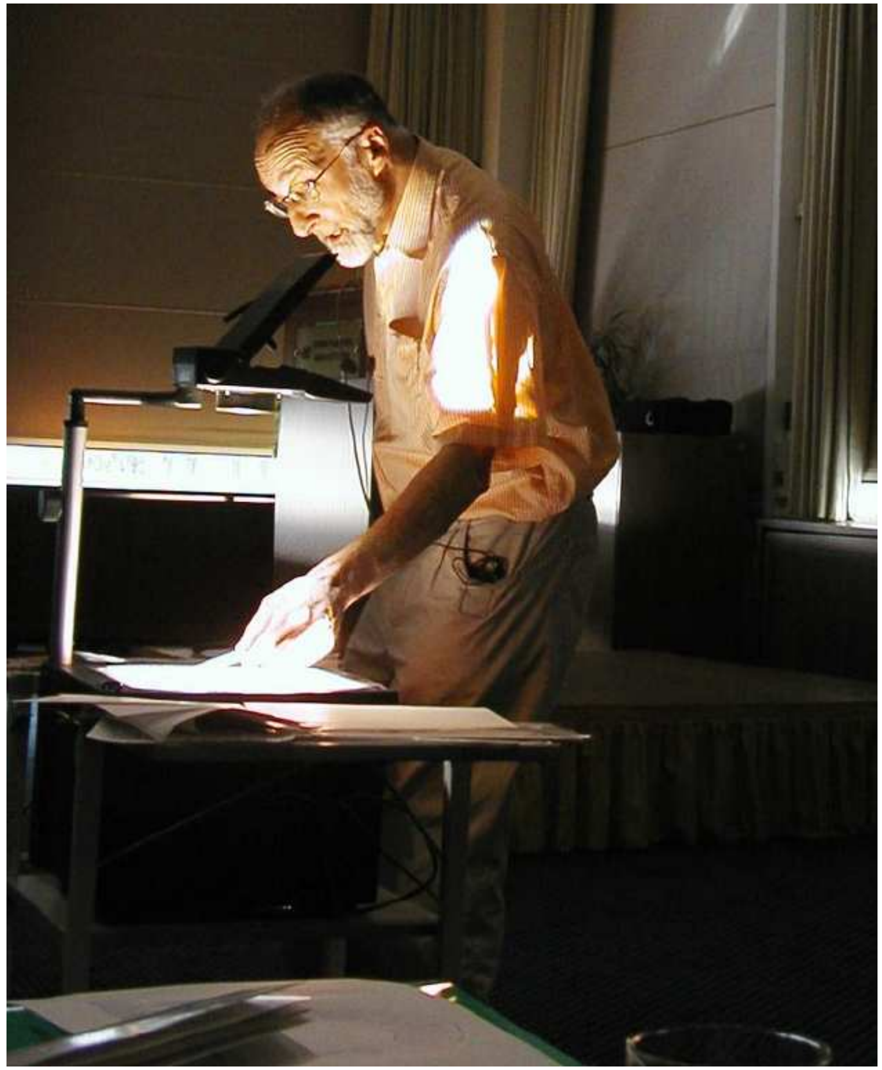
3.3 Inferieren von Typen

- Im Gegensatz zu imperativen Sprachen kann in funktionalen Programmiersprachen der Typ von Bezeichnern (i.a.) weggelassen werden.
- Diese werden dann automatisch hergeleitet :-)

Beispiel:

$$\begin{array}{lll} \mathbf{fun}\ \ \mathsf{fac}\,\mathsf{x} & = & \mathbf{if}\ \ \mathsf{x} \leq 0\ \ \mathbf{then}\ \ 1 \\ & & \mathbf{else}\ \ \mathsf{x} \cdot \mathsf{fac}\,(\mathsf{x} - 1) \end{array}$$

