup\{\, A \subseteq M \mid A \subseteq \mathcal{F}(A) \,\}$

Matching Logic Semantics

Proposition

Since φ has no negative occurrences of X, the following function

$$\mathcal{F}(A) = |\varphi|_{M,\rho[A/X]}$$

is a monotone function (w.r.t. "$\subseteq$").

Proposition

The semantics of $\mu X.\varphi$ is well-defined. In particular,

- $|\mu X.\varphi|_{M,\rho} = \bigcap\{ A \subseteq M \mid |\varphi|_{M,\rho[A/X]} \subseteq A \}$
- $|\nu X.\varphi|_{M,\rho} = \bigcup\{ A \subseteq M \mid A \subseteq |\varphi|_{M,\rho[A/X]} \}$

Matching Logic Semantics

Matching logic has a *pattern matching semantics*.

FOL

- Terms are interpreted as elements
- Formulas are interpreted as true/false

Matching Logic

- No distinction between terms and formulas
- Patterns are interpreted as subsets
- FOL functional interpretation is a special case

Matching Logic Semantics

Truth Values in Matching Logic

- $\top$ and $\bot$
- Semantically, M (the total carrier set) and $\emptyset$ (the empty set)
- Since M is nonempty, we won't confuse $\top$ and $\bot$

Definition (Validity)

Given a pattern set Γ, called a *theory*. For a model M, we write

$$M \vDash \Gamma$$

if for all axioms $\psi \in \Gamma$, $|\psi|_{M,\rho} = M$ for all valuations ρ.

Matching Logic Semantics

Example

A *combinatory algebra* A is a tuple $(A, @_A, k_A, s_A)$

- $k_A \ @_A \ a \ @_A \ b = a$ for all $a, b \in A$
- $s_A \ @_A \ a \ @_A \ b \ @_A \ c = (a \ @_A \ c) \ @_A \ (b \ @_A \ c)$ for all $a, b, c \in A$

We can regard A as a matching logic model. Then,

- $A \vDash kxy \leftrightarrow x$
- $A \vDash sxyz \leftrightarrow (xz)(yz)$

Matching Logic Semantics

Summary

- Models, powerset interpretation
- Pattern matching semantics
- Matching logic theories and validity

Next

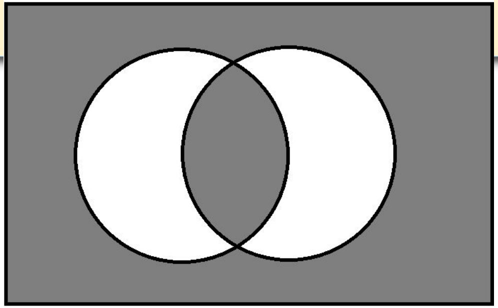
- Basic matching logic theories

Questions?

Theory of Equality

Goal

- To define equality in matching logic
- $\varphi_1 = \varphi_2$
- $\top$, if φ_1 and φ_2 are matched by the same elements
- $\bot$, otherwise
- Note that it is not $\varphi_1 \leftrightarrow \varphi_2$



$\varphi_1 \leftrightarrow \varphi_2$

Theory of Equality

Definition (Definedness)

Let $\lceil \_ \rceil \in \Sigma$ be a symbol, called *definedness*. We write $\lceil \varphi \rceil$ for $\lceil \_ \rceil\ \varphi$.
Add one axiom:

$$(\text{DEFINEDNESS}) \quad \forall x. \lceil x \rceil$$

- Intuitively, $\lceil \varphi \rceil$ states that φ is matched by some elements
  - i.e., is defined (i.e., not $\bot$)
- $\lceil x \rceil$ is $\top$, because x is matched by one element
- $\lceil \bot \rceil$ is $\bot$
- $\lceil \varphi \rceil$ is $\top$, if φ is not $\bot$ // nonempty-ness checking

Theory of Equality

Proposition (Definedness)

For $M \vDash (\mathrm{DEFINEDNESS})$, the following hold:

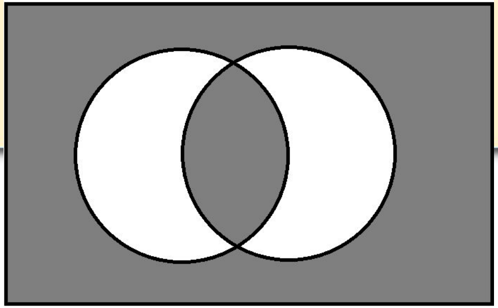
- $|\lceil\varphi\rceil|_{M,\rho} = M$ if $|\varphi|_{M,\rho} \neq \emptyset$
- $|\lceil\varphi\rceil|_{M,\rho} = \emptyset$ if $|\varphi|_{M,\rho} = \emptyset$

- **Totality**, the dual of definedness
- $\lfloor\varphi\rfloor \equiv \neg\lceil\neg\varphi\rceil$, states that φ is *total* (i.e., it is ⊤)
- $|\lfloor\varphi\rfloor|_{M,\rho} = M$ if $|\varphi|_{M,\rho} = M$
- $|\lfloor\varphi\rfloor|_{M,\rho} = \emptyset$ if $|\varphi|_{M,\rho} \neq M$

Theory of Equality

From definedness $\lceil\varphi\rceil$ and totality $\lfloor\varphi\rfloor$, we can define equality $\varphi_1 = \varphi_2$ and many others derived constructs.

- $\varphi_1 \leftrightarrow \varphi_2$ is the complement of set difference between φ_1 and φ_2
- Thus, $\varphi_1 = \varphi_2$ iff $\varphi_1 \leftrightarrow \varphi_2$ is total
- Thus, let $\varphi_1 = \varphi_2 \equiv \lfloor \varphi_1 \leftrightarrow \varphi_2 \rfloor$



$\varphi_1 \leftrightarrow \varphi_2$

Theory of Equality

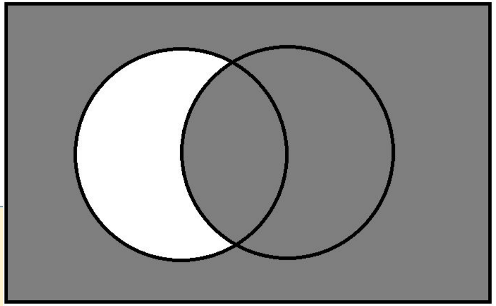
Proposition (Equality)

For $M \vDash (\text{DEFINEDNESS})$, the following hold:

- $|\varphi_1 = \varphi_2|_{M,\rho} = M$ if $|\varphi_1|_{M,\rho} = |\varphi_2|_{M,\rho}$
- $|\varphi_1 = \varphi_2|_{M,\rho} = \emptyset$ if $|\varphi_1|_{M,\rho} \neq |\varphi_2|_{M,\rho}$

- $\varphi_1 = \varphi_2$ is *true* equality (not a congruence)
- Defined *within* logic by axioms/theories (not an extension)

Theory of Equality



$\varphi_1 \to \varphi_2$

Besides equality, we can also define:

- Membership $x \in \varphi \equiv \lceil x \wedge \varphi \rceil$
- Subset relation $\varphi_1 \subseteq \varphi_2 \equiv \lfloor \varphi_1 \to \varphi_2 \rfloor$
- Functional patterns (i.e., terms) $\exists z. (\varphi = z)$

Axiom $(\mathrm{DEFINEDNESS})$ also gives us **equational deduction**

- Using the matching logic proof system (discussed later)

Theory of Sorts

A sort has a sort name and an inhabitant set

- sort name is Nat
- inhabitant set is $\{0,1,2,\dots\}$

Define an *inhabitant symbol* $[\![\_]\!] \in \Sigma$.

