A Formulation of Second-Order Matching

Djamel Ouassim Ait-Moussa

Supervisor: Adrien Guatto

TRE presentation, July 1st 2024

Université Paris Cité

Pattern matching in programming

- Used in functional programming languages.
- Adopted more and more by other languages, e.g. Rust or Python.

Pattern matching in programming

- Used in functional programming languages.
- Adopted more and more by other languages, e.g. Rust or Python.
- Allows users to discriminate and destructure symbolic data.

Pattern matching in programming

- Used in functional programming languages.
- Adopted more and more by other languages, e.g. Rust or Python.
- Allows users to discriminate and destructure symbolic data.

Example

A language **expr** to manipulate arithmetic expressions.

$$u_1 = (5 + 4) \times (7 + 9) \quad u_2 = 5 \times 7 + 4 \times 7 + 5 \times 9 + 4 \times 9$$

Pattern matching in programming

- Used in functional programming languages.
- Adopted more and more by other languages, e.g. Rust or Python.
- Allows users to discriminate and destructure symbolic data.

Example

A language **expr** to manipulate arithmetic expressions.

$$u_1 = (5 + 4) \times (7 + 9) \quad u_2 = 5 \times 7 + 4 \times 7 + 5 \times 9 + 4 \times 9$$

```
type expr = Int of int
          | Add of expr*expr | Mul of expr*expr
          | Div of expr*expr | Sub of expr*expr
```

Pattern matching in programming

- Used in functional programming languages.
- Adopted more and more by other languages, e.g. Rust or Python.
- Allows users to discriminate and destructure symbolic data.

Example

A language **expr** to manipulate arithmetic expressions.

$$u_1 = (5+4)\times(7+9) \quad u_2 = 5\times 7 + 4\times 7 + 5\times 9 + 4\times 9$$

```
type expr = Int of int
          | Add of expr*expr | Mul of expr*expr
          | Div of expr*expr | Sub of expr*expr
let rec dist e = match e with
  | Mul(x, Add(y, z)) | Mul(Add(y, z), x) ->
      let x' = dist x in
      Add(Mul(x', dist y),Mul(dx, dist z))
  | Mul(x, y) -> Mul(dist x, dist y)
  | Div(x, y) -> Div(dist x, dist y)
  | Add(x, y) -> Add(dist x, dist y)
  | Sub(x, y) -> Sub(dist x, dist y)
```

Theoretical foundation of pattern matching

Given two first-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

Example

$t = \texttt{Mul}(\texttt{x}, \texttt{Add}(\texttt{y}, \texttt{z}))$

$\sigma = \{x \mapsto \texttt{Add}(5, 4), y \mapsto 7, z \mapsto 9\}$

$t[\sigma] = \texttt{Mul}(\texttt{Add}(5, 4), \texttt{Add}(7, 9))$

Theoretical foundation of pattern matching

Given two first-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

First-order terms: trees over a signature of operators with fixed arity.

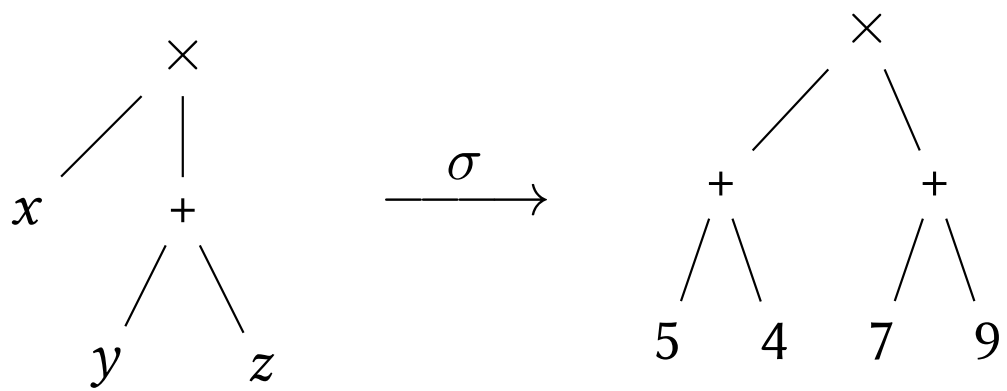
Example

$t = \mathbf{Mul}(\mathrm{x}, \ \mathbf{Add}(\mathrm{y}, \ \mathrm{z}))$

$\sigma = \{x \mapsto \mathbf{Add}(5, \ 4), y \mapsto 7, z \mapsto 9\}$

$t[\sigma] = \mathbf{Mul}(\mathbf{Add}(5, \ 4), \ \mathbf{Add}(7, \ 9))$

Signature: $\Sigma = \{\mathbf{Add} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \mathbf{Mul} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots\}$



Theoretical foundation of pattern matching

Given two first-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

First-order terms: trees over a signature of operators with fixed arity.

