UNIWERSYTET
WARSZAWSKI

Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

# Artificial Intelligence in Theorem Proving

(Sztuczna inteligencja w dowodzeniu)

**Cezary Kaliszyk**

MIMUW, March 2020

## Overview

### Last Lectures

- Premise Selection
- Adaptations of standard machine learning algorithms
- Deep learning for premise selection

### Today

- Unification in Predicate Logic (and other logics)
- Unification in Functional Programming ($\lambda$-calculus)

## Unification: Motivation

In the next part of the lecture, we will need the details of the unification procedure.

Intuitively unification is a way two similar terms can be made same using a substitution.

It has many uses in logic and in functional programming

Will be need it both as a basis for the functioning of ATPs (and we will want to guide ATPs using machine learning in the further part of the course) and it will also be used to define better machine learning features

The remainder of this lecture will formally define unification, first in logic, second in functional programming

# Substitution Definitions (1/2)

## Definition

- substitution is set
$$\theta = \{t_1/x_1, \ldots, t_n/x_n\}$$
with terms $t_1, \ldots, t_n$ and pairwise different variables $x_1, \ldots, x_n$
- given substitution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ and expression $E$, instance $E\theta$ of $E$ is obtained by simultaneously replacing each occurrence of $x_i$ in $E$ by $t_i$
- composition of substitutions $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ and $\sigma = \{s_1/y_1, \ldots, s_k/y_k\}$ is substitution
$$\theta\sigma = \{t_1\sigma/x_1, \ldots, t_n\sigma/x_n\} \cup \{s_i/y_i \mid \forall j\ y_i \neq x_j\}$$

## Example

$$\begin{aligned} \theta &= \{g(y,z)/x, a/y\} & E &= P(f(y), x, y) \\ \sigma &= \{f(y)/x, f(x)/z\} & E\theta &= P(f(a), g(y,z), a) \\ \theta\sigma &= \{g(y, f(x))/x, a/y, f(x)/z\} & \sigma\theta &= \{f(a)/x, f(g(y,z))/z, a/y\} \end{aligned}$$

## Substitution Definitions (2/2)

### Definition

- substitution $\theta$ is at least as general as substitution $\sigma$ if $\exists \mu \; \theta\mu = \sigma$
- unifier of set $S$ of terms is substitution $\theta$ such that $\forall s, t \in S \; s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

### Example

terms $f(x, g(y), x)$ and $f(z, g(u), h(u))$ are unifiable:

| | | |
|---|---|---|
| | $\{h(a)/x, a/y, h(a)/z, a/u\}$ | $\{a/u\}$ |
| unifiers | $\{h(u)/x, u/y, h(u)/z\}$ | mgu |
| | $\{h(g(u))/x, g(u)/y, h(g(u))/z, g(u)/u\}$ | $\{g(u)/u\}$ |

# Unification Algorithm

## Theorem

*unifiable terms have mgu which can be computed by unification algorithm*

## Unification Algorithm

Perform the following steps (in any order)

d decomposition

$$\frac{E_1,\ f(s_1,\ldots,s_n) \approx f(t_1,\ldots,t_n),\ E_2}{E_1,\ s_1 \approx t_1,\ \ldots,\ s_n \approx t_n,\ E_2}$$

t removal of trivial equations

$$\frac{E_1,\ t \approx t,\ E_2}{E_1,\ E_2}$$

v variable elimination

$$\frac{E_1,\ x \approx t,\ E_2}{(E_1,\ E_2)\sigma} \quad \text{and} \quad \frac{E_1,\ t \approx x,\ E_2}{(E_1,\ E_2)\sigma}$$

if $x$ does not occur in $t$ (occurs check) and $\sigma = \{t/x\}$

## Unification Example

Example

$$f(x, g(y), x) \approx f(z, g(u), h(u))$$

## Unification Example

Example

$f(x, g(y), x) \approx f(z, g(u), h(u))$

## Unification Example

### Example

$$f(x, g(y), x) \approx f(z, g(u), h(u))$$

$$\text{d} \Downarrow$$

$$x \approx z,\ g(y) \approx g(u),\ x \approx h(u)$$

## Unification Example

### Example

$$f(x, g(y), x) \approx f(z, g(u), h(u))$$

$$\text{d} \Downarrow$$

$$x \approx z,\ g(y) \approx g(u),\ x \approx h(u)$$

$$\text{v} \Downarrow \qquad \{z/x\}$$

$$g(y) \approx g(u),\ z \approx h(u)$$

## Unification Example

### Example

$$
\begin{array}{c}
f(x, g(y), x) \approx f(z, g(u), h(u)) \\
\text{d} \Downarrow \\
x \approx z,\ g(y) \approx g(u),\ x \approx h(u) \\
\text{v} \Downarrow \qquad \{z/x\} \\
g(y) \approx g(u),\ z \approx h(u) \\
\text{d} \Downarrow \\
y \approx u,\ z \approx h(u)
\end{array}
$$