Use a symbol s to represent a sort name.
Use $[\![s]\!]$ to represent the inhabitant set.

Theory of Sorts

Example (Natural Numbers)

- Nat represents the sort
- $zero$ and $succ$ represent $\mathbf{0}$ and the successor function
- (ZERO) $\exists z. z \in [\![Nat]\!] \wedge zero = x$
- (SUCC) $\forall x. x \in [\![Nat]\!] \rightarrow \exists z. z \in [\![Nat]\!] \wedge (succ\ x = z)$
- (NAT) $[\![Nat]\!] = \mu D. zero \vee (succ\ D)$

Notation (Sorted Quantification)

- (ZERO) $\exists z{:}\, Nat. zero = x$
- (SUCC) $\forall x{:}\, Nat. \exists z{:}\, Nat. succ\ x = z$

Theory of Sorts

Example (Natural Numbers)

- $(\mathrm{NAT}) \quad [\![Nat]\!] = \mu D.\, zero \vee (succ\ D)$

- $[\![Nat]\!]$ satisfies $[\![Nat]\!] = zero \vee (succ\ [\![Nat]\!])$
- $[\![Nat]\!]$ is the smallest such set (least fixpoint μ)

- Axiom (NAT) also gives us **inductive reasoning**
  - The Peano induction proof rule is derivable from (NAT)

Theory of Sorts

- To state that f is a function from $s_1, \ldots, s_n$ to s

$$\forall x_1 : s_1 \ldots \forall x_n : s_n . \exists y : s . f\ x_1\ \ldots\ x_n = s$$

$$f : s_1 \times \cdots \times s_n \to s$$

- To state that f is a *partial* function from $s_1, \ldots, s_n$ to s

$$\forall x_1 : s_1 \ldots \forall x_n : s_n . \exists y : s . f\ x_1\ \ldots\ x_n \subseteq s$$

- To state that s_1 is a subsort of s_2

$$[\![s_1]\!] \subseteq [\![s_2]\!]$$

- Flexible to capture complex sort structures
  - subsorts, parametric sorts, dependent types/sorts, …

Basic Matching Logic Theories

Summary

- Theory of definedness and equality
- Theory of sorts
- Some axioms about natural numbers

Next

- Theory of *transition systems* and *rewriting*

Questions?

Theory of Transition Systems

Definition (Transition Systems)

A transition system consists of

- A set S of *states*
- A binary *transition relation* $R \subseteq S \times S$

- If $(s, s') \in R$, s' is a *next state* of s; s is a *previous state* of s'
- s is a *terminating* state if it has no next states
- s is a *well-founded* state if it has no infinite traces

Theory of Transition Systems

- Let $State$ be the sort of states in S
- Let $\bullet \in \Sigma$ be a symbol, called *one-path next*

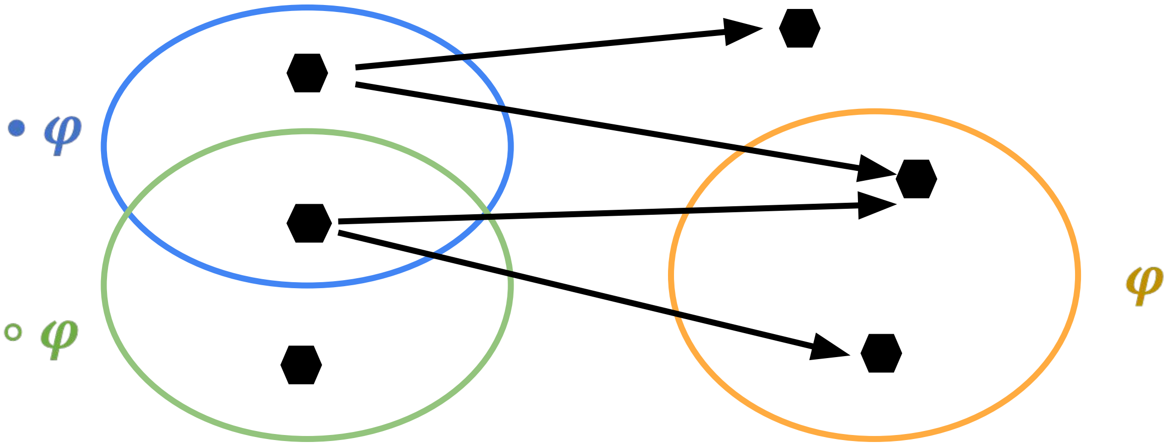
- Intuitively, $\bullet\, \varphi$ is matched by states whose *next states* match φ

$$s \xrightarrow{R} s' \xrightarrow{R} s'' \xrightarrow{R} s''' \qquad \text{// states in } S$$

$$\bullet\bullet\bullet\, \varphi \qquad \bullet\bullet\, \varphi \qquad \bullet\, \varphi \qquad \varphi \qquad \text{// patterns}$$

One-Path Next and All-Path Next

- $\bullet\,\varphi$ is one-path next
- $\circ\,\varphi \equiv \neg_{State} \bullet \neg_{State}\varphi$ is *all-path next*
  - $\neg_{State}\psi \equiv \neg\psi \wedge [\![State]\!]$



Theory of Transition Systems

From one-path next, we can define *temporal operations*

- $\bullet\, \varphi$, one-path next
- $\circ\, \varphi$, all-path next
- $\bullet\, \top$, non-terminating states (has next states)
- $\circ \bot$, terminating states (has no next states)
- $\bullet\bullet\, \varphi$, reaches φ in 2 steps
- $\Diamond\varphi \equiv \mu X.\ \varphi \vee \bullet X$, eventually φ
- $\Box\varphi \equiv \nu X\ .\varphi \wedge \circ X$, always φ
- $\varphi_1\ U\ \varphi_2 \equiv \mu X. \varphi_2 \vee (\varphi_1 \wedge \bullet X)$, "until"
- $WF \equiv \mu X. \circ X$, well-founded states

Theory of Transition Systems

From one-path next, we can define *temporal axioms*

- (FIN) $\forall s{:}\, State.\ s \in WF$
- (LIN) $\forall s{:}\, State.\ \bullet s \rightarrow \circ s$
- (INF) $\forall s{:}\, State.\ s \in \bullet \top$

Theorem (*Matching* μ *-Logic* [LICS 2019]**)**

- Linear temporal logic (LTL) is $(\mathrm{LIN}) + (\mathrm{INF})$.
- Finite-trace LTL is $(\mathrm{LIN}) + (\mathrm{FIN})$.
- Computation tree logic (CTL) is (INF).
- Modal μ-calculus is the empty theory over "$\bullet$".