Example

$t = \texttt{Mul}(\mathrm{x},\ \texttt{Add}(\mathrm{y},\ \mathrm{z}))$
$\sigma = \{x \mapsto \texttt{Add}(5,\ 4), y \mapsto 7, z \mapsto 9\}$
$t[\sigma] = \texttt{Mul}(\texttt{Add}(5,\ 4),\ \texttt{Add}(7,\ 9))$
Signature: $\Sigma = \{\texttt{Add} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \texttt{Mul} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots\}$

Finding the solution σ is **reverting** the operation $t[-]$.

Theoretical foundation of pattern matching

Given two first-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

First-order terms: trees over a signature of operators with fixed arity.

Example

$t = \texttt{Mul}(\mathrm{x},\ \texttt{Add}(\mathrm{y},\ \mathrm{z}))$
$\sigma = \{x \mapsto \texttt{Add}(5,\ 4), y \mapsto 7, z \mapsto 9\}$
$t[\sigma] = \texttt{Mul}(\texttt{Add}(5,\ 4),\ \texttt{Add}(7,\ 9))$
Signature: $\Sigma = \{\texttt{Add} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \texttt{Mul} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots\}$

Finding the solution σ is **reverting** the operation $t[-]$.

Theorem (Robinson, 1965)

First-order unification is decidable, and when a solution exists, there exists a unique most general solution that is computable.

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 1: use first-order terms

```
type expr = Int of int | Add of expr * expr | ...
          | Var of string
          | Sum of expr * expr * string * expr
```

E.g. u_1 is represented by

```
Sum(1, 5, "i", Sum(1, 16, "j", Mul(Var "i", Var "i"))).
```

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 1: use first-order terms

```
type expr = Int of int | Add of expr * expr | ...
          | Var of string
          | Sum of expr * expr * string * expr
```

E.g. u_1 is represented by

```
Sum(1, 5, "i", Sum(1, 16, "j", Mul(Var "i", Var "i"))).
```

Problems

- Pattern matching cannot express symbolic binding.

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 1: use first-order terms

```
type expr = Int of int | Add of expr * expr | ...
          | Var of string
          | Sum of expr * expr * string * expr
```

E.g. u_1 is represented by

```
Sum(1, 5, "i", Sum(1, 16, "j", Mul(Var "i", Var "i"))).
```

Problems

- Pattern matching cannot express symbolic binding.
- Similar, repetitive algorithms to handle symbolic binding, such as substitution, equality modulo renaming bound variables, etc.

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 2 (sketch): add a binding construct

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 2 (sketch): add a binding construct

```
type expr = Int of int | Add of expr*expr | ...
          | Var of string (* implicitly defined *)
          | Sum of expr * expr * (expr => expr)
```

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 2 (sketch): add a binding construct

```
type expr = Int of int | Add of expr*expr | ...
          | Sum of expr * expr * (expr ⟹ expr)
```

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 2 (sketch): add a binding construct

```
type expr = Int of int | Add of expr*expr | ...
          | Sum of expr * expr * (expr ⟹ expr)
```

E.g. u_1 is represented by `Sum(1,5,λi.Sum(1,16,λj.Mul(i,i)))`

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 2 (sketch): add a binding construct

```
type expr = Int of int | Add of expr*expr | ...
          | Sum of expr * expr * (expr ⟹ expr)
```

E.g. u_1 is represented by `Sum(1,5,λi.Sum(1,16,λj.Mul(i,i)))`

```
match u1 with (* try to optimize nested sums *)
  | Sum(x, y, λi.Sum(z, t, λj.f(i)))
    -> Mul(Add(1,Sub(t, z)), Sum(x, y)λi.z(i))
```

Binding

What if we want to add a sum operator to our **expr** language?

$$u_1 = \sum_{i=1}^{5} \sum_{j=1}^{16} i \times i \qquad u_2 = (1 + (16 - 1)) \times \sum_{i=1}^{16} i \times i$$

Idea 2 (sketch): add a binding construct

```
type expr = Int of int | Add of expr*expr | ...
          | Sum of expr * expr * (expr ⟹ expr)
```

E.g. u_1 is represented by `Sum(1,5,λi.Sum(1,16,λj.Mul(i,i)))`

```
match u1 with (* try to optimize nested sums *)
  | Sum(x, y, λi.Sum(z, t, λj.f(i)))
    -> Mul(Add(1,Sub(t, z)), Sum(x, y)λi.z(i))
```

- u_1 matches the pattern, becomes u_2

Theoretical foundation of second-order matching

Given two terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

terms: trees over a signature of operators with fixed arity, .

