Type Inference

- Why study?
  - increasingly popular language feature
    (but C++ auto and Java var are weak examples;
    admittedly, inference with sub-typing is hard)
  - a canonical example of a 'static analysis'
    ↳ what can we learn/know about a program without running it (i.e., at compile time)

- watch for the relationship between type systems ("what" - specification) and type inference ("how" - implementation)

Example: List Map

- uScheme

```
(val map (lambda (f xs)
              (foldr (lambda (x ys)
                         (cons (f x) ys))
                     nil
                     xs)))
```

- Typed uScheme

```
(val map (type-lambda ['a 'b]
            (lambda ([f : ('a -> 'b)] [xs : ('a list)])
              ((@ foldr 'a ('b list))
                   (lambda ([x : 'a] [ys : ('b list)])
                     ((@ cons 'b) (f x) ys))
                   (@ nil 'b)
                   xs))))
```

Example: List Map

- uScheme

```
(val map (lambda (f xs)
              (foldr (lambda (x ys)
                            (cons (f x) ys))
                     nil
                     xs)))
```

+ programmer didn't need to write much

− no protection / compile time errors if we use the wrong kind of list with the wrong kind of function

- Typed uScheme

```
(val map (type-lambda ['a 'b]
           (lambda ([f : ('a -> 'b)] [xs : ('a list)])
             ((@ foldr 'a ('b list))
                   (lambda ([x : 'a] [ys : ('b list)])
                     ((@ cons 'b) (f x) ys))
                   (@ nil 'b)
                   xs))))
```

+ type safety
  no runtime type errors, instead detected at compile time

− programmer needs to write down type annotations

Key Ideas

- For each unknown type in expression, introduce a fresh (unification) type variable
  "new"
- Enforce (generate + solve) equality constraints
- Introduce type-lambda at let/val defs
- Introduce @ at variable uses

not around arbitrary expressions

Question: Is every Typed uScheme program expressible in nano-ML?

Question: Is every Typed uScheme program expressible in nano-ML?

Answer: No ☹

Full type inference for polymorphism (ie, with no type annotations at all) is undecidable.

Today's solution: Function parameters can only have monomorphic types.

and not polymorphic (∀) types

(but let- and val- bound variables can have polymorphic types)

⇒ Consequence: Polymorphic functions are not first-class.

But, ongoing research in PL to develop partial type inference for first-class polymorphism.

↳ minimal and/or intuitive type annotations

Monotypes + Polytypes

$$\left.\begin{aligned}\tau ::= &\ \text{int} \mid \text{bool} \\ &\mid \tau_1 \times \cdots \times \tau_n \to \tau \\ &\mid \tau \text{ list} \\ &\mid \alpha \mid a\end{aligned}\right\} \text{monotypes}$$

a → unification type var (guesses)

α → quantification type var ("forall type")

$$\sigma ::= \forall \underbrace{\alpha_1, \ldots, \alpha_n}_{\text{quantification type vars, may be empty}} . \tau$$

polytypes

type schemes

$$\Gamma ::= \{ x \mapsto \sigma, \ldots \}$$

Type System (Specification; what type checks, ignoring how)

$\langle \Gamma, d \rangle \to \Gamma'$ — elaboration (type check a defn)

invariant: both Γ and Γ'
map vars to type schemes
with no free quantification type vars
and with no unification vars

$\{\ldots, x \mapsto \underbrace{\forall \alpha_1, \ldots, \alpha_n . \tau}, \ldots$

→ any α in τ is one of $\alpha_1, \ldots, \alpha_n$
→ no a in τ
→ $\alpha_1, \ldots, \alpha_n$ may be empty

intuition: at the top-level,
everything is known

Type System (Specification; what type checks, ignoring how)

$\Gamma \vdash e : \tau$ — type check an expression

invariant: Γ maps vars to type schemes
with no free quantification vars
but with unification vars