Theory of Transition Systems

From one-path next, we can define *rewriting*

- $\varphi_1 \Rightarrow^1 \varphi_2 \equiv \varphi_1 \rightarrow \bullet\, \varphi_2$ // one-step rewriting
- $\varphi_1 \Rightarrow \varphi_2 \equiv \varphi_1 \rightarrow \lozenge \varphi_2$ // zero or more step(s) rewriting
- $\varphi_1 \Rightarrow^+ \varphi_2 \equiv \varphi_1 \rightarrow \bullet \lozenge \varphi_2$ // one or more steps rewriting

Theory of Transition Systems

Summary

- One-path next
- Other temporal operations as derived constructs
- Axioms $(\mathrm{LIN}), (\mathrm{FIN}), (\mathrm{INF})$
- Rewriting $\varphi_1 \Rightarrow \varphi_2 \equiv \varphi_1 \to \Diamond\varphi_2$

Next

- Matching logic proof system

Questions?

Matching Logic Proof System

- A Hilbert proof system
- 13 proof rules
- Simple

- $\Gamma \vdash \varphi$
- φ can be proved, with additional axioms in Γ

FOL Reasoning	(Propositional Tautology)	φ if φ is a tautology over patterns
	(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}$
	($\exists$-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$
	($\exists$-Generalization)	$\dfrac{\varphi_1 \to \varphi_2}{(\exists x. \varphi_1) \to \varphi_2}$ if $x \notin FV(\varphi_2)$
Frame Reasoning	(Propagation$_\bot$)	$C[\bot] \to \bot$
	(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
	(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ if $x \notin FV(C)$
	(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$
Fixpoint Reasoning	(Set Variable 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)	$\neg (C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

FOL Reasoning

(Propositional Tautology)	φ	if φ is a tautology over patterns
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}$	
($\exists$-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$	
($\exists$-Generalization)	$\dfrac{\varphi_1 \to \varphi_2}{(\exists x.\varphi_1) \to \varphi_2}$	if $x \notin FV(\varphi_2)$

- Standard FOL proof rules
- Sound w.r.t. the powerset interpretation

Frame Reasoning

(Propagation$_\bot$)	$C[\bot] \to \bot$
(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ if $x \notin FV(C)$
(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$

Definition (Application Contexts)

A *context* C is a pattern with one placeholder variable $\square$.

We write $C[\psi] \equiv C[\psi/\square]$ for *context plugging*

C is an *application context*, if from root to $\square$ there are only applications

Frame Reasoning

(Propagation$_\bot$)	$C[\bot] \to \bot$
(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ if $x \notin FV(C)$
(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$

Semantically, **frame reasoning** = the **pointwise extension of applications**

$$|\varphi_1\ \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho}\ \overline{@_M}\ |\varphi_2|_{M,\rho}$$

Frame Reasoning

(Propagation$_\bot$)	$C[\bot] \to \bot$
(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi] \quad \text{if } x \notin FV(C)$
(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$

(Framing) can be generalized to any *positive contexts* C

- E.g., $\vdash \varphi_1 \to \varphi_2$ implies $\vdash \bullet\, \varphi_1 \to \bullet\, \varphi_2$
- Also implies $\vdash \Diamond\varphi_1 \to \Diamond\varphi_2$, because $\Diamond\varphi \equiv \mu X.\ \varphi \vee \bullet\, X$ is positive w.r.t. φ

Frame Reasoning

($\text{Propagation}_\perp$)	$C[\perp] \to \perp$
($\text{Propagation}_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
($\text{Propagation}_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi] \quad \text{if } x \notin FV(C)$
(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$

do reasoning logically

then generalize it to larger contexts

- (**Framing**) is natural in terms of semantics
- (**Framing**) works for both structures and dynamic relations
- Allows us to bring local reasoning to the top; very useful in practice

Fixpoint Reasoning

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

- Standard fixpoint proof rules as in modal μ-calculus
- (Fixpoint) $\varphi[(\mu X.\,\varphi)/X] \leftrightarrow \mu X.\,\varphi$
- "$\to$" is (PreFixpoint)
- "$\leftarrow$" is derivable from (Knaster Tarski), shown later

Fixpoint Reasoning

(Set Variable Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$	
(PreFixpoint)	$\varphi[(\mu X.\,\varphi)/X] \to \mu X.\,\varphi$	
(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{\mu X.\,\varphi \to \psi}$	if ψ is a prefixpoint then the lfp is smaller than ψ

- (Knaster Tarski) is a direct encoding of the Knaster-Tarski Fixpoint Theorem
- $|\mu X.\,\varphi|_{M,\rho} = \bigcap\{ A \subseteq M \mid |\varphi|_{M,\rho[A/X]} \subseteq A \}$
- Now, take A be (the evaluation of) ψ

Fixpoint Reasoning

Example (Prove $\vdash (\mu X.\varphi) \to \varphi[(\mu X.\varphi)/X]$**)**

1. $\vdash \varphi[\varphi[(\mu X.\varphi)/X]/X] \to \varphi[(\mu X.\varphi)/X]$
2. φ is a positive context w.r.t. X
3. $\vdash \varphi[(\mu X.\varphi)/X] \to \mu X.\varphi$ // (Framing)
4. This is (PreFixpoint), QED

$$\text{(PreFixpoint)} \quad \varphi[(\mu X.\varphi)/X] \to \mu X.\varphi$$

$$\text{(Knaster-Tarski)} \quad \frac{\varphi[\psi/X] \to \psi}{\mu X.\varphi \to \psi}$$

Fixpoint Reasoning

(Knaster Tarski) gives the **principle of induction**.

- $[\![Nat]\!] = \mu D.zero \vee (succ\ D)$

$$\frac{zero \rightarrow \Psi \qquad (succ\ \Psi) \rightarrow \Psi}{[\![Nat]\!] \rightarrow \Psi} \quad \text{(Knaster-Tarski)}$$

- This is Peano induction. To prove all natural numbers satisfy Ψ
  1. Prove that $zero$ satisfies Ψ
  2. Prove that if n satisfies Ψ, so does $(succ\ n)$

Technical Rules

(Existence)	$\exists x.\, x$
(Singleton)	$\neg\,(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

Theorem (Completeness)
In the fixpoint-free fragment, $\vDash \varphi$ implies $\vdash \varphi$.