Theoretical foundation of second-order matching

Given two second-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

Second-order terms: trees over a signature of operators with fixed arity, and a binding construct in the form "$\lambda x.t$".

Theoretical foundation of second-order matching

Given two second-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

Second-order terms: trees over a signature of operators with fixed arity, and a binding construct in the form "$\lambda x.t$".

Example

$t = \mathbf{Sum}(\mathtt{x},\ \mathtt{y},\ \lambda \mathtt{i}.\mathbf{Sum}(\mathtt{z},\ \mathtt{t},\ \lambda \mathtt{j}.\mathtt{f}(\mathtt{i})))$

$\sigma = \{x \mapsto 1, y \mapsto 5, z \mapsto 1, t \mapsto 16, f \mapsto \lambda i.\mathbf{Mul}(\mathtt{i},\ \mathtt{i})\}$

$t[\sigma] = \mathbf{Sum}(1,\ 5,\ \lambda \mathtt{i}.\mathbf{Sum}(1,\ 16,\ \lambda \mathtt{j}.\mathbf{Mul}(\mathtt{i},\ \mathtt{i})))$

Signature: $\Sigma^* \leftarrow \Sigma \cup \{\mathbf{Sum} : \mathbb{T} \to \mathbb{T} \to (\mathbb{T} \to \mathbb{T}) \to \mathbb{T}\}$

Theoretical foundation of second-order matching

Given two second-order terms t and u:

- **Unification problem:** find σ such that $t[\sigma] = u[\sigma]$.
- **Matching problem:** find σ such that $t[\sigma] = u$ (u closed).

Second-order terms: trees over a signature of operators with fixed arity, and a binding construct in the form "$\lambda x.t$".

Example

$t = \mathbf{Sum}(\mathtt{x},\ \mathtt{y},\ \lambda \mathtt{i}.\mathbf{Sum}(\mathtt{z},\ \mathtt{t},\ \lambda \mathtt{j}.\mathtt{f}(\mathtt{i})))$

$\sigma = \{x \mapsto 1, y \mapsto 5, z \mapsto 1, t \mapsto 16, f \mapsto \lambda i.\mathbf{Mul}(\mathtt{i},\ \mathtt{i})\}$

$t[\sigma] = \mathbf{Sum}(1,\ 5,\ \lambda \mathtt{i}.\mathbf{Sum}(1,\ 16,\ \lambda \mathtt{j}.\mathbf{Mul}(\mathtt{i},\ \mathtt{i})))$

Signature: $\Sigma^* \leftarrow \Sigma \cup \{\mathbf{Sum} : \mathbb{T} \to \mathbb{T} \to (\mathbb{T} \to \mathbb{T}) \to \mathbb{T}\}$

Theorem (Huet, 1976)

Second-order matching is decidable.

Goals in this TRE

- Reformulate Huet and Lang (1978)'s second-order matching algorithm in a declarative way.
- Give concise and complete proofs of correctness and termination.
- Implement the algorithm in OCaml.

Goals in this TRE

- Reformulate Huet and Lang (1978)'s second-order matching algorithm in a declarative way.
- Give concise and complete proofs of correctness and termination.
- Implement the algorithm in OCaml.

Non-goals:

- Design a programming language implementing second-order matching.

Second-order abstract syntax (1/3)

Types: $\tau, \upsilon ::= \mathbb{T} \mid \tau \to \upsilon$.

First-order types: $\mathbb{T}, \mathbb{T} \to \mathbb{T}, \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots$

Second-order types: $\tau_1 \to \ldots \to \tau_n \to \mathbb{T}$ where $(\tau_i)_{1 \leq i \leq n}$ are first-order types.

Second-order abstract syntax (1/3)

Types: $\tau, \upsilon ::= \mathbb{T} \mid \tau \to \upsilon$.

