Kapitel 6: Typinferenz

Andreas Abel

LFE Theoretische Informatik
Institut für Informatik
Ludwig-Maximilians-Universität München

6. Juni 2011

Quelle: Martin Wirsing, *Typüberprüfung und Typinferenz*
Foliensatz ProMo SS 2010

Einführende Fragen

- Wozu dient das SML-Typsystem?
- Welche Vorteile haben getypte Sprachen gegenüber ungetypten?
- Wie bestimmt man den Typ eines SML-Programmes? Z.B.
  - `fn x => fn f => f x`
  - `(fn x => x 0) 1`
  - `fn f => fn x => f x + f 1.0`
  - `fn x => x x`
  - `fn x => fn f => x (f x)`
- Inhalt:
  - Typisierungsregeln
  - Typsubstitutionen und Unifikation
  - Typinferenz

Typfehler zur Laufzeit?

- Was passiert bei der Auswertung von `(fn x => x 0) 1`?
- **Keine Typprüfung:** `1` wird als Sprungaddresse interpretiert. Systemabsturz oder schwere Betriebssystem-Ausnahme!
- **Dynamische Typprüfung:** Werte tragen zur Laufzeit Typ-Markierung *Ganzzahl*, *Funktion*, ... Da `1` keine Funktion ist, wird eine Ausnahme im Laufzeitsystem geworfen.
  Nachteil: Vor jeder Operation müssen Typen geprüft werden!
- **Statische Typprüfung:** Der Übersetzer weigert sich, ein Programm mit Typfehlern zu übersetzen. Es kommt nie zu derartigen Laufzeitfehlern.
- Robin Milner (1934-2010): *Well-typed programs do not go wrong.*
- **Der Übersetzer findet bereits viele Programmierfehler.**

Typsysteme in der Praxis

- **Keine Typprüfung:** Assembler
- **Dynamische Typprüfung:** LISP, Scheme, BASIC
  Skriptsprachen: JavaScript, Python
- **Statische Typprüfung:** C (unsicher), JAVA (teilw. dynamisch), Scala, SML, Ocaml, Haskell, Agda.
- Erfahrung mit statisch getypten funktionalen Sprachen:
  If it type-checks, it works.

Ausdrücke (abstrakte Syntax)

- Wir betrachten Programme e im funktionalen Kern von SML, dem λ-Kalkül (1936, Alonzo Church(1902-95)):

e	$::=$	c	Konstante	`2.718`
	$\mid$	x	Variable	`xdiff`
	$\mid$	$e\ e'$	Anwendung	`Math.sqrt 5.0`
	$\mid$	`fn` $x \Rightarrow e$	Funktionsabstraktion	`fn xd => xd - 1`

- Eine Konstante c ist z.B. eine Ganz- oder Fließkommazahl, eine Zeichenkette, oder eine Operation.

c	$::=$	n	Ganzzahl	`0, 1, ~1`
	$\mid$	r	Fließkommazahl	`0.01, 2.0`
	$\mid$	s	Zeichenkette	`"Anton", ""`
	$\mid$	f	Operation	`+, -, o, @`

- Infix-Operatoren `5 + 3` schreiben wir präfix `+ 5 3` in abstrakter Syntax.

Typausdrücke und Typisierungskontexte

- Wir beschränken uns auf Grund- und Funktionstypen.
- Grammatik für die abstrakte Syntax von Typen:

A, B	$::=$	$\alpha \mid \beta \mid \ldots$	Typvariable	`'a, 'b`
	$\mid$	$\texttt{int} \mid \texttt{real} \mid \texttt{string} \mid \ldots$	Grundtyp	
	$\mid$	$A \to B$	Funktionstyp	`int -> 'a`

- Klammern sind in der Grammatik nicht erwähnt, sie dienen nur der Disambiguierung.

$$A \to B \to C \;=\; A \to (B \to C) \;\neq\; (A \to B) \to C$$

Typzuweisung

- Eine Typzuweisung $e : A$ ist ein Paar aus einem SML-Ausdruck e und einen SML-Typen A, interpretiert als Aussage "e *hat Typ* A".