- $|\exists x.\, x|_{M,\rho} = \bigcup_{a \in M} |x|_{M,\rho[a/x]} = \bigcup_{a \in M} \{a\} = M$
- Since x is one element, one of $x \wedge \varphi$ and $x \wedge \neg\varphi$ is $\bot$

Matching Logic Proof System

Theorem (Equational Deduction)

The following equational proof rules are derivable:

- $\vdash \varphi = \varphi$
- $\vdash \varphi_1 = \varphi_2$ implies $\vdash \varphi_2 = \varphi_1$
- $\vdash \varphi_1 = \varphi_2$ and $\vdash \varphi_2 = \varphi_3$ imply $\vdash \varphi_1 = \varphi_3$
- $\vdash \varphi_1 = \varphi_2$ implies $\vdash C[\varphi_1] = C[\varphi_2]$
- $\vdash \varphi_1 = \varphi_2$ implies $\vdash \varphi_1[y/x] = \varphi_2[y/x]$

Matching Logic Proof System

Summary

- A simple proof system
- 4 categories of rules

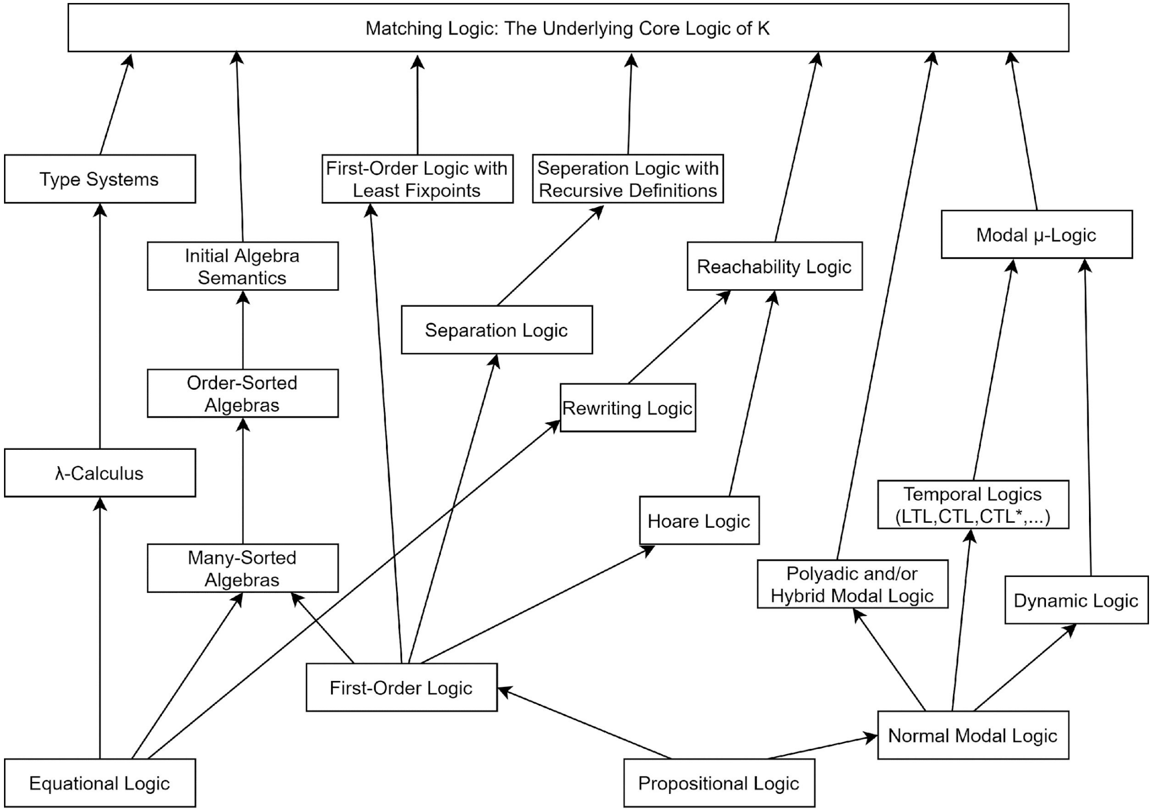
- A small **proof checker**
- Encode $\Gamma \vdash \varphi$ into a **proof object**

FOL Reasoning	(Propositional Tautology)	φ if φ is a tautology over patterns
	(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}$
	($\exists$-Quantifier)	$\varphi[y/x] \to \exists x.\, \varphi$
	($\exists$-Generalization)	$\dfrac{\varphi_1 \to \varphi_2}{(\exists x. \varphi_1) \to \varphi_2}$ if $x \notin FV(\varphi_2)$
Frame Reasoning	(Propagation$_\bot$)	$C[\bot] \to \bot$
	(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
	(Propagation$_\exists$)	$C[\exists x.\, \varphi] \to \exists x.\, C[\varphi]$ if $x \notin FV(C)$
	(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$
Fixpoint Reasoning	(Set Variable 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)	$\neg\,(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

Matching Logic

- One logic
- One proof system
- One proof checker
- Many theories

- Use **notations** to handle encoding cost
- Use **lemmas** to handle reasoning cost



Reading List

Core Papers

- *Matching Logic* by G. Rosu, LMCS 2017
- *Matching mu-Logic* by X. Chen & G. Rosu, LICS 2019
- *Matching Logic Explained* by X. Chen, D. Lucanu & G. Rosu, JLAMP 2020

Defining transition systems

- *Sec. 7&8 of Matching mu-Logic* by X. Chen & G. Rosu, LICS 2019

Defining unification

- *Unification in Matching Logic* by A. Arusoaie & D. Lucanu, FM 2019

Defining type systems

- *A General Approach to Define Binders using Matching Logic* by X. Chen & G. Rosu, ICFP 2020

Defining initial algebra semantics

- *Initial Algebra Semantics in Matching Logic* by X. Chen, D. Lucanu & G. Rosu, TechRep (http://hdl.handle.net/2142/107781) 2020

Automated matching logic prover

- *Towards a Unified Proof Framework for Automated Fixpoint Reasoning using Matching Logic* by X. Chen et al., OOPSLA 2020

Matching logic proof checker

- *Towards a Trustworthy Semantics-Based Language Framework via Proof Generation* by X. Chen et al., CAV 2021

Session 2: Unification & Antiunification

Outline

- Introduction
- First-order Term Unification
- First-order Term Unification in Matching Logic
- First-order Term Anti-Unification
- First-order Term Anti-Unification in Matching Logic
- Conclusion

Motivation

- The semantics of the programming languages is usually given by rule patterns of the form
$$t_i \wedge \phi_i \to \bullet(t_i' \wedge \phi_i')$$
where t_i, t_i' are term patterns and ϕ, ϕ_i' are predicate patterns (constraints).
Example:
$(\langle \texttt{if}\ (B)\ S_1\ \texttt{else}\ S_2 \rightsquigarrow S, \sigma\rangle \wedge \sigma(B) = true) \to \bullet\langle S_1 \rightsquigarrow S, \sigma\rangle$
$(\langle \texttt{if}\ (B)\ S_1\ \texttt{else}\ S_2 \rightsquigarrow S, \sigma\rangle \wedge \sigma(B) = false) \to \bullet\langle S_2 \rightsquigarrow S, \sigma\rangle$
- Assume a language L defined by just two rules
$$t_1 \wedge \phi_1 \to \bullet(t_1' \wedge \phi_1')$$
$$t_2 \wedge \phi_2 \to \bullet(t_2' \wedge \phi_2')$$
- A symbolic step $t \wedge \phi \Rightarrow t' \wedge \phi'$ is characterized by the following properties:
$$(t \wedge \phi \wedge t_1 \wedge \phi_1) \vee (t \wedge \phi \wedge t_2 \wedge \phi_2) \to \circ(t' \wedge \phi')$$
$$t' \wedge \phi' \to (t_1' \wedge \phi_1') \vee (t_2' \wedge \phi_2')$$
- The configuration $t' \wedge \phi'$ can be computed using (anti)unification algorithms.