$\{ x \mapsto \forall \alpha_1, \ldots, \alpha_n . \tau_x$

val-bound vars,
any α in τ_x is one of $\alpha_1, \ldots, \alpha_n$
no α in τ_x

$y \mapsto \forall . \tau_y$

↳ no quantification type variables

lambda-bound vars
must have monotypes
no α in τ_y
maybe α in τ_y

$z \mapsto \forall \alpha_1, \ldots, \alpha_n . \tau_z$

let-bound vars,
maybe polymorphic
any α in τ_z is one of $\alpha_1, \ldots, \alpha_n$
maybe α in τ_z

Type System

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash n : \text{int}}$$

$$\frac{\Gamma \vdash e_b : \text{bool} \quad \Gamma \vdash e_t : \tau \quad \Gamma \vdash e_f : \tau}{\Gamma \vdash (\text{if } e_b\ e_t\ e_f) : \tau}$$

$$\frac{\Gamma(x) = \forall \alpha_1, \ldots, \alpha_n . \tau}{\Gamma \vdash x : \tau[\alpha_1 \mapsto \tau_1, \ldots, \alpha_n \mapsto \tau_n]}$$

$$\frac{\Gamma\{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\} \vdash e : \tau}{\Gamma \vdash (\text{lambda } (x_1 \ldots x_n)\ e) : \tau_1 \times \cdots \times \tau_n \to \tau}$$

$$\frac{\Gamma \vdash e_f : \tau_1 \times \cdots \times \tau_n \to \tau \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (e_f\ e_1 \ldots e_n) : \tau}$$

$\boxed{\langle \Gamma, d \rangle \to \Gamma'}$

$$\frac{\Gamma \vdash e : \tau \quad \sigma = \text{gen}(\tau, \text{futv}(\Gamma))}{\langle \Gamma, (\text{val } x\ e) \rangle \to \Gamma\{x \mapsto \sigma\}}$$

$\text{gen}(\tau, A)$
$= \forall \alpha_1, \ldots, \alpha_n . \tau[a_1 \mapsto \alpha_1, \ldots, a_n \mapsto \alpha_n]$

where $\{a_1, \ldots, a_n\} = \text{futv}(\tau) - A$

intuition: turn unification ty vars into quantification ty vars

Type System (Specification)

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{}{\Gamma \vdash n : \text{int}} \qquad \frac{\Gamma \vdash e_b : \text{bool} \quad \Gamma \vdash e_t : \tau \quad \Gamma \vdash e_f : \tau}{\Gamma \vdash (\text{if } e_b\ e_t\ e_f) : \tau}$$

picked the right types (out of thin air) that makes expression type check

"magic"

$$\frac{\Gamma\{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\} \vdash e : \tau}{\Gamma \vdash (\text{lambda } (x_1 \ldots x_n)\ e) : \tau_1 \times \cdots \times \tau_n \to \tau}$$

$$\frac{\Gamma(x) = \forall \alpha_1, \ldots, \alpha_n . \tau}{\Gamma \vdash x : \tau[\alpha_1 \mapsto \tau_1, \ldots, \alpha_n \mapsto \tau_n]}$$

"magic"

picked the right types (out of thin air) that makes whole expression (that contains this variable) type check

$$\frac{\Gamma \vdash e_f : \tau_1 \times \cdots \times \tau_n \to \tau \quad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash (e_f\ e_1 \ldots e_n) : \tau}$$

$$\boxed{\langle \Gamma, d \rangle \to \Gamma'}$$

$$\frac{\Gamma \vdash e : \tau \qquad \sigma = gen(\tau, futv(\Gamma))}{\langle \Gamma, (\text{val } x\ e) \rangle \to \Gamma\{x \mapsto \sigma\}}$$

$gen(\tau, A)$

$= \forall \alpha_1, \ldots, \alpha_n . \tau[a_1 \mapsto \alpha_1, \ldots, a_n \mapsto \alpha_n]$

where $\{a_1, \ldots, a_n\} = futv(\tau) - A$

intuition: turn unification ty vars into quantification ty vars

Type Inference (Implementation; how to find types to type check)