Dafür findet der SML-Compiler: $\mathsf{fac} : \mathbf{int} \to \mathbf{int}$



Robin (Dumbledore) Milner, Edinburgh

Idee: J.R. Hindley, R. Milner

Stelle Axiome und Regeln auf, die den Typ eines Ausdrucks in Beziehung setzen zu den Typen seiner Teilausdrücke :-)

Der Einfachheit halber betrachten wir nur eine funktionale Kernsprache ...

$$
\begin{array}{lll}
e & ::= & b \mid x \mid (\Box_1\ e) \mid (e_1\ \Box_2\ e_2) \\
& \mid & (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2) \\
& \mid & (e_1, \ldots, e_k) \mid [\,] \mid (e_1\ :\ e_2) \\
& \mid & (\textbf{case}\ e_0\ \textbf{of}\ [\,] \ \rightarrow\ e_1;\ h : t\ \rightarrow\ e_2) \\
& \mid & (e_1\ e_2) \mid (\textbf{fn}\ (x_1, \ldots, x_m) \Rightarrow e) \\
& \mid & (\textbf{letrec}\ x_1 = e_1; \ldots; x_n = e_n\ \textbf{in}\ e_0)
\end{array}
$$

Beispiel:

$$
\begin{array}{lll}
\mathbf{letrec} & \mathsf{rev} & = \quad \mathbf{fn}\ x \Rightarrow \mathsf{r}\ x\ [\,]; \\
 & \mathsf{r} & = \quad \mathbf{fn}\ x \Rightarrow \mathbf{fn}\ y \Rightarrow\ \mathbf{case}\ x\ \mathbf{of} \\
 & & \qquad\qquad\qquad\quad [\,] \ \to\ y; \\
 & & \qquad\qquad\qquad\quad h : t \ \to\ \mathsf{r}\ t\ (h\ :\ y) \\
\mathbf{in} & \multicolumn{2}{l}{\mathsf{rev}\ (1\ :\ 2\ :\ 3\ :\ [\,])}
\end{array}
$$

Wir benutzen die üblichen Präzedenz-Regeln und Assoziativitäten, um hässliche Klammern zu sparen :-)

Als einzige Datenstrukturen betrachten wir Tupel und List :-))

Wir benutzen eine Syntax von Typen, die an SML angelehnt ist ...

$$t \quad ::= \quad \mathbf{int} \mid \mathbf{bool} \mid (t_1, \ldots, t_m) \mid \mathsf{list}\ t \mid t_1 \rightarrow t_2$$

Wir betrachten wieder Typ-Aussagen der Form:

$$\Gamma \vdash e : t$$

Wir benutzen eine Syntax von Typen, die an SML angelehnt ist ...

$$t \quad ::= \quad \mathbf{int} \mid \mathbf{bool} \mid (t_1, \ldots, t_m) \mid \mathsf{list}\ t \mid t_1 \to t_2$$

Wir betrachten wieder Typ-Aussagen der Form:

$$\Gamma \vdash e \; : \; t$$

Axiome:

Const:	$\Gamma \vdash c \; : \; t_c$	(t_c Typ der Konstante c)
Var:	$\Gamma \vdash x \; : \; \Gamma(x)$	(x Variable)

Regeln:

Op:
$$\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 + e_2 : \textbf{int}}$$

If:
$$\frac{\Gamma \vdash e_0 : \textbf{bool} \qquad \Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2) : t}$$

Tupel:
$$\frac{\Gamma \vdash e_1 : t_1 \quad \ldots \quad \Gamma \vdash e_m : t_m}{\Gamma \vdash (e_1, \ldots, e_m) : (t_1, \ldots, t_m)}$$

App:
$$\frac{\Gamma \vdash e_1 : t_1 \to t_2 \qquad \Gamma \vdash e_2 : t_1}{\Gamma \vdash (e_1\ e_2) : t_2}$$

Fun:
$$\frac{\Gamma \oplus \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\} \vdash e : t}{\Gamma \vdash \textbf{fn}\ (x_1, \ldots, x_m) \Rightarrow e : (t_1, \ldots, t_m) \to t}$$

...