Problem

- The (anti)unification algorithms work on the term algebra.
  We need an axiomatization of the term algebra in Matching Logic.
- A unification algorithm computes the most general unifier (mgu).
  We need to characterize the mgu in Matching Logic.
- An anti-unification algorithm computes the least general generalization (lgg).
  We need to characterize the lgg in Matching Logic.
- How to transform the execution of an algorithm into a ML proof?
- What is the minimal set of lemmas needed to handle the reasoning effort?

Term Algebra in ML (Example)

spec *LISTofNAT*

Symbols : $inh, Nat, List, zero, succ, nil, cons$

Notations :

$[\![\varphi]\!] \equiv inh\,\varphi$

$\exists x{:}s.\varphi \equiv \exists x.x \in [\![s]\!] \wedge \varphi$

$\forall x{:}s.\varphi \equiv \forall x.x \in [\![s]\!] \to \varphi$

Axioms :

(Inductive Domain) : $[\![Nat]\!] = \mu N.zero \vee succ\ X$
$[\![List]\!] = \mu L.nil \vee cons\ [\![Nat]\!]\ L$

(Function) : $\exists y.y \in [\![Nat]\!] \wedge zero = y,$
$\forall x.x \in [\![Nat]\!] \to \exists y.y \in [\![Nat]\!] \wedge succ\ x = y;$
$\exists y.y \in [\![List]\!] \wedge nil = y,$
$\forall x.x \in [\![Nat]\!] \wedge l \in [\![List]\!] \to \exists y.y \in [\![List]\!] \wedge cons\ x\ l = y$

(NoConfusion I) : $zero \neq nil$
$\forall x{:}Nat.\forall l{:}List.zero \neq cons\ x\ l$
$\forall x{:}Nat.zero \neq succ\ x$
$\forall l{:}List.nil \neq succ\ x$
$\forall x{:}Nat.\forall l{:}List.nil \neq cons\ x\ l$
$\forall n{:}Nat.\forall x{:}Nat.\forall l{:}List.succ\ n \neq cons\ x\ l$

(NoConfusion II) : $\forall x{:}Nat.\forall x'{:}Nat.succ\ x = succ\ x' \to x = x'$
$\forall x, x'{:}Nat.\forall l, l'{:}List.cons\, x\, l = cons\, x'\, l' \to x = x' \wedge l = l'$

endspec

LISTofNAT in Maude

```
fmod LISTofNAT is
  sorts Nat List .
  op zero : -> Nat [ctor] .
  op succ : Nat -> Nat [ctor] .
  op nil : -> List [ctor] .
  op cons : Nat List -> List [ctor] .
endfm
```

Annotation semantics:
ctor : No Confusion I + II +
Inductive Domain (No Junk)

fmod-endfm (initial semantics): it is a consequence of the ML specification

Lemmas for handling the reasoning effort

Proof System = ML Proof System +

∃-Subst	$\exists z.t \wedge (z = u) \leftrightarrow t[u/z]$, if $z \notin var(u)$
∃-Gen	$z = (f\,\overline{t}) \leftrightarrow \exists \overline{y}.z = (f\,\overline{y}) \wedge \overline{y} = \overline{t}$, if $\overline{y} \notin var((f\,\overline{t})) \cup \{z\}$
¬Occrs	$(x = t) \leftrightarrow \bot$, if $x \in var(t)$

Fig. 9: A particular set of proof rules used holding in term algebra. Here, $\overline{t}$ is a placeholder for $t_1 \ldots t_n$, $\overline{y}$ is a placeholder for $y_1 \ldots, y_n$, and $\overline{y} = \overline{t}$ stands for $\bigwedge_{j=1}^{n} y_j = t_j$.

∃-Context	$\dfrac{\varphi_2 \leftrightarrow \varphi_2'}{(\exists \overline{x}.\varphi_1 \wedge \varphi_2) \leftrightarrow \exists \overline{x}.\varphi_1 \wedge \varphi_2'}$
∃-Scope	$((\exists \overline{x}.\varphi_1) \odot \varphi_2) \leftrightarrow \exists \overline{x}.\varphi_1 \odot \varphi_2$, if $\overline{x} \notin free(\varphi_2)$
∃-Collapse	$((\exists \overline{x}.\varphi_1) \vee (\exists \overline{x}.\varphi_2)) \leftrightarrow \exists \overline{x}.\varphi_1 \vee \varphi_2$

A particular set of derived proof rules used to generate certificates for anti-unification.

What’s next

- Definitions
- Martelli-Montanari unification algorithm
- First-order Term Unification in Matching Logic
- First-order Term Anti-Unification
- First-order Term Anti-Unification in Matching Logic
- Conclusion

First-order term unification - Definitions

Definition

A substitution σ is a unifier of t and t' if $t\sigma = t'\sigma$

$$
\begin{tikzcd}
t \arrow[dr, "\sigma"'] & & t' \arrow[dl, "\sigma"] \\
& u &
\end{tikzcd}
$$

First-order term unification - Example

$$
\begin{tikzcd}[column sep=small]
(cons\ (succ\ x)\ (cons\ y\ l)) \arrow[dr, "\sigma"'] & & (cons\ x'\ (cons\ (succ\ y')\ l')) \arrow[dl, "\sigma"] \\
& (cons\ (succ\ x)\ (cons\ (succ\ y')\ l)) &
\end{tikzcd}
$$

Are there other unifiers for this example?

$$\sigma \triangleq \{x' \mapsto (succ\ x), y \mapsto (succ\ y'), l' \mapsto l\}$$

First-order term unification - Definitions

$$\begin{tikzcd} t \arrow[dr, "\sigma"'] & & t' \arrow[dl, "\sigma"] \\ & u & \end{tikzcd}$$

We are interested in finding **the most general** unifier!

Definition

σ is *more general* than η, written as $\sigma \leq \eta$, if there is a substitution θ such that $\sigma\theta = \eta$

First-order term unification - Unification problem

$$\text{Unification problem} = \text{either } \{t_1 \doteq t'_1, \ldots, t_n \doteq t'_n\} \text{ or } \dashv$$

$$\{(cons\ (succ\ x)\ (cons\ y\ l)) \doteq (cons\ x'\ (cons\ (succ\ y')\ l'))\}$$

$$\{(succ\ x) \doteq x', (cons\ y\ l) \doteq (cons\ (succ\ y')\ l')\}$$

First-order term unification - Solved forms

Solved form: either $\dashv$

or: $\{x_1 \doteq u_1, \ldots, x_k \doteq u_k\}$, where $x_i \notin vars(u_j)$ and $x_i \neq x_j$,
$i, j \in \{1, \ldots, k\}$

First-order term unification - Unification algorithm

Delete: $P \cup \{t \doteq t\} \Rightarrow P$
Decomposition: $P \cup \{(f\ t_1\ \ldots\ t_n) \doteq (f\ t'_1\ \ldots\ t'_n)\} \Rightarrow P \cup \{t_1 \doteq t'_1, \ldots, t_n \doteq t'_n\}$
Orient: $P \cup \{(f\ t_1, \ldots\ t_n) \doteq x\} \Rightarrow P \cup \{x \doteq (f\ t_1\ \ldots\ t_n)\}$
Elimination: $P \cup \{x \doteq t\} \Rightarrow P\{x \mapsto t\} \cup \{x \doteq t\}$ if $x \notin vars(t), x \in vars(P)$
Symbol clash: $P \cup \{(f\ t_1\ \ldots\ t_n) \doteq (g\ t'_1\ \ldots\ t'_n)\} \Rightarrow \dashv$
Occurs check: $P \cup \{x \doteq (f\ t\ \ldots\ t)\} \Rightarrow \dashv$, if $x \in vars((f\ t\ \ldots\ t))$