Constraints

$C ::= \top$ trivial, always true constraint

$\mid \tau_1 \sim \tau_2$ types must be made equal
- no free quantification type vars
- may have unification type vars
- make equal by substituting for unification type vars

$\mid C \wedge C$ conjunction, both must be (consistently) made true

Type Inference (Implementation; how to find types to type check)

judgements: $C, \Gamma \vdash e : \tau$

inputs to alg.

outputs from alg.

in the interpreter

typeof: exp * type-env → ty * con

$e \quad \Gamma \quad \tau \quad C$

C solved by Θ

input

output

in the interpreter

solve: con → subst

$C \quad \Theta$

Type Inference (Implementation; how to find types to type check)

judgements: $C, \Gamma \vdash e : \tau$

C solved by Θ

Theorem: If $C, \Gamma \vdash e : \tau$

and C solved by Θ } implementation

meets

then $\Theta(\Gamma) \vdash e : \Theta(\tau)$ } Specification

Intuition: Θ provides the guessed τ's

where the "magic" happens

Type Inference

$$\boxed{C, \Gamma \vdash e : \tau}$$

$$\frac{}{\ ,\Gamma \vdash n :}$$

$$\frac{C_b, \Gamma \vdash e_b : \tau_b \quad C_t, \Gamma \vdash e_t : \tau_t \quad C_f, \Gamma \vdash e_f : \tau_f}{\ ,\Gamma \vdash (\text{if } e_b \ e_t \ e_f) :}$$

$$\frac{\Gamma(x) = \forall \alpha_1, \ldots, \alpha_n . \tau}{\ ,\Gamma \vdash x :}$$

$$\frac{\Gamma\{x_1 \mapsto \forall . \alpha_1, \ldots, x_n \mapsto \forall . \alpha_n\} \vdash e : \tau}{\ ,\Gamma \vdash (\text{lambda } (x_1 \ldots x_n) \ e) :}$$

$$\frac{C_f, \Gamma \vdash e_f : \tau_f \quad C_i, \Gamma \vdash e_i : \tau_i}{\ ,\Gamma \vdash (e_f \ e_1 \ldots e_n) :}$$

$$\boxed{\langle \Gamma, d \rangle \to \Gamma'}$$

$$\frac{C, \Gamma \vdash e : \tau \quad C \text{ solved by } \theta \quad \sigma = gen(\theta(\tau), \phi)}{\langle \Gamma, (\text{val } x \ e) \rangle \to \Gamma\{x \mapsto \sigma\}}$$

Type Inference

$$\boxed{C, \Gamma \vdash e : \tau}$$

$$\frac{}{T, \Gamma \vdash n : int}$$

$$\frac{C_b, \Gamma \vdash e_b : \tau_b \quad C_t, \Gamma \vdash e_t : \tau_t \quad C_f, \Gamma \vdash e_f : \tau_f}{C_b \land C_t \land C_f \land \tau_b \sim bool \land \tau_t \sim \tau_e, \Gamma \vdash (\text{if } e_b \; e_t \; e_f) : \tau_t}$$

$$\frac{\Gamma(x) = \forall \alpha_1, \ldots, \alpha_n . \tau \quad a_1, \ldots, a_n \text{ fresh}}{T, \Gamma \vdash x : \tau[\alpha_1 \mapsto a_1, \ldots, \alpha_n \mapsto a_n]}$$

$$\frac{a_1, \ldots, a_n \text{ fresh} \qquad C, \Gamma\{x_1 \mapsto \forall . a_1, \ldots, x_n \mapsto \forall . a_n\} \vdash e : \tau}{C, \Gamma \vdash (\text{lambda } (x_1 \ldots x_n) \; e) : a_1 \times \cdots \times a_n \to \tau}$$