$\ldots$

Cons: $$\frac{\Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : \mathsf{list}\ t}{\Gamma \vdash (e_1 : e_2) : \mathsf{list}\ t}$$

Case: $$\frac{\Gamma \vdash e_0 : \mathsf{list}\ t_1 \qquad \Gamma \vdash e_1 : t \qquad \Gamma \cup \{x \mapsto t_1, y \mapsto \mathsf{list}\ t_1\} \vdash e_2 : t}{\Gamma \vdash (\textbf{case}\ e_0\ \textbf{of}\ [\,] \to e_1;\ x : y \to e_2) : t}$$

Letrec: $$\frac{\Gamma' \vdash e_1 : t_1 \quad \ldots \quad \Gamma' \vdash e_m : t_m \quad \Gamma' \vdash e_0 : t}{\Gamma \vdash (\textbf{letrec}\ x_1 = e_1; \ldots; x_m = e_m\ \textbf{in}\ e_0) : t}$$

wobei $\Gamma' = \Gamma \oplus \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$

$\dots$

Cons: $$\frac{\Gamma \vdash e_1 \;:\; t \qquad \Gamma \vdash e_2 \;:\; \textsf{list}\ t}{\Gamma \vdash (e_1 : e_2) \;:\; \textsf{list}\ t}$$

Case: $$\frac{\Gamma \vdash e_0 \;:\; \textsf{list}\ t_1 \qquad \Gamma \vdash e_1 \;:\; t \qquad \Gamma \cup \{x \mapsto t_1, y \mapsto \textsf{list}\ t_1\} \vdash e_2 \;:\; t}{\Gamma \vdash (\textbf{case}\ e_0\ \textbf{of}\ [\,] \to e_1;\ x : y \to e_2) \;:\; t}$$

Letrec: $$\frac{\Gamma' \vdash e_1 \;:\; t_1 \quad \dots \quad \Gamma' \vdash e_m \;:\; t_m \quad \Gamma' \vdash e_0 \;:\; t}{\Gamma \vdash (\textbf{letrec}\ x_1 = e_1; \dots; x_m = e_m\ \textbf{in}\ e_0) \;:\; t}$$

wobei $\Gamma' = \Gamma \oplus \{x_1 \mapsto t_1, \dots, x_m \mapsto t_m\}$

Könnten wir die Typen für alle Variablen-Vorkommen raten, ließe sich mithilfe der Regeln überprüfen, dass unsere Wahl korrekt war :-)

$\ldots$

Cons: $$\frac{\Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : \text{list } t}{\Gamma \vdash (e_1 : e_2) : \text{list } t}$$

Case: $$\frac{\Gamma \vdash e_0 : \text{list } t_1 \qquad \Gamma \vdash e_1 : t \qquad \Gamma \cup \{x \mapsto t_1, y \mapsto \text{list } t_1\} \vdash e_2 : t}{\Gamma \vdash (\mathbf{case}\ e_0\ \mathbf{of}\ [\,] \to e_1;\ x : y \to e_2) : t}$$

Letrec: $$\frac{\Gamma' \vdash e_1 : t_1 \quad \ldots \quad \Gamma' \vdash e_m : t_m \quad \Gamma' \vdash e_0 : t}{\Gamma \vdash (\mathbf{letrec}\ x_1 = e_1; \ldots; x_m = e_m\ \mathbf{in}\ e_0) : t}$$

wobei $\Gamma' = \Gamma \oplus \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$

Könnten wir die Typen für alle Variablen-Vorkommen raten, ließe sich mithilfe der Regeln überprüfen, dass unsere Wahl korrekt war :-)