`4`	:	`int`	wahr
`true`	:	`string`	falsch
`3.3`	:	`'a`	falsch
`fn x => x - 1`	:	`int -> int`	wahr
`fn x => fn y => x`	:	`'a -> 'b -> 'a`	wahr
`fn x => fn y => y`	:	`'a -> 'b -> 'a`	falsch
`z (x + 0)`	:	`bool`	???

- Der Typausdruck `z (x + 0)` enthält freie Variablen `x` und `z`.
- Wir benötigen eine Typzuweisung für die freien Variablen!

Typisierungskontext

- Eine endliche Menge $\Gamma = \{x_1{:}A_1, \ldots x_n{:}A_n\}$ $(n \geq 0)$ von Typzuweisungen an **paarweise verschiedene** Variablen $x_1, \ldots, x_n$ heißt **Typisierungskontext** (engl. *typing context/environment*).

$\{\mathtt{b} : \mathtt{bool},\ \mathtt{half} : \mathtt{int} \to \mathtt{int}\}$	gültig
$\{\mathtt{z} : \mathtt{int} \to \mathtt{bool},\ \mathtt{x} : \mathtt{int}\}$	gültig
$\{\mathtt{h} : \mathtt{int} \to \mathtt{bool},\ \mathtt{h} : \mathtt{int}\}$	ungültig

- Γ ist eine **endliche Abbildung** von Variablen x auf Typen $\Gamma(x)$.
- Wir schreiben $\Gamma, x{:}A$ für das **Einfügen** der Typzuweisung $x{:}A$ in Γ. Eine schon vorhandene Typzuweisung für x wird überschrieben.
- Bsp.: $\{h : \mathtt{int} \to \mathtt{bool}\}, h : \mathtt{int} = \{h : \mathtt{int}\}$.

Typisierung

- Ein **Typ(isierungs)urteil** (engl: *typing judgement*) ist eine Aussage der Form $\boxed{\Gamma \vdash e : A}$.
- Bedeutung: "Unter der Annahme, dass die Variablen die in Γ angegebenen Typen haben, hat e den Typ A".
- Kurz: "Im Kontext Γ hat e den Typ A".
- Formal ist $\_ \vdash \_ : \_$ eine dreistellige Relation zwischen Typkontexten, Ausdrücken, und Typen.

$$
\begin{array}{llll}
\{\texttt{y} : \texttt{int}\} & \vdash \ \texttt{fn x} \Rightarrow \texttt{+ y (* 2 x)} & : \ \texttt{int} \to \texttt{int} & \text{gültig} \\
\{\texttt{x} : \texttt{'a}\} & \vdash \ \texttt{fn y} \Rightarrow \texttt{x} & : \ \texttt{'b} \to \texttt{'a} & \text{gültig} \\
\{\} & \vdash \ \texttt{fn x} \Rightarrow \texttt{+ x x} & : \ \texttt{'a} \to \texttt{'a} & \text{ungültig}
\end{array}
$$

Typregeln

- Gültige Typurteile werden mittels Typ**regeln hergeleitet**.
- Eine Regel hat die Form $P_1 \wedge \cdots \wedge P_n \implies K$, d.h., aus den **Prämissen** $P_1 \ldots P_n$ folgt die **Konklusion** K, geschrieben

$$\frac{P_1 \quad \ldots \quad P_n}{K}$$

- In unserem Fall sind die P_i und K Typurteile.
- Regeln ohne Prämissen heißen **Axiome**.
- Für jede Konstante c gibt es ein Typaxiom, z.B.

$$\begin{array}{lcl} \Gamma \vdash 1 & : & \texttt{int} \\ \Gamma \vdash \texttt{true} & : & \texttt{bool} \\ \Gamma \vdash + & : & \texttt{int} \to \texttt{int} \to \texttt{int} \end{array}$$

- Γ ist jeweils beliebig, z.B. $\Gamma = \{\}$ oder $\Gamma = \{\texttt{x} : \texttt{int}, \texttt{b} : \texttt{bool}\}$.
- $\Gamma \vdash 1 : \texttt{int}$ ist eigentlich ein **Axiomenschema**, denn für jedes Γ gibt es ein Axiom.