$$\frac{C_f, \Gamma \vdash e_f : \tau_f \quad C_i, \Gamma \vdash e_i : \tau_i \quad a_r \text{ fresh}}{C_f \land C_1 \land \cdots \land C_n \land \tau_f \sim \tau_1 \times \cdots \times \tau_n \to a_r, \Gamma \vdash (e_f \; e_1 \ldots e_n) : a_r}$$

$$\boxed{\langle \Gamma, d \rangle \to \Gamma'}$$

$$\frac{C, \Gamma \vdash e : \tau \quad C \text{ solved by } \theta \quad \sigma = gen(\theta(\tau), \phi)}{\langle \Gamma, (\text{val } x \; e) \rangle \to \Gamma\{x \mapsto \sigma\}}$$

Type Inference

$$\boxed{C \text{ solved by } \theta}$$

$$\frac{}{\top \text{ solved by}} \qquad \frac{}{int \sim int \text{ solved by}}$$

$$\frac{}{\tau \text{ list} \sim \tau' \text{ list solved by}} \qquad \frac{}{\tau_{a1} \to \tau_{r1} \sim \tau_{a2} \to \tau_{r2} \text{ solved by}}$$

$$\frac{C_1 \text{ solved by } \theta_1 \qquad \theta_1(C_2) \text{ solved by } \theta_2}{C_1 \land C_2 \text{ solved by } \theta_2 \circ \theta_1} \qquad \frac{}{\alpha \sim \alpha \text{ solved by}}$$

$$\frac{}{\alpha \sim \tau \text{ solved by}} \qquad \frac{}{\tau \sim \alpha \text{ solved by}}$$

Type Inference

$\boxed{C \text{ solved by } \Theta}$ Unification like in Prolog

$$\frac{}{\top \text{ solved by } \{\}} \qquad \frac{}{int \sim int \text{ solved by } \{\}}$$

$$\frac{\tau \sim \tau' \text{ solved by } \Theta}{\tau\ list \sim \tau'\ list \text{ solved by } \Theta} \qquad \frac{\tau_{a1} \sim \tau_{a2} \wedge \tau_{r1} \sim \tau_{r2} \text{ solved by } \Theta}{\tau_{a1} \to \tau_{r1} \sim \tau_{a2} \to \tau_{r2} \text{ solved by } \Theta}$$

info. from solving C_1 "pushed" into C_2 before solving

$$\frac{C_1 \text{ solved by } \Theta_1 \qquad \Theta_1(C_2) \text{ solved by } \Theta_2}{C_1 \wedge C_2 \text{ solved by } \Theta_2 \circ \Theta_1} \qquad \frac{}{a \sim a \text{ solved by } \{\}}$$

combine substitutions b/c Θ_2 doesn't include Θ_1

$$\frac{a \notin \underline{futv}(\tau)}{a \sim \tau \text{ solved by } \{a \mapsto \tau\}} \qquad \frac{a \notin \underline{futv}(\tau)}{\tau \sim a \text{ solved by } \{a \mapsto \tau\}}$$

Last two rules: $a \sim (a\ list)\ list$ cannot be solved
no substitution makes both sides equal

} Same occurs check as Prolog

Example

Perform type inference on

```
(val length (lambda (xs)
              (foldr (lambda (x l) (+ 1 l))
                     0
                     xs)))
```