## Unification Example

Example

$$f(x, g(y), x) \approx f(z, g(u), h(u))$$

$$\text{d} \Downarrow$$

$$x \approx z,\ g(y) \approx g(u),\ x \approx h(u)$$

$$\text{v} \Downarrow \qquad \{z/x\}$$

$$g(y) \approx g(u),\ z \approx h(u)$$

$$\text{d} \Downarrow$$

$$y \approx u,\ z \approx h(u)$$

$$\text{v} \Downarrow \qquad \{u/y\}$$

$$z \approx h(u)$$

## Unification Example

Example

$$f(x, g(y), x) \approx f(z, g(u), h(u))$$

$$\text{d} \Downarrow$$

$$x \approx z,\ g(y) \approx g(u),\ x \approx h(u)$$

$$\text{v} \Downarrow \qquad \{z/x\}$$

$$g(y) \approx g(u),\ z \approx h(u)$$

$$\text{d} \Downarrow$$

$$y \approx u,\ z \approx h(u)$$

$$\text{v} \Downarrow \qquad \{u/y\}$$

$$z \approx h(u)$$

$$\text{v} \Downarrow \qquad \{h(u)/z\}$$

## Unification Example

Example

$$\begin{tikzcd}
f(x, g(y), x) \approx f(z, g(u), h(u)) \arrow[d, Rightarrow, "\mathsf{d}"'] \\
x \approx z,\ g(y) \approx g(u),\ x \approx h(u) \arrow[d, Rightarrow, "\mathsf{v}"', "\{z/x\}"] \\
g(y) \approx g(u),\ z \approx h(u) \arrow[d, Rightarrow, "\mathsf{d}"'] \\
y \approx u,\ z \approx h(u) \arrow[d, Rightarrow, "\mathsf{v}"', "\{u/y\}"] \\
z \approx h(u) \arrow[d, Rightarrow, "\mathsf{v}"', "\{h(u)/z\}"] \\
\square
\end{tikzcd}$$

## Unification Example

### Example

$$f(x, g(y), x) \approx f(z, g(u), h(u))$$

$$\text{d} \Downarrow$$

$$x \approx z,\ g(y) \approx g(u),\ x \approx h(u)$$

$$\text{v} \Downarrow \qquad \{z/x\}$$

$$g(y) \approx g(u),\ z \approx h(u)$$

$$\text{d} \Downarrow$$

$$y \approx u,\ z \approx h(u)$$

$$\text{v} \Downarrow \qquad \{u/y\}$$

$$z \approx h(u)$$

$$\text{v} \Downarrow \qquad \{h(u)/z\}$$

$$\square$$

mgu $\{z/x\}\{u/y\}\{h(u)/z\}$

## Unification Example

### Example

$$\begin{tikzcd}
f(x,g(y),x) \approx f(z,g(u),h(u)) \arrow[d, Rightarrow, "\mathsf{d}"'] \\
x \approx z,\ g(y) \approx g(u),\ x \approx h(u) \arrow[d, Rightarrow, "\mathsf{v}"', "\{z/x\}"] \\
g(y) \approx g(u),\ z \approx h(u) \arrow[d, Rightarrow, "\mathsf{d}"'] \\
y \approx u,\ z \approx h(u) \arrow[d, Rightarrow, "\mathsf{v}"', "\{u/y\}"] \\
z \approx h(u) \arrow[d, Rightarrow, "\mathsf{v}"', "\{h(u)/z\}"] \\
\square
\end{tikzcd}$$

mgu $\{h(u)/x, u/y, h(u)/z\}$

## Unification: Properties

### Theorem

- *there are no infinite derivations*

$$U \Rightarrow_{\sigma_1} V \Rightarrow_{\sigma_2} W \Rightarrow_{\sigma_3} \cdots$$

(these are useful if we want to report a reason why particular terms do not unify)

## Unification: Properties

### Theorem

- *there are no infinite derivations*

$$U \Rightarrow_{\sigma_1} V \Rightarrow_{\sigma_2} W \Rightarrow_{\sigma_3} \cdots$$