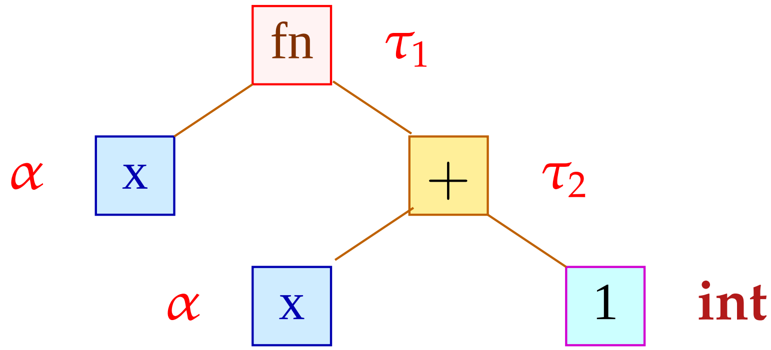
Wie raten wir die Typen der Variablen ???

Idee:

- Mache die Namen der verschiedenen Variablen eindeutig.
- Führe Typ-Variablen für die unbekannten Typen der Variablen und Teilausdrücke ein.
- Sammle die Gleichungen, die notwendigerweise zwischen den Typ-Variablen gelten müssen.
- Finde für diese Gleichungen Lösungen :-)

Beispiel:

$$\mathbf{fn}\ x \Rightarrow x + 1$$



Gleichungen:

$$\begin{array}{lcl} \tau_1 & = & \alpha \to \tau_2 \\ \tau_2 & = & \mathbf{int} \\ \alpha & = & \mathbf{int} \end{array}$$

Wir schließen: $\tau_1 = \mathbf{int} \to \mathbf{int}$

Für jede Programm-Variable x und für jedes Vorkommen eines Teilausdrucks e führen wir die Typ-Variable $\alpha[x]$ bzw. $\tau[e]$ ein.

Jede Regel-Anwendung gibt dann Anlass zu einigen Gleichungen ...

Const:	$e \equiv c$	$\Longrightarrow$	$\tau[e] = \tau_c$
Var:	$e \equiv x$	$\Longrightarrow$	$\tau[e] = \alpha[x]$
Op:	$e \equiv e_1 + e_2$	$\Longrightarrow$	$\tau[e] = \tau[e_1] = \tau[e_2] = \textbf{int}$
Tupel:	$e \equiv (e_1, \ldots, e_m)$	$\Longrightarrow$	$\tau[e] = (\tau[e_1], \ldots, \tau[e_m])$
Cons:	$e \equiv e_1 : e_2$	$\Longrightarrow$	$\tau[e] = \tau[e_2] = \mathsf{list}\ \tau[e_1]$
	$\ldots$		

$\ldots$

If:	$e \equiv \textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2$	$\Longrightarrow$	$\tau[e_0] = \textbf{bool}$ $\tau[e] = \tau[e_1] = \tau[e_2]$
Case:	$e \equiv \textbf{case}\ e_0\ \textbf{of}\ [\,] \ \to\ e_1;\ x : y\ \to\ e_2$	$\Longrightarrow$	$\tau[e_0] = \alpha[y] = \mathsf{list}\ \alpha[x]$ $\tau[e] = \tau[e_1] = \tau[e_2]$
Fun:	$e \equiv \textbf{fn}\ (x_1, \ldots, x_m)\ \Rightarrow\ e_1$	$\Longrightarrow$	$\tau[e] = (\alpha[x_1], \ldots, \alpha[x_m]) \to \tau[e_1]$
App:	$e \equiv e_1\ e_2$	$\Longrightarrow$	$\tau[e_1] = \tau[e_2] \to \tau[e]$
Letrec:	$e \equiv \textbf{letrec}\ x_1 = e_1; \ldots; x_m = e_m\ \textbf{in}\ e_0$	$\Longrightarrow$	$\alpha[x_1] = \tau[e_1] \ldots$ $\alpha[x_m] = \tau[e_m]$ $\tau[e] = \tau[e_0]$

Bemerkung:

- Die möglichen Typ-Zuordnungen an Variablen und Programm-Ausdrücke erhalten wir als Lösung eines Gleichungssystems über Typ-Termen :-)
- Das Lösen von Systemen von Term-Gleichungen nennt man auch Unifikation :-)