Expect to find:

$$\forall \alpha.\ \alpha\ \text{list} \rightarrow \text{int}$$

$$\dfrac{a_x, a_l \text{ fresh} \qquad \dfrac{\dfrac{\Gamma_2(\texttt{+}) = \forall\,.\,\mathsf{int}\times\mathsf{int}\to\mathsf{int}}{\underline{\quad}, \Gamma_2 \vdash \texttt{+} : \underline{\qquad\qquad}}\,(\textsc{Var}) \qquad \dfrac{}{\underline{\quad}, \Gamma_1 \vdash \texttt{1} : \underline{\quad}}\,(\textsc{Int}) \qquad \dfrac{\Gamma_2(\texttt{l}) = \forall\,.\,a_l}{\underline{\quad}, \Gamma_2 \vdash \texttt{l} : \underline{\quad}}\,(\textsc{Var}) \qquad a_{r2} \text{ fresh}}{\underline{\qquad\qquad\qquad\qquad\qquad}, \Gamma_1 \vdash \texttt{(+ 1 l)} : \underline{\quad}}\,(\textsc{App})}{\underline{\qquad\qquad\qquad\qquad\qquad}, \Gamma_1 \vdash \texttt{(lambda (x l) (+ 1 l))} : \underline{\qquad\qquad}}\,(\textsc{Lam})$$

$$\dfrac{a_{xs} \text{ fresh} \qquad \dfrac{\dfrac{\begin{array}{c}\Gamma_1(\texttt{foldr}) = \forall\alpha,\beta\,.\,(\alpha\times\beta\to\beta)\times\beta\times\alpha\ \mathsf{list}\to\beta \\ a, b \text{ fresh}\end{array}}{\underline{\quad}, \Gamma_1 \vdash \texttt{foldr} : \underline{\qquad\qquad\qquad\qquad}}\,(\textsc{Var}) \qquad \vdots \qquad \dfrac{}{\underline{\quad}, \Gamma_1 \vdash \texttt{0} : \underline{\quad}}\,(\textsc{Int}) \qquad \dfrac{\Gamma_1(\texttt{xs}) = \forall\,.\,a_{xs}}{\underline{\quad}, \Gamma_1 \vdash \texttt{x} : \underline{\quad}}\,(\textsc{Var}) \qquad a_{r1} \text{ fresh}}{\underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad},\ \Gamma_1 \vdash \texttt{(foldr (lambda (x l) (+ 1 l)) 0 xs)} : \underline{\quad}}\,(\textsc{App})}{\underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad},\ \Gamma_0 \vdash \texttt{(lambda (xs) (foldr (lambda (x l) (+ 1 l)) 0 xs))} : \underline{\qquad\quad}}\,(\textsc{Lam})$$

$$\dfrac{{\color{red}C}, \Gamma_0 \vdash \texttt{(lambda (xs) (foldr (lambda (x l) (+ 1 l)) 0 xs))} : \underline{\qquad\quad} \qquad {\color{red}C} \text{ solved by } {\color{red}\theta} \qquad \texttt{generalize}({\color{red}\theta(\underline{\qquad\quad})}, \emptyset) = \underline{\qquad\qquad}}{\langle \Gamma_0, \texttt{(val length (lambda (xs) (foldr (lambda (x l) (+ 1 l)) 0 xs)))} \rangle \to \Gamma_0\{\texttt{length} \mapsto \underline{\qquad\qquad}\}}\,(\textsc{Val})$$

$$\begin{aligned}
\Gamma_0 &= \{\texttt{+} \mapsto \forall\,.\,\mathsf{int}\times\mathsf{int}\to\mathsf{int}, \texttt{foldr} \mapsto \forall\alpha,\beta\,.\,(\alpha\times\beta\to\beta)\times\beta\times\alpha\ \mathsf{list}\to\beta, \ldots\} \\
\Gamma_1 &= \underline{\qquad\qquad\qquad} \\
\Gamma_2 &= \underline{\qquad\qquad\qquad\qquad\quad} \\
C &= \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \\
\theta &= \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \\
{\color{red}\theta(\underline{\qquad\quad})} &= \underline{\qquad\quad}
\end{aligned}$$