First-order term unification - Example

Delete:	$P \cup \{t \doteq t\} \Rightarrow P$
Decomposition:	$P \cup \{(f\ t_1\ \ldots\ t_n) \doteq (f\ t'_1\ \ldots\ t'_n)\} \Rightarrow P \cup \{t_1 \doteq t'_1, \ldots, t_n \doteq t'_n\}$
Orient:	$P \cup \{(f\ t_1, \ldots\ t_n) \doteq x\} \Rightarrow P \cup \{x \doteq (f\ t_1\ \ldots\ t_n)\}$
Elimination:	$P \cup \{x \doteq t\} \Rightarrow P\{x \mapsto t\} \cup \{x \doteq t\}$ if $x \notin vars(t), x \in vars(P)$
Symbol clash:	$P \cup \{(f\ t_1\ \ldots\ t_n) \doteq (g\ t'_1\ \ldots\ t'_n)\} \Rightarrow \dashv$
Occurs check:	$P \cup \{x \doteq (f\ t\ \ldots\ t)\} \Rightarrow \dashv$, if $x \in vars((f\ t\ \ldots\ t))$

$\{(cons\,(succ\,x)\,(cons\,y\,l)) \doteq (cons\,x'\,(cons\,(succ\,y')\,l'))\}$ $\Rightarrow$ **(Decomposition)**

$\{(succ\,x) \doteq x',\ (cons\,y\,l) \doteq (cons\,(succ\,y')\,l')\}$ $\Rightarrow$ **(Orient)**

$\{x' \doteq (succ\,x),\ (cons\,y\,l) \doteq (cons\,(succ\,y')\,l')\}$ $\Rightarrow$ **(Decomposition)**

$\{x' \doteq (succ\,x),\ y \doteq (succ\,y'),\ l \doteq l'\}$

First-order term unification - Example

$$
\begin{array}{lcl}
\{(cons\,(succ\,x)\,(cons\,y\,l)) \doteq (cons\,x'\,(cons\,(succ\,y')\,l'))\} & \Rightarrow & (\textbf{Decomposition}) \\
\{(succ\,x) \doteq x',\ (cons\,y\,l) \doteq (cons\,(succ\,y')\,l')\} & \Rightarrow & (\textbf{Orient}) \\
\{x' \doteq (succ\,x),\ (cons\,y\,l) \doteq (cons\,(succ\,y')\,l')\} & \Rightarrow & (\textbf{Decomposition}) \\
\{x' \doteq (succ\,x),\ y \doteq (succ\,y'),\ l \doteq l'\} & \triangleq & P'
\end{array}
$$

Solved form!

$$\boxed{\sigma \triangleq \{x' \mapsto (succ\ x), y \mapsto (succ\ y'), l \mapsto l'\}}$$

First-order term unification - Example

$$\{(cons\,(succ\,x)\,(cons\,y\,l)) \doteq (cons\,zero\,(cons\,(succ\,y')\,l'))\} \quad \Rightarrow \quad (\textbf{Decomposition})$$
$$\{\boxed{(succ\,x) \doteq zero},\ (cons\,y\,l) \doteq (cons\,(succ\,y')\,l')\} \quad \Rightarrow \quad (\textbf{SymbolClash})$$
$$\dashv$$

Solved form!

No substitution in this case!

First-order term unification - Example

Theorem (Martelli&Montanari)

Let P be a unification problem. Then :

1. *Progress: If P is not in solved form, then there exists P' such that $P \Rightarrow P'$;*
2. *Solution preservation: If $P \Rightarrow P'$ then $\mathit{unifiers}(P) = \mathit{unifiers}(P')$;*
3. *Termination: There is no infinite sequence $P \Rightarrow P_1 \Rightarrow P_2 \Rightarrow \cdots$;*
4. *Most general unifier: If θ is a solution for P, then for any maximal sequence of transformations that starts with P and ends with P', either P' is $\dashv$ or P' is in solved form and $\sigma_{P'} \leq \theta$. There is no solution for P iff P' is $\dashv$.*

First-order term unification in Matching Logic

Semantic unification in ML = conjunction of term patterns

$$\boxed{(cons\,(succ\,x)\,(cons\,y\,l))} \wedge (cons\,x'\,(cons\,(succ\,y')\,l'))$$

GOAL: simplify such conjunctions

$$\boxed{(cons\,(succ\,x)\,(cons\,y\,l))} \wedge \underbrace{(x' = (succ\,x) \wedge y = (succ\,y') \wedge l = l')}_{\phi^{\sigma}}$$

The substitution is obtained using the unification algorithm!

First-order term unification in Matching Logic

Definition

For each $P = \{t_1 \doteq t'_1, \ldots, t_n \doteq t'_n\}$ we define $\phi^P = \bigwedge_{i=1}^n t_i = t'_i$.

Lemma

For all unification problems P and P', if $P \Rightarrow P'$ then $TERM(S, \Sigma) \models \phi^P \leftrightarrow \phi^{P'}$.

First-order term unification in Matching Logic

Lemma

If $\{t_1 \doteq t_2\} \Rightarrow^! P$ *then* $TERM(S, \Sigma) \models (t_1 \wedge t_2) \leftrightarrow (t_i \wedge \phi^P)$, *where* $i \in \{1, 2\}$.

Soundness

Definition

Two term patterns t_1 and t_2 are unifiable (in ML) iff $TERM(S, \Sigma) \models \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$, where $\overline{x} = vars(t_1 \wedge t_2)$.
Consequently, the term patterns t_1 and t_2 are not unifiable iff $TERM(S, \Sigma) \models \neg \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$.

Theorem (Soundness)

If $\{t_1 \doteq t_2\} \Rightarrow^! P$ *then the following hold:*

1. *If* $P \neq \dashv$ *then* $TERM(S, \Sigma) \models \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$;
2. *If* $P = \dashv$ *then* $TERM(S, \Sigma) \models \neg \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$.

Completeness

Definition

Two term patterns t_1 and t_2 are unifiable (in ML) iff $TERM(S, \Sigma) \models \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$, where $\overline{x} = vars(t_1 \wedge t_2)$.
Consequently, the term patterns t_1 and t_2 are not unifiable iff $TERM(S, \Sigma) \models \neg \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$.

Theorem (Completeness)

Let t_1 and t_2 be two term patterns.