Typisierung von Variablen

- Der Typ einer Variablen ist im Kontext notiert:

$$\frac{(x:A) \in \Gamma}{\Gamma \vdash x : A}\ \text{VAR}$$

- Äquivalent:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ \text{VAR}$$

- Beispiele:

$\{x : \texttt{real}\}$	$\vdash$	$x : \texttt{real}$	gültig
$\{x : \texttt{real}, y : \texttt{int}, z : \texttt{bool}\}$	$\vdash$	$y : \texttt{int}$	gültig
$\{x : \texttt{real}, y : \texttt{int}, z : \texttt{bool}\}$	$\vdash$	$y : \texttt{real}$	falscher Typ
$\{\}$	$\vdash$	$x : \texttt{real}$	ungebundene Variable

Typisierung der Anwendung

- Der Typ der Anwendung einer Funktion e vom Typ $A \to B$ auf ein Argument e' vom Typ A ist B.

$$\frac{\Gamma \vdash e : A \to B \qquad \Gamma \vdash e' : A}{\Gamma \vdash e\ e' : B}\ \text{APP}$$

- Falls e keinen Funktionstyp hat oder e' nicht im Definitionsbereich (engl. *domain*) A liegt, ist die Anwendung nicht wohlgetypt.
- Beispiel: $\Gamma = \{\texttt{a} : \texttt{int},\ \texttt{y} : \texttt{int}\}$

$$\frac{\Gamma \vdash \texttt{fn x} \Rightarrow \texttt{+ a x} : \texttt{int} \to \texttt{int} \qquad \Gamma \vdash \texttt{* 2 y} : \texttt{int}}{\Gamma \vdash (\texttt{fn x} \Rightarrow \texttt{+ a x})\ (\texttt{* 2 y}) : \texttt{int}}\ \text{APP}$$

Typisierung der Abstraktion

- Hat e den Typ B unter der Annahme $x : A$, so hat $\texttt{fn}\ x \Rightarrow e$ den Typ $A \to B$.

$$\frac{\Gamma, x{:}A \vdash e : B}{\Gamma \vdash \texttt{fn}\ x \Rightarrow e : A \to B}\ \text{ABS}$$

- Herleitungsbaum (Bsp.):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{x : \alpha,\ f : \alpha \to \beta\} \vdash f : \alpha \to \beta}\ \text{VAR} \quad \dfrac{}{\{x : \alpha,\ f : \alpha \to \beta\} \vdash x : \alpha}\ \text{VAR}}{\{x : \alpha,\ f : \alpha \to \beta\} \vdash f\ x : \beta}\ \text{APP}}{\{x : \alpha\} \vdash \texttt{fn}\ f \Rightarrow f\ x : (\alpha \to \beta) \to \beta}\ \text{ABS}}{\{\} \vdash \texttt{fn}\ x \Rightarrow \texttt{fn}\ f \Rightarrow f\ x : \alpha \to (\alpha \to \beta) \to \beta}\ \text{ABS}$$

Herleitung

- Eine **Herleitung** ist eine Folge von (Typ)aussagen $J_1, \ldots, J_n$, so dass jedes J_i entweder ein Axiom ist, oder mit Konklusion einer Regel mit Prämissen in $J_1, \ldots, J_{i-1}$.
- Die *hergeleitete Aussage* ist die letzte, J_n.
- Herleitung in linearer Form:

$$
\begin{array}{lll}
1 & \{x : \alpha,\ f : \alpha \to \beta\} \vdash f : \alpha \to \beta & \textsc{var} \\
2 & \{x : \alpha,\ f : \alpha \to \beta\} \vdash x : \alpha & \textsc{var} \\
3 & \{x : \alpha,\ f : \alpha \to \beta\} \vdash f\ x : \beta & \textsc{app}(1,2) \\
4 & \{x : \alpha\} \vdash \mathtt{fn}\ f \Rightarrow f\ x : (\alpha \to \beta) \to \beta & \textsc{abs}(3) \\
5 & \{\} \vdash \mathtt{fn}\ x \Rightarrow \mathtt{fn}\ f \Rightarrow f\ x : \alpha \to (\alpha \to \beta) \to \beta & \textsc{abs}(4)
\end{array}
$$

- Eine Aussage ist **herleitbar**, wenn sie eine Herleitung hat.