First-order types: $\mathbb{T}, \mathbb{T} \to \mathbb{T}, \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots$

Second-order types: $\tau_1 \to \ldots \to \tau_n \to \mathbb{T}$ where $(\tau_i)_{1 \leq i \leq n}$ are first-order types.

Signatures: A signature Σ is a set of second-order typed *operators*.

Second-order abstract syntax (1/3)

Types: $\tau, \upsilon ::= \mathbb{T} \mid \tau \to \upsilon$.

First-order types: $\mathbb{T}, \mathbb{T} \to \mathbb{T}, \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots$

Second-order types: $\tau_1 \to \ldots \to \tau_n \to \mathbb{T}$ where $(\tau_i)_{1 \leq i \leq n}$ are first-order types.

Signatures: A signature Σ is a set of second-order typed *operators*.

Example

$$\begin{aligned}\Sigma = \{&\texttt{Add} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \texttt{Mul} : \mathbb{T} \to \mathbb{T} \to \mathbb{T},\\ &\texttt{Sub} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \texttt{Div} : \mathbb{T} \to \mathbb{T} \to \mathbb{T},\\ &\texttt{Sum} : \mathbb{T} \to \mathbb{T} \to (\mathbb{T} \to \mathbb{T}) \to \mathbb{T}\}\end{aligned}$$

Second-order abstract syntax (1/3)

Types: $\tau, \upsilon ::= \mathbb{T} \mid \tau \to \upsilon$.

First-order types: $\mathbb{T}, \mathbb{T} \to \mathbb{T}, \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots$

Second-order types: $\tau_1 \to \ldots \to \tau_n \to \mathbb{T}$ where $(\tau_i)_{1 \leq i \leq n}$ are first-order types.

Signatures: A signature Σ is a set of second-order typed *operators*.

Example

$$\Sigma = \{\texttt{Add} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \texttt{Mul} : \mathbb{T} \to \mathbb{T} \to \mathbb{T},$$
$$\texttt{Sub} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \texttt{Div} : \mathbb{T} \to \mathbb{T} \to \mathbb{T},$$
$$\texttt{Sum} : \mathbb{T} \to \mathbb{T} \to (\mathbb{T} \to \mathbb{T}) \to \mathbb{T}\}$$

Add has a first-order type, while **Sum** has a second-order type

Second-order abstract syntax (1/3)

Types: $\tau, \upsilon ::= \mathbb{T} \mid \tau \to \upsilon$.

First-order types: $\mathbb{T}, \mathbb{T} \to \mathbb{T}, \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \ldots$

Second-order types: $\tau_1 \to \ldots \to \tau_n \to \mathbb{T}$ where $(\tau_i)_{1 \leq i \leq n}$ are first-order types.

Signatures: A signature Σ is a set of second-order typed *operators*.

Example

$$\Sigma = \{\mathbf{Add} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \mathbf{Mul} : \mathbb{T} \to \mathbb{T} \to \mathbb{T},$$
$$\mathbf{Sub} : \mathbb{T} \to \mathbb{T} \to \mathbb{T}, \mathbf{Div} : \mathbb{T} \to \mathbb{T} \to \mathbb{T},$$
$$\mathbf{Sum} : \mathbb{T} \to \mathbb{T} \to (\mathbb{T} \to \mathbb{T}) \to \mathbb{T}\}$$

Add has a first-order type, while **Sum** has a second-order type

Contexts: A context Γ is a set of first-order typed *variables*

Example

$\Gamma = \{x : \mathbb{T},\ f : \mathbb{T} \to \mathbb{T}\}$

Second-order abstract syntax (2/3)

Second-order abstract syntax (2/3)

Pre-terms: $t ::= x \mid F \mid a(t_1, \ldots, t_n) \mid \lambda x.t$ (a variable or operator)

Second-order abstract syntax (2/3)

Pre-terms: $t ::= x \mid F \mid a(t_1, \ldots, t_n) \mid \lambda x.t$ (a variable or operator)

Terms: $\boxed{\Gamma \vdash t : \tau}$ (β-short η-long λ-terms)

$$\frac{\Gamma, x : \mathbb{T} \vdash t : \upsilon}{\Gamma \vdash \lambda x.t : \mathbb{T} \to \upsilon}$$

$$\frac{\Sigma \cup \Gamma \ni a : \tau_1 \to \ldots \to \tau_n \to \mathbb{T} \qquad (\Gamma \vdash u_i : \tau_i)_{1 \leq i \leq n}}{\Gamma \vdash a(u_1, \ldots, u_n) : \mathbb{T}}$$