Bemerkung:

- Die möglichen Typ-Zuordnungen an Variablen und Programm-Ausdrücke erhalten wir als Lösung eines Gleichungssystems über Typ-Termen :-)
- Das Lösen von Systemen von Term-Gleichungen nennt man auch Unifikation :-)

Beispiel:

$$g(z, f(x)) = g(f(x), f(a))$$

Eine Lösung dieser Gleichung ist die Substitution $\{x \mapsto a, z \mapsto f(a)\}$

In dem Fall ist das offenbar die einzige :-)

Satz:

Jedes System von Term-Gleichungen:

$$s_i = t_i \qquad i = 1, \ldots, m$$

hat entweder keine Lösung oder eine allgemeinste Lösung.

Satz:

Jedes System von Term-Gleichungen:

$$s_i = t_i \qquad i = 1, \ldots, m$$

hat entweder keine Lösung oder eine allgemeinste Lösung.

Eine allgemeinste Lösung ist eine Substitution σ mit den Eigenschaften:

- σ ist eine Lösung, d.h. $\sigma(s_i) = \sigma(t_i)$ für alle i.
- σ ist allgemeinst, d.h. für jede andere Lösung τ gilt: $\tau = \tau' \circ \sigma$ für eine Substitution τ' :-)

Beispiele:

(1) $f(a) = g(x)$ — hat keine Lösung :-)

(2) $x = f(x)$ — hat ebenfalls keine Lösung ;-)

(3) $f(x) = f(a)$ — hat genau eine Lösung:-)

(4) $f(x) = f(g(y))$ — hat unendlich viele Lösungen :-)

(5) $x_0 = f(x_1, x_1), \ldots, x_{n-1} = f(x_n, x_n)$ —

hat mindestens exponentiell große Lösungen !!!

Bemerkungen:

- Es gibt genau eine Lösung, falls die allgemeinste Lösung keine Variablen enthält, d.h. ground ist :-)
- Gibt es zwei verschiedene Lösungen, dann bereits unendlich viele ;-)
- Achtung: Es kann mehrere allgemeinste Lösungen geben !!!

  Beispiel: $x = y$

  Allgemeinste Lösungen sind : $\{x \mapsto y\}$ oder $\{y \mapsto x\}$

  Diese sind allerdings nicht sehr verschieden :-)
- Eine allgemeinste Lösung kann immer idempotent gewählt werden, d.h. $\sigma = \sigma \circ \sigma$.

  Beispiel: $x = x \qquad y = y$

  Nicht idempotente Lösung: $\{x \mapsto y, y \mapsto x\}$

  Idempotente Lösung: $\{x \mapsto x, y \mapsto y\}$

Berechnung einer allgemeinsten Lösung:

$$
\begin{array}{lll}
\textbf{fun}\ \ \mathsf{occurs}\,(x,t) & = & \textbf{case}\ \ t \\
& & \quad\textbf{of}\ \ x \quad\Rightarrow\quad \mathsf{true} \\
& & \quad\ \mid\ \ f(t_1,\ldots,t_k) \quad\Rightarrow\quad \mathsf{occurs}\,(x,t_1) \vee \ldots \vee \mathsf{occurs}\,(x,t_k) \\
& & \quad\ \mid\ \ \_ \quad\Rightarrow\quad \mathsf{false} \\
\textbf{fun}\ \ \mathsf{unify}\,(s,t)\,\theta & = & \textbf{if}\ \ \theta\,s \equiv \theta\,t\ \ \textbf{then}\ \ \theta \\
& & \textbf{else}\ \ \textbf{case}\ \ (\theta\,s, \theta\,t) \\
& & \quad\textbf{of}\ \ (x,t) \ \Rightarrow\ \textbf{if}\ \ \mathsf{occurs}\,(x,t)\ \ \textbf{then}\ \ \mathsf{Fail} \\
& & \qquad\qquad\qquad\quad \textbf{else}\ \ \{x \mapsto t\} \circ \theta \\
& & \quad\ \mid\ \ (t,x) \ \Rightarrow\ \textbf{if}\ \ \mathsf{occurs}\,(x,t)\ \ \textbf{then}\ \ \mathsf{Fail} \\
& & \qquad\qquad\qquad\quad \textbf{else}\ \ \{x \mapsto t\} \circ \theta \\
& & \quad\ \mid\ \ (a,a) \ \Rightarrow\ \theta \\
& & \quad\ \mid\ \ (f(s_1,\ldots,s_k), f(t_1,\ldots,t_k)) \ \Rightarrow\ \mathsf{unifyList}\,[(s_1,t_1),\ldots,(s_k,t_k)]\,\theta \\
& & \quad\ \mid\ \ \_ \ \Rightarrow\ \mathsf{Fail}
\end{array}
$$