Kontexterweiterung

- Eine Typaussage bleibt gültig, wenn wir den Typisierungkontext **erweitern**. Bsp:

$$\begin{array}{ll} \{x:\alpha\} & \vdash \ \mathtt{fn}\ f \Rightarrow f\ x : (\alpha \to \beta) \to \beta \\ \{x:\alpha, y:\gamma\} & \vdash \ \mathtt{fn}\ f \Rightarrow f\ x : (\alpha \to \beta) \to \beta \\ \{f:\mathtt{int}\to\mathtt{bool}, x:\alpha, y:\gamma\} & \vdash \ \mathtt{fn}\ f \Rightarrow f\ x : (\alpha \to \beta) \to \beta \end{array}$$

Lemma (Kontexterweiterung/Abschwächung (engl. *weakening*))

Wenn $\Gamma' \supseteq \Gamma$ und $\Gamma \vdash e : A$ herleitbar ist, dann auch $\Gamma' \vdash e : A$.

- Beweisbar durch Induktion über die Länge der Herleitung $J_1, \ldots, J_n, \Gamma \vdash e : A$.
- Die folgende Regel ist also **zulässig** (engl. *admissible*):

$$\frac{\Gamma' \supseteq \Gamma \qquad \Gamma \vdash e : A}{\Gamma' \vdash e : A}\ \textsc{weak}$$

Polymorphie

- Viele Ausdrücke haben mehrere Typisierungen. Z.B. `f x`

$$\begin{array}{ll} \{x : \texttt{int},\ f : \texttt{int} \to \texttt{int}\} & \vdash f\,x : \texttt{int} \\ \{x : \texttt{bool},\ f : \texttt{bool} \to \texttt{int}\} & \vdash f\,x : \texttt{int} \\ \{x : \alpha,\ f : \alpha \to \alpha\} & \vdash f\,x : \alpha \\ \{x : \alpha,\ f : \alpha \to \beta\} & \vdash f\,x : \beta \end{array}$$

- Die letzte dieser Typisierungen ist die **allgemeinste** (engl. *principal typing*); jede andere Typisierung von `f x` erhält man daraus durch Einsetzen von Typen für die Typvariablen α und β (**Instanziierung**).

$$\{x : \alpha,\ f : \alpha \to \beta\}[\texttt{bool}/\alpha,\ \texttt{int}/\beta] \vdash f\,x : \beta[\texttt{bool}/\alpha,\ \texttt{int}/\beta]$$

- Die allgemeinste Typisierung ist bis auf Typvariablen-Umbenennung eindeutig.

Typsubstitutionen

- Eine Typsubstitution σ ist eine endliche Abbildung von Typvariablen α auf Typausdrücke A.

$$\begin{array}{rcl}
\sigma_1 & = & [\mathtt{bool}/\alpha, \mathtt{int}/\beta] \\
& = & [\mathtt{bool}/\alpha, \mathtt{int}/\beta, \gamma/\gamma] \\
& = & [\mathtt{bool}/\alpha, \mathtt{int}/\beta, \gamma/\gamma, \delta/\delta] \ldots \\
\sigma_2 & = & [\beta/\alpha, (\beta \to \mathtt{bool})/\beta, \mathtt{int}/\gamma] \\
\sigma_3 & = & [\gamma/\alpha, \delta/\beta] \\
\sigma_4 & = & [\beta/\alpha, \alpha/\beta]
\end{array}$$