- *if $s$ and $t$ are unifiable then for every maximal derivation*

$$s \approx t \Rightarrow_{\sigma_1} E_1 \Rightarrow_{\sigma_2} E_2 \Rightarrow_{\sigma_3} \cdots \Rightarrow_{\sigma_n} E_n$$

(these are useful if we want to report a reason why particular terms do not unify)

## Unification: Properties

### Theorem

- *there are no infinite derivations*

$$U \Rightarrow_{\sigma_1} V \Rightarrow_{\sigma_2} W \Rightarrow_{\sigma_3} \cdots$$

- *if $s$ and $t$ are unifiable then for every maximal derivation*

$$s \approx t \Rightarrow_{\sigma_1} E_1 \Rightarrow_{\sigma_2} E_2 \Rightarrow_{\sigma_3} \cdots \Rightarrow_{\sigma_n} E_n$$

  - $E_n = \square$

(these are useful if we want to report a reason why particular terms do not unify)

## Unification: Properties

### Theorem

- *there are no infinite derivations*

$$U \Rightarrow_{\sigma_1} V \Rightarrow_{\sigma_2} W \Rightarrow_{\sigma_3} \cdots$$

- *if $s$ and $t$ are unifiable then for every maximal derivation*

$$s \approx t \Rightarrow_{\sigma_1} E_1 \Rightarrow_{\sigma_2} E_2 \Rightarrow_{\sigma_3} \cdots \Rightarrow_{\sigma_n} E_n$$

  - $E_n = \square$
  - $\sigma_1\sigma_2\sigma_3\cdots\sigma_n$ *is mgu of $s$ and $t$*

(these are useful if we want to report a reason why particular terms do not unify)

## Unification: Properties

### Theorem

- *there are no infinite derivations*

$$U \Rightarrow_{\sigma_1} V \Rightarrow_{\sigma_2} W \Rightarrow_{\sigma_3} \cdots$$

- *if $s$ and $t$ are unifiable then for every maximal derivation*

$$s \approx t \Rightarrow_{\sigma_1} E_1 \Rightarrow_{\sigma_2} E_2 \Rightarrow_{\sigma_3} \cdots \Rightarrow_{\sigma_n} E_n$$

  - $E_n = \square$
  - $\sigma_1\sigma_2\sigma_3\cdots\sigma_n$ *is mgu of $s$ and $t$*

### Optional Failure Rules

$$\frac{E_1,\ f(s_1,\ldots,s_n) \approx g(t_1,\ldots,t_m),\ E_2}{\perp} \qquad \frac{E_1,\ x \approx t,\ E_2}{\perp} \quad \frac{E_1,\ t \approx x,\ E_2}{\perp}$$

if $x$ occurs in $t$

(these are useful if we want to report a reason why particular terms do not unify)

## Unification beyond first-order logic

We've seen unification for first-order logic terms

But unification also works for example on types (e.g. types in simply-typed $\lambda$-calculus, or in functional programming)

We'll introduce a minimal ML (Ocaml / SML) like language and show how unification is used to find most general types of expressions (programming language interpreters do this!)

Warning: There is a notation change in how substitutions are represented in logic and in functional programming. Be careful! (In the exam or in final assignment I will accept any notation)

## Core ML

### Definition (Expressions)

$$e ::= \overbrace{x \mid e\ e \mid \lambda x.e}^{\lambda\text{-Calculus}} \mid \underbrace{c}_{\text{primitives/constants}} \mid \underbrace{\textbf{let}\ x = e\ \textbf{in}\ e}_{\text{let binding}} \mid \underbrace{\textbf{if}\ e\ \textbf{then}\ e\ \textbf{else}\ e}_{\text{conditional}}$$

### Primitives

**Boolean:** true, false, $<$, $>$, ...

**Arithmetic:** $\times$, $+$, $\div$, $-$, 0, 1, ...

**Tuples:** pair, fst, snd

**Lists:** nil, cons, hd, tl

## What is Type Checking?

- Given some environment (assigning types to primitives)
- together with a core ML expression and a type
- check whether the expression really has that type
- (with respect to that environment)

## Preliminaries

### Definition (Types)

$$\tau ::= \underbrace{\alpha}_{\text{type variable}} \mid \overbrace{\tau \to \tau}^{\text{function type constructor}} \mid \underbrace{g(\tau, \ldots, \tau)}_{\text{data type constructor}}$$

### Convention

- type variables $\alpha$, $\alpha_0$, $\alpha_1$, . . . , $\beta$, $\beta_0$, . . .
- function type constructor '$\to$' is right associative
- base data type constructors: int, bool (instead of int(), bool())

### Example

int $\to$ bool, (int $\to$ list(int)) $\to$ bool, list($\alpha_0$) $\to$ int, . . .