Second-order abstract syntax (2/3)

Pre-terms: $t ::= x \mid F \mid a(t_1, \ldots, t_n) \mid \lambda x.t$ (a variable or operator)

Terms: $\boxed{\Gamma \vdash t : \tau}$ (β-short η-long λ-terms)

$$\frac{\Gamma, x : \mathbb{T} \vdash t : \upsilon}{\Gamma \vdash \lambda x.t : \mathbb{T} \to \upsilon}$$

$$\frac{\Sigma \cup \Gamma \ni a : \tau_1 \to \ldots \to \tau_n \to \mathbb{T} \qquad (\Gamma \vdash u_i : \tau_i)_{1 \leq i \leq n}}{\Gamma \vdash a(u_1, \ldots, u_n) : \mathbb{T}}$$

Example

$\Gamma = \{x : \mathbb{T}, y : \mathbb{T}, f : \mathbb{T} \to \mathbb{T}\}$
$\Sigma \ni \mathrm{Sum} : \mathbb{T} \to \mathbb{T} \to (\mathbb{T} \to \mathbb{T}) \to \mathbb{T}$

$$\frac{\dfrac{\Gamma \ni x : \mathbb{T}}{\Gamma \vdash x : \mathbb{T}} \qquad \dfrac{\Gamma \ni y : \mathbb{T}}{\Gamma \vdash y : \mathbb{T}} \qquad \dfrac{\dfrac{\Gamma, i : \mathbb{T} \ni f : \mathbb{T} \to \mathbb{T} \qquad \Gamma, i : \mathbb{T} \ni i : \mathbb{T}}{\Gamma, i : \mathbb{T} \vdash f(i) : \mathbb{T}}}{\Gamma \vdash \lambda i.f(i) : \mathbb{T} \to \mathbb{T}}}{\Gamma \vdash \mathrm{Sum}(x, y, \lambda i.f(i)) : \mathbb{T}}$$

Second-order abstract syntax (3/3)

Substitution: a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, denoted σ or ρ.
Applying a substitution to a term hereditarily: replacing each each free occurrence of an x_i in t with t_i, and apply the arguments it had.

Second-order abstract syntax (3/3)

Substitution: a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, denoted σ or ρ.
Applying a substitution to a term hereditarily: replacing each each free occurrence of an x_i in t with t_i, and apply the arguments it had.

Example

$$\lambda y.F(x, y)[x \mapsto G(A), y \mapsto G(B)] = \lambda y.F(G(A), y)$$

Second-order abstract syntax (3/3)

Substitution: a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, denoted σ or ρ.
Applying a substitution to a term hereditarily: replacing each each free occurrence of an x_i in t with t_i, and apply the arguments it had.

Example

$$\lambda y.F(x, y)[x \mapsto G(A), y \mapsto G(B)] = \lambda y.F(G(A), y)$$

$$f(F(y), A)[f \mapsto \lambda a\, b.G(H(b), a), y \mapsto B] = G(H(A), F(B))$$

Second-order abstract syntax (3/3)

Substitution: a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, denoted σ or ρ.
Applying a substitution to a term hereditarily: replacing each each free occurrence of an x_i in t with t_i, and apply the arguments it had.

Example

$$\lambda y.F(x, y)[x \mapsto G(A), y \mapsto G(B)] = \lambda y.F(G(A), y)$$

$$f(F(y), A)[f \mapsto \lambda a\, b.G(H(b), a), y \mapsto B] = G(H(A), F(B))$$

Definition

A closed substitution σ_1 is *more general* than σ_2 if $\sigma_1 \subseteq \sigma_2$.

Matching problem

Definition (Matching problem)

A *matching problem* is a triple (t, u, τ), denoted $t \leq^{\tau}_{?} u$, such that $\Gamma \vdash t : \tau$ and $\emptyset \vdash u : \tau$.

Definition (Solution)

We call *solution* to the problem $t \leq^{\tau}_{?} u$ any closed substitution σ such that $t[\sigma] = u$. We denote the set of solutions for by $\mathcal{F}_{t \leq^{\tau}_{?} u}$.

Matching problem

Definition (Matching problem)

A *matching problem* is a triple (t, u, τ), denoted $t \leq^{\tau}_{?} u$, such that $\Gamma \vdash t : \tau$ and $\emptyset \vdash u : \tau$.