- Für die Anwendung $A\sigma$ einer Substitution σ auf Typen A gilt z.B.:

$$\begin{array}{lcl}
\mathtt{int}\,\sigma & = & \mathtt{int} \\
(A \to B)\sigma & = & A\sigma \to B\sigma \\
\alpha[\mathtt{bool}/\alpha, \mathtt{int}/\beta] & = & \mathtt{bool}
\end{array}$$

Instanziierung

- Die Anwendung $\Gamma\sigma$ einer Substitution σ auf einen Typkontext Γ ist punktweise definiert: $(\Gamma\sigma)(x) = (\Gamma(x))\sigma$.

$$\{x : \mathtt{int},\ y : \alpha \to \beta,\ z : \alpha\}[\gamma/\alpha, \mathtt{int}/\beta, \mathtt{bool}/\gamma] \quad = \\ \{x : \mathtt{int},\ y : \gamma \to \mathtt{int},\ z : \gamma\}$$

- Alle Typaxiome und -regeln bleiben unter Substitution gültig.

Lemma (Typerhaltung unter Substitution)

Wenn $\Gamma \vdash e : A$, *dann* $\Gamma\sigma \vdash e : A\sigma$.

- Zulässige Regel:

$$\frac{\Gamma \vdash e : A}{\Gamma\sigma \vdash e : A\sigma}\ \textsc{subst}$$

Substitutionskomposition

- Die **Komposition** $\sigma\sigma'$ ist definiert durch $A(\sigma\sigma') = (A\sigma)\sigma'$.
- Berechnung der Komposition zweier Substitutionen:

$$\begin{array}{rcl}
\alpha[(\alpha \to \beta)/\alpha][\mathtt{int}/\alpha,\ \mathtt{bool}/\beta] & = & (\alpha \to \beta)[\mathtt{int}/\alpha,\ \mathtt{bool}/\beta] \\
 & = & \mathtt{int} \to \mathtt{bool} \\
\beta[(\alpha \to \beta)/\alpha][\mathtt{int}/\alpha,\ \mathtt{bool}/\beta] & = & \beta[\mathtt{int}/\alpha,\ \mathtt{bool}/\beta] \\
 & = & \mathtt{bool} \\
\hline
[(\alpha \to \beta)/\alpha][\mathtt{int}/\alpha,\ \mathtt{bool}/\beta] & = & [(\mathtt{int} \to \mathtt{bool})/\alpha,\ \mathtt{bool}/\beta]
\end{array}$$

- Komposition ist **assoziativ**, $\sigma(\sigma'\sigma'') = (\sigma\sigma')\sigma''$, aber nicht kommutativ.
- Die **Identität**ssubstitution $\mathrm{id} = []$ ist **neutrales Element** der Komposition, es gilt $\mathrm{id}\,\sigma = \sigma$ und $\sigma\,\mathrm{id} = \sigma$.

Prinzipale Typisierung

- Eine Typisierung $\Gamma \vdash e : A$ eines Ausdrucks e heißt **prinzipal** oder **allgemeinst**möglich, falls für jede weitere Typisierung $\Gamma' \vdash e : A'$ von e eine Substitution σ gibt, so dass $A' = A\sigma$ und $\Gamma' \supseteq \Gamma\sigma$.

$$
\begin{array}{lll}
\{x : \alpha,\ f : \alpha \to \beta\} & \vdash f\,x : \beta & \text{prinzipal} \\
\{x : \mathtt{int},\ f : \mathtt{int} \to \mathtt{int}\} & \vdash f\,x : \mathtt{int} & \sigma = [\mathtt{int}/\alpha, \mathtt{int}/\beta] \\
\{x : \alpha,\ f : \alpha \to \beta,\ y : \gamma\} & \vdash f\,x : \beta & \sigma = \mathrm{id} \\
\{x : \gamma,\ f : \gamma \to \beta\} & \vdash f\,x : \beta & \sigma = [\gamma/\alpha] \quad \text{(auch prinzipal)}
\end{array}
$$

- Überladene Operatoren haben *keine* allgemeinste Typisierung.

$$
\begin{array}{lll}
\mathtt{+} & : & \mathtt{int} \to \mathtt{int} \to \mathtt{int} \\
\mathtt{+} & : & \mathtt{real} \to \mathtt{real} \to \mathtt{real}
\end{array}
$$

- Ohne Überladung hat jeder SML-Ausdruck eine prinzipale Typisierung.