## Preliminaries (cont'd)

(Typing) environment $E$: maps (variables and) primitives to types

$(e : \tau) \in E$ "*$e$ is of type $\tau$ in $E$*"

(note: parentheses around $e : \tau$ will be usually dropped)

(Typing) judgment:

$E \vdash e : \tau$ "*it can be proved that expression $e$ has type $\tau$ in environment $E$*"

### Example

- environment $P = \{+ : \text{int} \to \text{int} \to \text{int}, \text{nil} : \text{list}(\alpha), \text{true} : \text{bool}, \ldots\}$
- judgement $P \vdash \text{true} : \text{bool}$
- judgement $P \nvdash \text{true} : \text{int}$

### Convention

$E, e : \tau$ abbreviates $E \cup \{e : \tau\}$

## The Type Checking System $\mathcal{C}$

$$\frac{}{E, e : \tau \vdash e : \tau}\ \text{(ref)} \qquad \frac{E \vdash e_1 : \tau_2 \to \tau_1 \quad E \vdash e_2 : \tau_2}{E \vdash e_1\ e_2 : \tau_1}\ \text{(app)}$$

$$\frac{E, x : \tau_1 \vdash e : \tau_2}{E \vdash \lambda x.e : \tau_1 \to \tau_2}\ \text{(abs)} \qquad \frac{E \vdash e_1 : \tau_1 \quad E, x : \tau_1 \vdash e_2 : \tau_2}{E \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau_2}\ \text{(let)}$$

$$\frac{E \vdash e_1 : \text{bool} \quad E \vdash e_2 : \tau \quad E \vdash e_3 : \tau}{E \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : \tau}\ \text{(ite)}$$

- environment $E = \{\text{true} : \text{bool}, + : \text{int} \to \text{int} \to \text{int}\}$
- judgment $E \vdash (\lambda x.x)\ \text{true} : \text{bool}$

Proof.

$$\dfrac{\dfrac{E, x : \text{bool} \vdash x : \text{bool}}{E \vdash \lambda x.x : \text{bool} \to \text{bool}}\,(\text{abs}) \qquad E \vdash \text{true} : \text{bool}}{E \vdash (\lambda x.x)\ \text{true} : \text{bool}}\,(\text{app})$$

■

- environment $E = \{\mathsf{true} : \mathsf{bool}, + : \mathsf{int} \to \mathsf{int} \to \mathsf{int}\}$
- judgment $E \vdash \lambda x.x + x : \mathsf{int} \to \mathsf{int}$

Try it! If you have issues ask on the chat.

## What is Type Inference?

- Given some environment
- together with a core ML expression
- and a type
- infer a unifier (type substitution) —if possible—
- such that the most general type of the expression is obtained

## Preliminaries (switch to functional notation)

Type variables:

$$\mathcal{TV}\mathsf{ar}(\tau) \overset{\text{def}}{=} \begin{cases} \{\alpha\} & \text{if } \tau = \alpha \\ \mathcal{TV}\mathsf{ar}(\tau_1) \cup \mathcal{TV}\mathsf{ar}(\tau_2) & \text{if } \tau = \tau_1 \to \tau_2 \\ \bigcup_{1 \le i \le n} \mathcal{TV}\mathsf{ar}(\tau_i) & \text{if } \tau = g(\tau_1, \ldots, \tau_n) \end{cases}$$

Type substitution: $\sigma$ is mapping from type variables to types

Application:

$$\tau\sigma \overset{\text{def}}{=} \begin{cases} \sigma(\alpha) & \text{if } \tau = \alpha \\ \tau_1\sigma \to \tau_2\sigma & \text{if } \tau = \tau_1 \to \tau_2 \\ g(\tau_1\sigma, \ldots, \tau_n\sigma) & \text{if } \tau = g(\tau_1, \ldots, \tau_n) \end{cases}$$

$$E\sigma \overset{\text{def}}{=} \{e : \tau\sigma \mid e : \tau \in E\}$$

Composition: $\sigma_1\sigma_2 \overset{\text{def}}{=} \sigma_2 \circ \sigma_1$, i.e., $\alpha \mapsto \sigma_2(\sigma_1(\alpha))$