1. *If $TERM(S, \Sigma) \models \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$ then $\{t_1 \doteq t_2\} \Rightarrow^! P \neq \dashv$;*
2. *If $TERM(S, \Sigma) \models \neg \lceil \exists \overline{x}. t_1 \wedge t_2 \rceil$ then $\{t_1 \doteq t_2\} \Rightarrow^! \dashv$.*

Certification

$$TERM(S, \Sigma) \models (t_1 \wedge t_2) \leftrightarrow (t_i \wedge \phi^P)$$

IDEA: derived proof rules that correspond to each step of the unification algorithm

STEPS:

- Execute the unification algorithm on the input unification problem: $\{t_1 \doteq t_2\}$
- Obtain an execution trace, e.g., $T = \textbf{Decomposition}, \textbf{Orientation}$
- Based on the obtained trace generate a proof where each derived proof rule is replaced by its certificate schemata:

...
instance of the certificate schema for **Decomposition**
instance of the certificate schema for **Orientation**
...

Certificate schemata for Decomposition

(k)	$\varphi \to \varphi' \wedge (f\ t_1\ \ldots\ t_n) = (f\ t'_1\ \ldots\ t'_n)$	(premise)
(k + 1)	$(f\ t_1\ \ldots\ t_n) = (f\ t'_1\ \ldots\ t'_n) \to (t_1 = t'_1) \wedge \cdots \wedge (t_n = t'_n)$	NoConfusion II
(k + 2)	$\varphi' \wedge (f\ t_1\ \ldots\ t_n) = (f\ t'_1\ \ldots\ t'_n) \to \varphi' \wedge (t_1 = t'_1) \wedge \cdots \wedge (t_n = t'_n)$	$\to_{context}$: k + 1, φ'
(k + 3)	$\varphi \to \varphi' \wedge (t_1 = t'_1) \wedge \cdots \wedge (t_n = t'_n)$	$\to_{tranz}$: k, k + 2

Certificate example

Orientation

Decomposition

(1)	$(cons\ x\ a) = (cons\ a\ z) \rightarrow (cons\ x\ a) = (cons\ a\ z)$	$\rightarrow_{refl}$
(2)	$(cons\ x\ a) = (cons\ a\ z) \rightarrow (a = z) \wedge (x = a)$	NoConfusion II
(3)	$(cons\ x\ a) = (cons\ a\ z) \rightarrow (a = z) \wedge (x = a)$	$\rightarrow_{context}$: 2
(4)	$(cons\ x\ a) = (cons\ a\ z) \rightarrow (a = z) \wedge (x = a)$	$\rightarrow_{tranz}$: 1, 3
(5)	$(a = z) \rightarrow (z = a)$	$=_{symmetry}$
(6)	$(a = z) \wedge (x = a) \rightarrow (x = a) \wedge (z = a)$	$\rightarrow_{context}$: 5
(7)	$(cons\ x\ a) = (cons\ a\ z) \rightarrow (x = a) \wedge (z = a)$	$\rightarrow_{tranz}$: 4, 6
(8)	$(cons\ x\ a) \wedge (cons\ x\ a) = (cons\ a\ z) \rightarrow (cons\ x\ a) \wedge (x = a) \wedge (z = a)$	$\rightarrow_{context}$: 7
(9)	$(cons\ a\ z) \wedge (cons\ x\ a) \rightarrow (cons\ x\ a) \wedge ((cons\ x\ a) = (cons\ a\ z))$	Proposition 9
(10)	$(cons\ a\ z) \wedge (cons\ x\ a) \rightarrow (cons\ x\ a) \wedge (x = a) \wedge (z = a)$	$\rightarrow_{tranz}$: 9, 8

Common to all proofs

Demo

Anti-unification

First-order Term Anti-Unification

Definition

t is a common *generalisation* of t_1 and t_2 if there are σ_1 and σ_2 s.t. $t\sigma_1 = t_1$ and $t\sigma_2 = t_2$

$$
\begin{tikzcd}
 & t \arrow[dl, "\sigma_1"'] \arrow[dr, "\sigma_2"] & \\
t_1 & & t_2
\end{tikzcd}
$$

First-order Term Anti-Unification - Example

generalisation

$$\begin{tikzcd}
 & (cons\ z_1\ z_2) \arrow[dl, "\sigma'_1"'] \arrow[dr, "\sigma'_2"] & \\
(cons\ (succ\ x_1)\ (cons\ zero\ l_1)) & & (cons\ x_2\ (cons\ (succ\ x_2)\ l_2))
\end{tikzcd}$$

$$\sigma'_1 = \{z_1 \mapsto (succ\ \ x_1), z_2 \mapsto (cons\ \ zero\ \ l_1)\}$$
$$\sigma'_2 = \{z_1 \mapsto x_2, z_2 \mapsto (cons\ \ (succ\ x_2)\ \ l_2)\}$$

First-order Term Anti-Unification - Example

another generalisation

$$
\begin{tikzcd}
& (\mathit{cons}\ z_1\ (\mathit{cons}\ z_3\ z_4)) \arrow[dl, "\sigma_1"'] \arrow[dr, "\sigma_2"] & \\
(\mathit{cons}\ (\mathit{succ}\ x_1)\ (\mathit{cons}\ \mathit{zero}\ l_1)) & & (\mathit{cons}\ x_2\ (\mathit{cons}\ (\mathit{succ}\ x_2)\ l_2))
\end{tikzcd}
$$

$$
\begin{aligned}
\sigma_1 &= \{z_1 \mapsto (\mathit{succ}\ x_1), z_3 \mapsto \mathit{zero}, z_4 \mapsto l_1\} \\
\sigma_2 &= \{z_1 \mapsto x_2, z_3 \mapsto (\mathit{succ}\ x_2), z_4 \mapsto l_2\}
\end{aligned}
$$

LGG = Least General Generalisation

t' is more *general* than a term t if there is σ s.t. $t'\sigma = t$

$$\boxed{\sigma = \{z_2 \mapsto (cons\ z_3\ z4)\}}$$

$$\begin{tikzcd}
\mathbf{t} & & t' \\
\boxed{(cons\ z_1\ \boxed{(cons\ z_3\ z_4)})} \arrow[d, "\sigma_1"'] \arrow[drr, "\sigma_2", near end] & & \boxed{(cons\ z_1\ \boxed{z_2})} \arrow[dll, "\sigma'_1"', near end] \arrow[d, "\sigma'_2"] \\
(cons\ (succ\ x_1)\ (cons\ zero\ l_1)) & & (cons\ x_2\ (cons\ (succ\ x_2)\ l_2))
\end{tikzcd}$$

Plotkin's algorithm for finding the LGG

Definition

Anti-unification problem $=$ a pair $\langle t, P \rangle$, where:

- t is a term, and
- P is a non-empty set of pairs $z \mapsto u \sqcup v$, (z is a variable and u and v are terms)