Typinferenz

- **Typinferenz** berechnet zu jedem Ausdruck den allgemeinsten Typ (Fehlermeldung, falls der Ausdruck keinen Typ hat).
- Der prinzipale Typ einer Applikation $e\ e'$ muss aus den prinzipalen Typen von e und e' berechnet werden.

$$\frac{\{\} \vdash e : (\mathtt{int} \to \beta) \to (\gamma \to \beta) \qquad \{x : \alpha\} \vdash e' : \alpha \to \mathtt{real}}{\{x : \mathtt{int}\} \vdash e\ e' : \gamma \to \mathtt{real}}$$

- Beide Prämissen müssen mit $[\mathtt{int}/\alpha,\ \mathtt{real}/\beta]$ instanziiert werden.
- Allgemeinste Lösung der Gleichung $\mathtt{int} \to \beta = \alpha \to \mathtt{real}$.

$$\frac{\{f : \alpha\} \vdash f : \alpha \qquad \{x : \beta\} \vdash x : \beta}{\{f : \beta \to \gamma, x : \beta\} \vdash f\ x : \gamma}$$

- Löse Gleichung $\alpha = \beta \to \gamma$ für neue Typvariable γ.

Unifikation

- Eine Substitution σ ist **allgemeiner** als σ', falls es eine Substitution τ gibt mit $\sigma' = \sigma\tau$.
- Die **speziellere** Substitution σ' ist also eine Instanz von σ.
- Ein **Unifikator** zweier Typen A und A' ist eine Substitution σ, so dass $A\sigma = A'\sigma$.

$$(\alpha \to \beta)[\mathtt{int}/\alpha,\ \mathtt{int}/\beta] = (\mathtt{int} \to \beta)[\mathtt{int}/\alpha,\ \mathtt{int}/\beta]$$

- Der **allgemeinste Unifikator** von A und A' ist die allgemeinste Substitution σ, so dass $A\sigma = A'\sigma$.

$$(\alpha \to \beta)[\mathtt{int}/\alpha] = (\mathtt{int} \to \beta)[\mathtt{int}/\alpha]$$

Robinson's Unifikationsalgorithmus

Der Unifikationsalgorithmus unify arbeitet auf einer Menge $E = \{A_1 = B_1, \ldots, A_n = B_n\}$ von formalen Typgleichungen und liefert

- die allgemeinste Substitution σ, so dass $A_1\sigma = B_1\sigma$, ..., $A_n\sigma = B_n\sigma$,
- oder einen Typfehler.

1. $\mathsf{unify}(\{\}) = \mathsf{id}$ (* Fertig! *)
2. $\mathsf{unify}(\{A = A\} \uplus E) = \mathsf{unify}(E)$ (* redundante Gleichung *)
3. $\mathsf{unify}(\{A \to A' = B \to B'\} \uplus E) = \mathsf{unify}(\{A = B, A' = B'\} \uplus E)$
4. $\mathsf{unify}(\{\alpha = B\} \uplus E) = \mathsf{unify}(\{B = \alpha\} \uplus E) =$
   if α occurs in B then "Error: circularity"
   else let $\sigma = [B/\alpha]$ in $\sigma\,\mathsf{unify}(E\sigma)$ (* Substitutionskomposition! *)
5. Sonst $\mathsf{unify}(E) =$ "Error: type mismatch" (*z.B. $(\texttt{int} = \texttt{bool}) \in E$*)