$$
\begin{array}{l}
\quad\quad\quad\ldots \\
\textbf{and}\ \ \mathsf{unifyList}\ \mathsf{list}\ \theta\ \ =\ \ \textbf{case}\ \ \mathsf{list} \\
\quad\quad\quad\quad \textbf{of}\ \ [\,]\ \ \longrightarrow\ \ \theta \\
\quad\quad\quad\quad\ \mid\ \ ((s,t) :: \mathsf{rest})\ \ \Rightarrow\ \ \textbf{let val}\ \ \theta = \mathsf{unify}\ (s,t)\ \theta \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \ \textbf{in}\ \ \textbf{if}\ \ \theta = \mathsf{Fail}\ \ \textbf{then}\ \ \mathsf{Fail} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \ \textbf{else}\ \ \mathsf{unifyList}\ \mathsf{rest}\ \theta \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \ \textbf{end}
\end{array}
$$

$\dots$

$$
\begin{array}{l}
\textbf{and}\;\; \mathsf{unifyList}\;\mathsf{list}\;\theta \;\; = \;\; \textbf{case}\;\; \mathsf{list} \\
\qquad\qquad\quad \textbf{of}\;\; [\,] \;\to\; \theta \\
\qquad\qquad\quad \mid\;\; ((s,t) :: \mathsf{rest}) \;\Rightarrow\; \textbf{let val}\;\; \theta = \mathsf{unify}\;(s,t)\;\theta \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\;\; \textbf{in}\;\; \textbf{if}\;\; \theta = \mathsf{Fail}\;\; \textbf{then}\;\; \mathsf{Fail} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\; \textbf{else}\;\; \mathsf{unifyList}\;\mathsf{rest}\;\theta \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\;\; \textbf{end}
\end{array}
$$

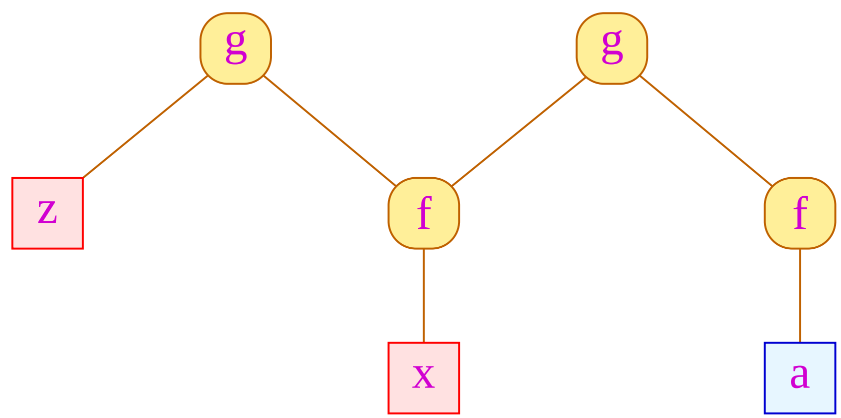
Diskussion:

- Der Algorithmus startet mit $\mathsf{unifyList}\;[(s_1, t_1), \dots, (s_m, t_m)]\;\{\,\}$ ...
- Der Algorithmus liefert sogar eine idempotente allgemeinste Lösung :-)
- Leider hat er möglicherweise exponentielle Laufzeit :-(
- Lässt sich das verbessern ???