Definition (Solution)

We call *solution* to the problem $t \leq^{\tau}_{?} u$ any closed substitution σ such that $t[\sigma] = u$. We denote the set of solutions for by $\mathcal{F}_{t \leq^{\tau}_{?} u}$.

Example

Given the problem $f(A, g(B)) \leq^{\mathbb{T}}_{?} A$, possible solutions are

$$\sigma_1 = \{f \mapsto \lambda x\, y.A\}$$

$$\sigma_2 = \{f \mapsto \lambda x\, y.A, g \mapsto \lambda x.x\}$$

$$\sigma_3 = \{f \mapsto \lambda x\, y.x\}$$

σ_1 is more general than σ_2, while σ_1 and σ_3 cannot be compared.

Solutions to a matching problem

Remark

Given the problem $f(A, g(B)) \leq^{\mathbb{T}}_{?} A$, for any closed term t, $\sigma_t = \{f \mapsto \lambda x\, y.x, g \mapsto t\}$ is a solution. But this is artificial: these solutions are induced by the general $\sigma = \{f \mapsto \lambda x\, y.x\}$.

Solutions to a matching problem

Remark

Given the problem $f(A, g(B)) \leq^{\mathbb{T}}_{?} A$, for any closed term t, $\sigma_t = \{f \mapsto \lambda x\, y.x, g \mapsto t\}$ is a solution. But this is artificial: these solutions are induced by the general $\sigma = \{f \mapsto \lambda x\, y.x\}$.

Lemma (Canonical solution)

Let $\sigma \in \mathcal{F}_{t \leq^{\tau}_{?} u}$. There exists a unique $\mathrm{c}(\sigma) \in \mathcal{F}_{t \leq^{\tau}_{?} u}$ such that $\mathrm{c}(\sigma) \subseteq \sigma$ and $\mathrm{c}(\sigma)$ is minimal for $\subseteq$ in $\mathcal{F}_{t \leq^{\tau}_{?} u}$. If $\mathrm{c}(\sigma) = \sigma$ we say that σ is canonical.

Solutions to a matching problem

Remark

Given the problem $f(A, g(B)) \leq^{\mathbb{T}}_{?} A$, for any closed term t, $\sigma_t = \{f \mapsto \lambda x\, y.x, g \mapsto t\}$ is a solution. But this is artificial: these solutions are induced by the general $\sigma = \{f \mapsto \lambda x\, y.x\}$.

Lemma (Canonical solution)

Let $\sigma \in \mathcal{F}_{t \leq^{\tau}_{?} u}$. There exists a unique $\mathrm{c}(\sigma) \in \mathcal{F}_{t \leq^{\tau}_{?} u}$ such that $\mathrm{c}(\sigma) \subseteq \sigma$ and $\mathrm{c}(\sigma)$ is minimal for $\subseteq$ in $\mathcal{F}_{t \leq^{\tau}_{?} u}$. If $\mathrm{c}(\sigma) = \sigma$ we say that σ is canonical.

Definition (Canonical solutions set)

We define $\mathcal{S}_{t \leq^{\tau}_{?} u} ::= \{\mathrm{c}(\sigma) \mid \sigma \in \mathcal{F}_{t \leq^{\tau}_{?} u}\}$

Matching inference system (1/2)

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

1) $t = \lambda x.t'$: add x to the context Δ of bound variables.

$$\textsc{Introduce}\ \frac{\Gamma \mid \Delta, x : \mathbb{T} \vdash t \leq^{\tau}_{?} u \dashv \sigma}{\Gamma \mid \Delta \vdash \lambda x.t \leq^{\mathbb{T}\to\tau}_{?} \lambda x.u \dashv \sigma}$$

Matching inference system (1/2)

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

1) $t = \lambda x.t'$: add x to the context Δ of bound variables.

$$\textsc{Introduce}\ \frac{\Gamma \mid \Delta, x : \mathbb{T} \vdash t \leq^{\tau}_{?} u \dashv \sigma}{\Gamma \mid \Delta \vdash \lambda x.t \leq^{\mathbb{T}\to\tau}_{?} \lambda x.u \dashv \sigma}$$

2) $t = a(t_1, \ldots, t_n)$ where a is an operator or a bound variable:

- Check that u is of the form $a(u_1, \ldots, u_n)$.
- Find solution σ_1 to $t_1 \leq^{\tau_1}_{?} u_1$.
- Find solution σ_2 to $t_2[\sigma_1] \leq^{\tau_2}_{?} u_2$.
- ...