## Example

$$\begin{aligned}
\tau &= \alpha \to (\alpha_1 \to \alpha_3) \\
\sigma &= \{\alpha/\mathsf{int} \to \mathsf{int}, \alpha_1/\mathsf{list}(\alpha_2)\} \\
\sigma_2 &= \{\alpha_3/\alpha_4, \alpha_2/\alpha, \alpha/\alpha_1\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{TV}\mathsf{ar}(\tau) &= \{\alpha, \alpha_1, \alpha_3\} \\
\tau\sigma &= (\mathsf{int} \to \mathsf{int}) \to (\mathsf{list}(\alpha_2) \to \alpha_3) \\
\mathcal{TV}\mathsf{ar}(\tau\sigma) &= \{\alpha_2, \alpha_3\} \\
\sigma\sigma_2 &= \{\alpha/\mathsf{int} \to \mathsf{int}, \alpha_1/\mathsf{list}(\alpha), \alpha_3/\alpha_4, \alpha_2/\alpha\}
\end{aligned}$$

## Unification Problems

### Definition

- unification problem is (finite) sequence of equations

$$\tau_1 \approx \tau_1'; \ldots; \tau_n \approx \tau_n'$$

- $\square$ denotes empty sequence
- type substitution $\sigma$ is unifier of unification problem if

$$\tau_1\sigma = \tau_1'\sigma; \ldots; \tau_n\sigma = \tau_n'\sigma$$

- process of computing a unifier is called unification

## The Unification System $\mathcal{U}$

$$\frac{E_1; g(\tau_1, \ldots, \tau_n) \approx g(\tau_1', \ldots, \tau_n'); E_2}{E_1; \tau_1 \approx \tau_1'; \ldots; \tau_n \approx \tau_n'; E_2}\ (\mathrm{d}_1)$$

$$\frac{E_1; \tau_1 \to \tau_2 \approx \tau_1' \to \tau_2'; E_2}{E_1; \tau_1 \approx \tau_1'; \tau_2 \approx \tau_2'; E_2}\ (\mathrm{d}_2)$$

$$\frac{E_1; \alpha \approx \tau; E_2 \quad \alpha \notin \mathcal{TV}\mathrm{ar}(\tau)}{(E_1; E_2)\{\alpha/\tau\}}\ (\mathrm{v}_1)$$

$$\frac{E_1; \tau \approx \alpha; E_2 \quad \alpha \notin \mathcal{TV}\mathrm{ar}(\tau)}{(E_1; E_2)\{\alpha/\tau\}}\ (\mathrm{v}_2)$$

$$\frac{E_1; \tau \approx \tau; E_2}{E_1; E_2}\ (\mathrm{t})$$

## Unification Problem (cont'd)

Notation

$E \Rightarrow_{\sigma}^{(r)} E'$ if rule $r$ from $\mathcal{U}$ applied to equations $E$ yields $E'$

## Unification Problem (cont'd)

### Notation
$E \Rightarrow_{\sigma}^{(r)} E'$ if rule $r$ from $\mathcal{U}$ applied to equations $E$ yields $E'$

### Theorem
*if* $E_1 \Rightarrow_{\sigma_1}^{(r_1)} E_2 \Rightarrow_{\sigma_2}^{(r_2)} \dots \Rightarrow_{\sigma_{n-1}}^{(r_{n-1})} \square$ *then* $E_1$ *has unifier* $\sigma_1 \cdots \sigma_{n-1}$

## Unification Problem (cont'd)

Notation

$E \Rightarrow_{\sigma}^{(r)} E'$ if rule $r$ from $\mathcal{U}$ applied to equations $E$ yields $E'$

Theorem

*if* $E_1 \Rightarrow_{\sigma_1}^{(r_1)} E_2 \Rightarrow_{\sigma_2}^{(r_2)} \dots \Rightarrow_{\sigma_{n-1}}^{(r_{n-1})} \square$ *then* $E_1$ *has unifier* $\sigma_1 \cdots \sigma_{n-1}$

Example

$$\begin{aligned} \mathrm{list}(\mathrm{bool}) \approx \mathrm{list}(\alpha) \quad &\Rightarrow_{\iota}^{(\mathrm{d}_1)} \quad \mathrm{bool} \approx \alpha \\ &\Rightarrow_{\{\alpha/\mathrm{bool}\}}^{(\mathrm{v}_2)} \quad \square \end{aligned}$$