Beispiele Unifikation

- Nur Variablen:

$$\begin{array}{lll} \mathsf{unify}\{\alpha = \beta, \beta = \gamma\} & = [\beta/\alpha]\mathsf{unify}\{\beta = \gamma\} & = [\beta/\alpha][\gamma/\beta]\mathsf{unify}\{\} \\ = [\beta/\alpha][\gamma/\beta]\mathsf{id} & = [\beta/\alpha][\gamma/\beta] & = [\gamma/\alpha, \gamma/\beta] \end{array}$$

- Typkonstruktorkonflikt:

$$\begin{array}{ll} \mathsf{unify}\{\mathtt{int} \to \alpha = \alpha \to \mathtt{real}\} & = \mathsf{unify}\{\mathtt{int} = \alpha,\ \alpha = \mathtt{real}\} \\ = [\mathtt{int}/\alpha]\mathsf{unify}\{\mathtt{int} = \mathtt{real}\} & = \text{"Error: type mismatch"} \end{array}$$

- Zirkulärer Typ:

$$\begin{array}{l} \mathsf{unify}\{\beta = \beta, \alpha = \alpha \to \mathtt{real}\} = \mathsf{unify}\{\alpha = \alpha \to \mathtt{real}\} \\ = \text{"Error: circularity"} \end{array}$$

Typinferenzalgorithmus

$\text{infer}_\Gamma(e) = (A, \sigma)$ nimmt einen Ausdruck e in Typkontext Γ und liefert den allgemeinsten Typ A von e samt der allgemeinsten Instanziierung σ von Γ, so dass $\Gamma\sigma \vdash e : A$.

1. $\text{infer}_\Gamma(x) =$ if $(x : A) \in \Gamma$ then (A, id)
   else "Error: unbound variable"
2. $\text{infer}_\Gamma(c) =$ let $A =$ type of constant c
   in (A, id)
3. $\text{infer}_\Gamma(\texttt{fn}\ x \Rightarrow e) =$
   let α new (previously unused) type variable
   let $(B, \sigma) = \text{infer}_{\Gamma, x:\alpha}(e)$ (* hence $(\Gamma\sigma,\ x : \alpha\sigma) \vdash e : B$ *)
   in $(\alpha\sigma \to B,\ \sigma)$ (* thus, $\Gamma\sigma \vdash \texttt{fn}\ x \Rightarrow e : \alpha\sigma \to B$ *)

Typinferenzalgorithmus (Applikation)

$\mathsf{infer}_\Gamma(e) = (A, \sigma)$ nimmt einen Ausdruck e in Typkontext Γ und liefert den allgemeinsten Typ A von e samt der allgemeinsten Instanziierung σ von Γ, so dass $\Gamma\sigma \vdash e : A$.

4. $\mathsf{infer}_\Gamma(e\ e') =$
   let $(C, \sigma) = \mathsf{infer}_\Gamma(e)$ (* $\Gamma\sigma \vdash e : C$ *) (* $\Gamma\sigma\sigma' \vdash e : C\sigma'$ *)
   let $(A, \sigma') = \mathsf{infer}_{\Gamma\sigma}(e')$ (* $\Gamma\sigma\sigma' \vdash e' : A$ *)
   let $\sigma'' = \mathsf{unify}\{C\sigma' = A \to \beta\}$ for new β (* $C\sigma'\sigma'' = A\sigma'' \to \beta\sigma''$ *)
   in $(\beta\sigma'',\ \sigma\sigma'\sigma'')$ (* $\Gamma\sigma\sigma'\sigma'' \vdash e\ e' : \beta\sigma''$ *)

Dieser Algorithmus heißt $\mathcal{W}$ und geht auf Roger Hindley und Robin Milner zurück.

Zusammenfassung

- *Well-typed programs do not go wrong* (Robin Milner)
- Typisierung $\Gamma \vdash e : A$.
- Typaxiome, Typregeln, Typherleitung.
- Polymorphie und allgemeinste (prinzipale) Typen.
- SML-Typinferenz berechnet zu jedem Ausdruck prinzipalen Typ.
- Typinferenz beruht auf Unifikation (Robinson).