Idee:

- Wir repräsentieren die Terme der Gleichungen als Graphen.
- Dabei identifizieren wir bereits isomorphe Teilterme ;-)
- ...

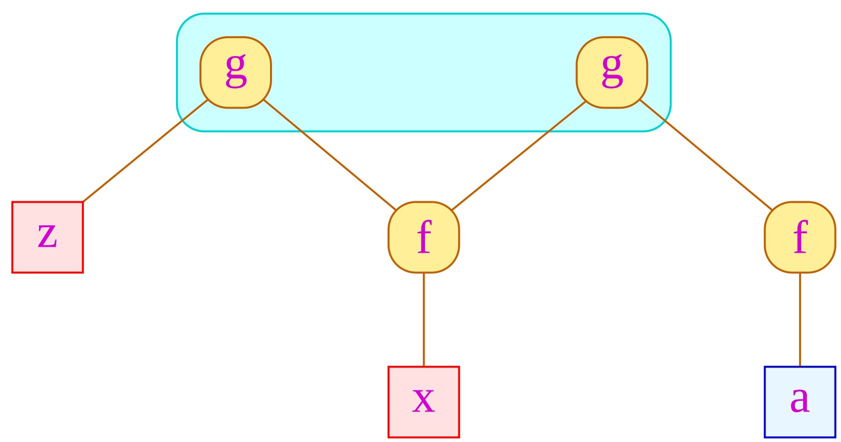
... im Beispiel: $g(z, f(x)) = g(f(x), f(a))$



Idee:

- Wir repräsentieren die Terme der Gleichungen als Graphen.
- Dabei identifizieren wir bereits isomorphe Teilterme ;-)
- ...

... im Beispiel: $g(z, f(x)) \; = \; g(f(x), f(a))$

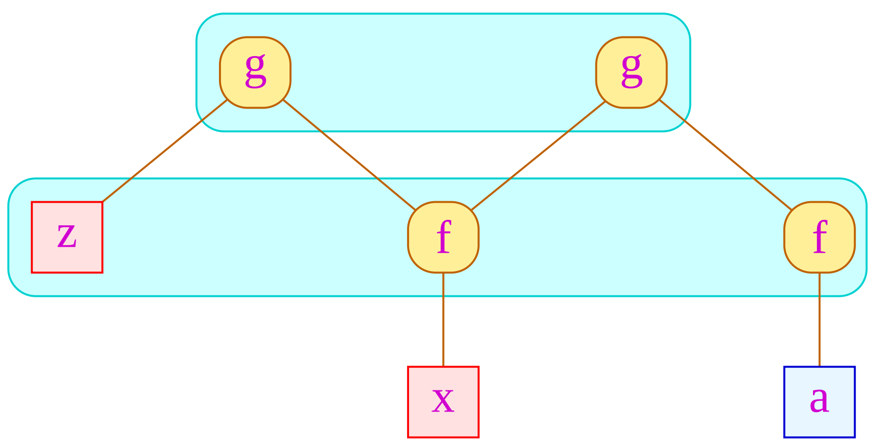


Idee:

- Wir repräsentieren die Terme der Gleichungen als Graphen.
- Dabei identifizieren wir bereits isomorphe Teilterme ;-)
- ...

... im Beispiel:

$$g(z, f(x)) \;=\; g(f(x), f(a))$$



Idee:

- Wir repräsentieren die Terme der Gleichungen als Graphen.
- Dabei identifizieren wir bereits isomorphe Teilterme ;-)
- ...

... im Beispiel:

$$g(z, f(x)) \;=\; g(f(x), f(a))$$