$$\textsc{Simplify}\ \frac{\begin{array}{c}\Sigma \cup \Delta \ni a : \tau_1 \times \ldots \tau_n \to \mathbb{T} \\ (\Gamma \mid \Delta \vdash t_i[\sigma_{i-1} \circ \ldots \circ \sigma_1]_{\mathsf{h}} \leq^{\tau_i}_{?} u_i \dashv \sigma_i)_{1 \leq i \leq n}\end{array}}{\Gamma \mid \Delta \vdash a(t_1, \ldots, t_n) \leq^{\mathbb{T}}_{?} a(u_1, \ldots, u_n) \dashv \sigma_n \circ \ldots \circ \sigma_1}$$

Matching inference system (2/2)

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

3) $t = f(t_1, \ldots, t_n)$:

- Choose a value for the variable f.
- Substitute it, then solve the remaining problem.

Matching inference system (2/2)

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

3) $t = f(t_1, \ldots, t_n)$:

- Choose a value for the variable f.
- Substitute it, then solve the remaining problem.

Intuition

The choice for f must be closed and of the form $\lambda x_1 \ldots x_n.b(v_1, \ldots, v_m)$.

Matching inference system (2/2)

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

3) $t = f(t_1, \ldots, t_n)$:

- Choose a value for the variable f.
- Substitute it, then solve the remaining problem.

$$\textsc{Project}\ \frac{\theta := \{f \mapsto \lambda x_1 \ldots x_n.x_i\} \qquad 1 \leq i \leq n \qquad \Gamma \mid \Delta \vdash f(t_1, \ldots, t_n)[\theta]_{\mathsf{h}} \leq^{\tau}_{?} u \dashv \sigma}{\Gamma, f : \_ \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^{\mathbb{T}}_{?} u \dashv \sigma \circ \theta}$$

Intuition

The choice for f must be closed and of the form $\lambda x_1 \ldots x_n.b(v_1, \ldots, v_m)$.

- Either b is one of $x_1, \ldots, x_n$: we Project

Matching inference system (2/2)

$$\boxed{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u \dashv \sigma}$$

3) $t = f(t_1, \ldots, t_n)$:

- Choose a value for the variable f.
- Substitute it, then solve the remaining problem.

$$\textsc{Imitate}\ \frac{\theta := \{f \mapsto \lambda x_1 \ldots x_n.F(\overrightarrow{\lambda y_1, \ldots, y_{k_i}.f_i(x_1, \ldots, x_n, y_1, \ldots, y_{k_i})})\} \quad \Gamma, f_1 : \_, \ldots, f_m : \_ \mid \Delta \vdash f(t_1, \ldots, t_n)[\theta]_{\mathsf{h}} \leq^{A}_{?} F(u_1, \ldots, u_m) \dashv \sigma}{\Gamma, f : \_ \mid \Delta \vdash f(t_1, \ldots, t_n) \leq^{A}_{?} F(u_1, \ldots, u_m) \dashv (\sigma \circ \theta) - f_1, \ldots, f_m}$$

Intuition

The choice for f must be closed and of the form $\lambda x_1 \ldots x_n.b(v_1, \ldots, v_m)$.

- Either b is one of $x_1, \ldots, x_n$: we Project
- Or b is an operator $F \in \Sigma$: we Imitate, and introduce m variables for the v_i's

Soundness and completeness of the system

Theorem (Soundness)

If we have $\Gamma \mid \emptyset \vdash t \leq^{\tau}_{?} u \dashv \sigma$, *then* σ *is a canonical solution to the problem* $t \leq^{\tau}_{?} u$.

Soundness and completeness of the system

Theorem (Soundness)

If we have $\Gamma \mid \emptyset \vdash t \leq^{\tau}_{?} u \dashv \sigma$, *then* σ *is a canonical solution to the problem* $t \leq^{\tau}_{?} u$.

Theorem (Completeness)

For all $\sigma \in \mathcal{S}_{t \leq^{\tau}_{?} u}$, *we have* $\Gamma \mid \emptyset \vdash t \leq^{\tau}_{?} u \dashv \sigma$.

Soundness and completeness of the system

Theorem (Soundness)

If we have $\Gamma \mid \emptyset \vdash t \leq^{\tau}_{?} u \dashv \sigma$, *then* σ *is a canonical solution to the problem* $t \leq^{\tau}_{?} u$.