## Unification Problem (cont'd)

### Notation
$E \Rightarrow^{(r)}_{\sigma} E'$ if rule $r$ from $\mathcal{U}$ applied to equations $E$ yields $E'$

### Theorem
*if* $E_1 \Rightarrow^{(r_1)}_{\sigma_1} E_2 \Rightarrow^{(r_2)}_{\sigma_2} \ldots \Rightarrow^{(r_{n-1})}_{\sigma_{n-1}} \square$ *then* $E_1$ *has unifier* $\sigma_1 \cdots \sigma_{n-1}$

### Example

$$\begin{aligned} \mathsf{list}(\mathsf{bool}) \approx \mathsf{list}(\alpha) \quad & \Rightarrow^{(\mathsf{d}_1)}_{\iota} \quad \mathsf{bool} \approx \alpha \\ & \Rightarrow^{(\mathsf{v}_2)}_{\{\alpha/\mathsf{bool}\}} \quad \square \end{aligned}$$

### Remarks
- unification always terminates
- the order of applying inference rules has no (dramatic) effect

## Type Inference Problems

- $E \rhd e : \alpha_0$ is type inference problem
- $\sigma$ s.t., $E\sigma \vdash e : \alpha_0\sigma$ (if exists) is solution (otherwise: $e$ not typable)

## The Type Inference System $\mathcal{I}$

$$\frac{E, e : \tau_0 \rhd e : \tau_1}{\tau_0 \approx \tau_1} \text{ (con)} \qquad \frac{E \rhd e_1\ e_2 : \tau}{E \rhd e_1 : \alpha \to \tau; E \rhd e_2 : \alpha} \text{ (app)}$$

$$\frac{E \rhd \lambda x.e : \tau}{E, x : \alpha_1 \rhd e : \alpha_2; \tau \approx \alpha_1 \to \alpha_2} \text{ (abs)} \qquad \frac{E \rhd \textbf{let } x = e_1 \textbf{ in } e_2 : \tau}{E \rhd e_1 : \alpha; E, x : \alpha \rhd e_2 : \tau} \text{ (let)}$$

$$\frac{E \rhd \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : \tau}{E \rhd e_1 : \text{bool}; E \rhd e_2 : \tau; E \rhd e_3 : \tau} \text{ (ite)}$$

## Recipe - Type Inference

### Input

core ML expression $e$ and typing environment $E$

## Recipe - Type Inference

### Input

core ML expression $e$ and typing environment $E$

### Algorithm

1. start with $E \triangleright e : \alpha_0$ (fresh type variable $\alpha_0$)
2. use $\mathcal{I}$ to transform $E \triangleright e : \alpha_0$ into unification problem $u$
   (if at any point no rule applicable Not Typable)
3. if possible solve $u$ (obtaining unifier $\sigma$) otherwise Not Typable

## Recipe - Type Inference

### Input

core ML expression $e$ and typing environment $E$

### Algorithm

1. start with $E \rhd e : \alpha_0$ (fresh type variable $\alpha_0$)
2. use $\mathcal{I}$ to transform $E \rhd e : \alpha_0$ into unification problem $u$
   (if at any point no rule applicable Not Typable)
3. if possible solve $u$ (obtaining unifier $\sigma$) otherwise Not Typable

### Output

the most general type of $e$ w.r.t. $E$ is $\alpha_0\sigma$

## Example

Find most general type of **let** $id = \lambda x.x$ **in** $id$ 1 w.r.t. $P$

See the two attached videos! (Or you can try yourself)

## Some notes on unification:

The algorithms for first-order unification are worst-case exponential (Robinson's algorithm).

There exist linear algorithms, but with a really bad constant, so exponential ones are used in practice

Unification for more complex structures quickly becomes hard, in fact higher-order unification is undecidable

## Additional Literature (not required)

More on various properties of unification

Warren D. Goldfarb.
The undecidability of the second-order unification problem.
*Theor. Comput. Sci.*, 13:225–230, 1981.

Krystof Hoder and Andrei Voronkov.
Comparing unification algorithms in first-order theorem proving.
In Bärbel Mertsching, Marcus Hund, and Muhammad Zaheer Aziz, editors, *KI 2009: Advances in Artificial Intelligence, 32nd Annual German Conference on AI, Paderborn, Germany, September 15-18, 2009. Proceedings*, volume 5803 of *Lecture Notes in Computer Science*, pages 435–443. Springer, 2009.

## Summary

### This Lecture

- Unification in Logic
- Unification in Functional Programming

### Next

- Syntactic features
- Model features
- Unification based features
- ML-evaluation
- Machine learning in automated theorem proving