Plotkin's algorithm for finding the LGG

$$\langle t, P \cup \{z \mapsto (f\ u_1\ \ldots\ u_n) \sqcup (f\ v_1\ \ldots\ v_n)\}\rangle \rightsquigarrow$$
$$\langle t[(f\ z_1\ \ldots\ z_n)/z], P \cup \{z_1 \mapsto u_1 \sqcup v_1, \ldots, z_n \mapsto u_n \sqcup v_n\}\rangle,$$

where $z_1, \ldots, z_n$ are fresh variables

Plotkin's algorithm - Example

$t_1 = (cons\,(succ\,x_1)\,(cons\,zero\,l_1))$ $\qquad$ $t_2 = (cons\,x_2\,(cons\,(succ\,x_2)\,l_2))$

$$\langle z, \{z \mapsto t_1 \sqcup t_2\}\rangle =$$

$$\langle z, \{z \mapsto (cons\,(succ\,x_1)\,(cons\,zero\,l_1)) \sqcup (cons\,x_2\,(cons\,(succ\,x_2)\,l_2))\}\rangle \rightsquigarrow$$

$$\langle z[(cons\,z_1\,z_2)/z], \{z_1 \mapsto (succ\,x_1) \sqcup x_2, z_2 \mapsto (cons\,zero\,l_1) \sqcup (cons\,(succ\,x_2)\,l_2)\}\rangle =$$

$$\langle (cons\,z_1\,z_2), \{z_1 \mapsto (succ\,x_1) \sqcup x_2, z_2 \mapsto (cons\,zero\,l_1) \sqcup (cons\,(succ\,x_2)\,l_2)\}\rangle \rightsquigarrow^{!}$$

$$\langle (cons\,z_1\,z_2)[(cons\,z_3\,z_4)/z_2], \{z_1 \mapsto (succ\,x_1) \sqcup x_2, z_3 \mapsto zero \sqcup (succ\,x_2), z_4 \mapsto l_1 \sqcup l_2\}\rangle =$$

$$\langle (cons\,z_1\,(cons\,z_3\,z_4)), \{z_1 \mapsto (succ\,x_1) \sqcup x_2, z_3 \mapsto zero \sqcup (succ\,x_2), z_4 \mapsto l_1 \sqcup l_2\}\rangle$$

LGG

$$\sigma_1 = \{z_1 \mapsto (succ\,x_1), z_3 \mapsto zero, z_4 \mapsto l_1\}$$

$$\sigma_2 = \{z_1 \mapsto x_2, z_3 \mapsto (succ\,x_2), z_4 \mapsto l_2\}$$

Anti-unification in Matching Logic

Anti-unification in ML = disjunction of term patterns

$$(\mathit{cons}\ (\mathit{succ}\ x_1)\ (\mathit{cons}\ \mathit{zero}\ l_1)) \boxed{\vee} (\mathit{cons}\ x_2\ (\mathit{cons}\ (\mathit{succ}\ x_2)\ l_2))$$

GOAL: simplify such disjunctions

LGG

Substitutions

$$\exists z_1 . \exists z_3 . \exists z_4 . \boxed{(\mathit{cons}\ z_1\ (\mathit{cons}\ z_3\ z_4))} \wedge \Big(\underbrace{(z_1 = (\mathit{succ}\ x_1) \wedge z_3 = \mathit{zero} \wedge z_4 = l_1)}_{\boxed{\phi^{\sigma_1}}} \vee \underbrace{(z_1 = x_2 \wedge z_3 = (\mathit{succ}\ x_2) \wedge z_4 = l_2)}_{\boxed{\phi^{\sigma_2}}} \Big)$$

Anti-unification in Matching Logic

Definition

For each anti-unification problem $\langle t, P\rangle$ we define a corresponding ML pattern

$$\phi^{\langle t,P\rangle} \triangleq \exists \overline{z}.t \wedge \left(\phi^{\sigma_1} \vee \phi^{\sigma_2}\right),$$

where $\sigma_1 = \{z \mapsto u \mid z \mapsto u \sqcup v \in P\}$,
$\sigma_2 = \{z \mapsto v \mid z \mapsto u \sqcup v \in P\}$, and
$vars(t) = dom(\sigma_1) = dom(\sigma_2) = \overline{z}$.

Anti-unification in Matching Logic

Theorem (Soundness)

Let t_1 and t_2 be two term patterns and z a variable such that $z \notin vars(t_1) \cup vars(t_2)$.

If $\langle z, \{z \mapsto t_1 \sqcup t_2\}\rangle \leadsto^! \langle t, P\rangle$, then $TERM(S, F) \models (t_1 \vee t_2) \leftrightarrow \phi^{\langle t,P\rangle}$.

Certificate generation

$$TERM(S, F) \models (t_1 \vee t_2) \leftrightarrow \phi^{\langle t, P \rangle}$$

IDEA:

- Execute Plotkin’s algorithm for finding the LGG and keep a trace of the steps
- Generate a proof for each step based on a proof schemata
- Compose proofs for each step

Certificate generation

These equivalences are the most difficult ones! Each equivalence corresponds to a step in Plotkin's algorithm.

(1)	$t_1 \lor t_2 \leftrightarrow$ $\exists z. z \land (z = (cons\,(succ\,x_1)\,(cons\,zero\,l_1)) \lor z = (cons\,x_2\,(cons\,(succ\,x_2)\,l_2)))$	$\lor_{gen}$
(2.1)	$\exists z. z \land (z = (cons\,(succ\,x_1)\,(cons\,zero\,l_1)) \lor z = (cons\,x_2\,(cons\,(succ\,x_2)\,l_2))) \leftrightarrow$ $\exists z_1. \exists z_2. (cons\ z_1\ z_2) \land$ $\big((z_1 = (succ\,x_1) \land z_2 = (cons\,zero\,l_1)) \lor (z_1 = x_2 \land z_2 = (cons\,(succ\,x_2)\,l_2))\big)$	$\rightsquigarrow_{step}$
(2.2)	$\exists z_1. \exists z_2. (cons\ z_1\ z_2) \land$ $\big((z_1 = (succ\,x_1) \land z_2 = (cons\,zero\,l_1)) \lor (z_1 = x_2 \land z_2 = (cons\,(succ\,x_2)\,l_2))\big) \leftrightarrow$ $\exists z_1. \exists z_3. \exists z_4. (cons\ z_1\ (cons\ z_3\ z_4)) \land$ $\big((z_1 = (succ\,x_1) \land z_3 = zero \land z_4 = l_1) \lor (z_1 = x_2 \land z_3 = (succ\,x_2) \land z_4 = l_2)\big)$	$\rightsquigarrow_{step}$
(3.1)	$t_1 \lor t_2 \leftrightarrow$ $\exists z_1. \exists z_2. (cons\ z_1\ z_2) \land$ $\big((z_1 = (succ\,x_1) \land z_2 = (cons\,zero\,l_1)) \lor (z_1 = x_2 \land z_2 = (cons\,(succ\,x_2)\,l_2))\big)$	$\leftrightarrow_{tranz}: 1, 2.1$
(3.2)	$t_1 \lor t_2 \leftrightarrow$ $\exists z_1. \exists z_3. \exists z_4. (cons\ z_1\ (cons\ z_3\ z_4)) \land$ $\big((z_1 = (succ\,x_1) \land z_3 = zero \land z_4 = l_1) \lor (z_1 = x_2 \land z_3 = (succ\,x_2) \land z_4 = l_2)\big)$	$\leftrightarrow_{tranz}: 3.1, 2.2$

Demo

Conclusion

- Matching Logic can specify the term algebra up to an isomorphism
- Consequently, some computations in the term algebra can be axiomatized in Matching Logic
- In this presentation we considered the unification and anti-unification
- Initial algebra for the equational specification can also be specified in Matching Logic up to an isomorphism
- The next challenge is to see how the computations modulo equational axioms can be captured by Matching Logic