Theorem (Completeness)

For all $\sigma \in \mathcal{S}_{t \leq^{\tau}_{?} u}$, *we have* $\Gamma \mid \emptyset \vdash t \leq^{\tau}_{?} u \dashv \sigma$.

Intuition for the proof of completeness

- We reason by well-founded induction on the pair (u, t) equipped with the lexicographic order $\prec \times_{\mathcal{L}} \prec$ where $\prec$ is the structural order on terms; and we proceed by cases on t.
- For each case and each σ, we choose a rule (e.g. SIMPLIFY) and express σ as a some $e(\sigma_1, \ldots, \sigma_n)$ where the σ_i are solutions to a smaller problem appearing as a premise in the rule.

The inference system is an algorithm

Judgement input and judgement output

$$\underbrace{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u}_{input} \dashv \underbrace{\sigma}_{output}$$

The inference system is an algorithm

Judgement input and judgement output

$$\underbrace{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u}_{\textit{input}} \dashv \underbrace{\sigma}_{\textit{output}}$$

For all the rules (Introduce, Simplify, Project, Imitate):

- Input of the premises $\Leftarrow^{\text{lin}}$ input of the result.
- Output of the result $\Leftarrow^{\text{lin}}$ output of the premises.

The inference system is an algorithm

Judgement input and judgement output

$$\underbrace{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u}_{\textit{input}} \dashv \underbrace{\sigma}_{\textit{output}}$$

For all the rules (Introduce, Simplify, Project, Imitate):

- Input of the premises $\Leftarrow^{\text{lin}}$ input of the result.
- Output of the result $\Leftarrow^{\text{lin}}$ output of the premises.

$\Longrightarrow$ The system specifies a recursive nondeterministic algorithm.

The inference system is an algorithm

Judgement input and judgement output

$$\underbrace{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u}_{\textit{input}} \dashv \underbrace{\sigma}_{\textit{output}}$$

For all the rules (Introduce, Simplify, Project, Imitate):

- Input of the premises $\Leftarrow^{\text{lin}}$ input of the result.
- Output of the result $\Leftarrow^{\text{lin}}$ output of the premises.

$\Longrightarrow$ The system specifies a recursive nondeterministic algorithm.

Theorem (Termination)

The algorithm specified by the system terminates.

The inference system is an algorithm

Judgement input and judgement output

$$\underbrace{\Gamma \mid \Delta \vdash t \leq^{\tau}_{?} u}_{\textit{input}} \dashv \underbrace{\sigma}_{\textit{output}}$$

For all the rules (Introduce, Simplify, Project, Imitate):

- Input of the premises $\Leftarrow^{\text{lin}}$ input of the result.
- Output of the result $\Leftarrow^{\text{lin}}$ output of the premises.

$\Longrightarrow$ The system specifies a recursive nondeterministic algorithm.

Theorem (Termination)

The algorithm specified by the system terminates.

Corollary

The set $\mathcal{S}_{t \leq^{\tau}_{?} u}$ is computable.

Conclusion

- We have exposed a simple structure of the set of solutions.
- We presented a second-order matching algorithm as an inference system made of mostly atomic rules.

Future work

- Improve the inference system: reformulate the IMITATE rule using smaller, atomic rules.
- Study a functional language implementing the construct.

Bibliography

- Huet, Gérard (Jan. 1976). “Résolution d’Équations dans des Langages d’ordre 1,2,...,ω.”. PhD thesis. URL: https://gallium.inria.fr/~huet/PUBLIC/Huet1976.pdf.
- Huet, Gérard and Bernard Lang (1978). “Proving and applying program transformations expressed with second-order patterns”. In: *Acta Informatica* 11.1. ISSN: 1432-0525. DOI: 10.1007/bf00264598. URL: http://dx.doi.org/10.1007/bf00264598.
- Robinson, J. A. (Jan. 1965). “A Machine-Oriented Logic Based on the Resolution Principle”. In: *J. ACM* 12.1, pp. 23–41. ISSN: 0004-5411. DOI: 10.1145/321250.321253. URL: https://doi.org/10.1145/321250.321253.

Differences with report

- filter for a problem $\mapsto$ solution to a problem
- solution to a problem $\mapsto$ complete+minimal set of solutions
- Monosorted : only one primitive type $\mathbb{T}$

Misc

Complexity: NP-complete in general. (Lewis D. Baxter, The Complexity of Unification, Ph.D. Thesis, University of Waterloo, 1976)