$$\dfrac{a_x, a_l \text{ fresh} \qquad \dfrac{\dfrac{\Gamma_2(\texttt{+}) = \forall\, .\, \mathsf{int}\times\mathsf{int}\to\mathsf{int}}{\underline{\mathrm{T}}, \Gamma_2 \vdash \texttt{+} : \underline{\mathsf{int}\times\mathsf{int}\to\mathsf{int}}}\,(\textsc{Var}) \quad \dfrac{}{\underline{\mathrm{T}}, \Gamma_1 \vdash \texttt{1} : \underline{\mathsf{int}}}\,(\textsc{Int}) \quad \dfrac{\Gamma_2(\texttt{l}) = \forall\, .\, a_l}{\underline{\mathrm{T}}, \Gamma_2 \vdash \texttt{l} : \underline{a_l}}\,(\textsc{Var}) \quad a_{r2} \text{ fresh}}{\underline{\mathrm{T}\wedge\mathrm{T}\wedge\mathrm{T}\wedge\mathsf{int}\times\mathsf{int}\to\mathsf{int}\sim\mathsf{int}\times a_l\to a_{r2}}, \Gamma_1 \vdash \texttt{(+ 1 l)} : \underline{a_{r2}}}\,(\textsc{App})}{\underline{\mathrm{T}\wedge\mathrm{T}\wedge\mathrm{T}\wedge\mathsf{int}\times\mathsf{int}\to\mathsf{int}\sim\mathsf{int}\times a_l\to a_{r2}}, \Gamma_1 \vdash \texttt{(lambda (x l) (+ 1 l))} : \underline{a_x\times a_l\to a_{r2}}}\,(\textsc{Lam})$$

$$\dfrac{a_{xs} \text{ fresh} \qquad \dfrac{\dfrac{\begin{array}{c}\Gamma_1(\texttt{foldr}) = \forall\alpha,\beta\, .\, (\alpha\times\beta\to\beta)\times\beta\times\alpha\ \mathsf{list}\to\beta \\ a, b \text{ fresh}\end{array}}{\underline{T}, \Gamma_1 \vdash \texttt{foldr} : \underline{(a\times b\to b)\times b\times a\ \mathsf{list}\to b}}\,(\textsc{Var}) \quad \vdots \quad \dfrac{}{\underline{\mathrm{T}}, \Gamma_1 \vdash \texttt{0} : \underline{\mathsf{int}}}\,(\textsc{Int}) \quad \dfrac{\Gamma_1(\texttt{xs}) = \forall\, .\, a_{xs}}{\underline{\mathrm{T}}, \Gamma_1 \vdash \texttt{x} : \underline{a_{xs}}}\,(\textsc{Var}) \quad a_{r1} \text{ fresh}}{\begin{array}{l}\underline{\mathrm{T}\wedge(\mathrm{T}\wedge\mathrm{T}\wedge\mathrm{T}\wedge\mathsf{int}\times\mathsf{int}\to\mathsf{int}\sim\mathsf{int}\times a_l\to a_{r2})\wedge\mathrm{T}\wedge\mathrm{T}\wedge(a\times b\to b)\times b\times a\ \mathsf{list}\to b\sim(a_x\times a_l\to a_{r2})\times\mathsf{int}\times a_{xs}\to a_{r1}}, \\ \Gamma_1 \vdash \texttt{(foldr (lambda (x l) (+ 1 l)) 0 xs)} : \underline{a_{r1}}\end{array}}\,(\textsc{App})}{\begin{array}{l}\underline{\mathrm{T}\wedge(\mathrm{T}\wedge\mathrm{T}\wedge\mathrm{T}\wedge\mathsf{int}\times\mathsf{int}\to\mathsf{int}\sim\mathsf{int}\times a_l\to a_{r2})\wedge\mathrm{T}\wedge\mathrm{T}\wedge(a\times b\to b)\times b\times a\ \mathsf{list}\to b\sim(a_x\times a_l\to a_{r2})\times\mathsf{int}\times a_{xs}\to a_{r1}}, \\ \Gamma_0 \vdash \texttt{(lambda (xs) (foldr (lambda (x l) (+ 1 l)) 0 xs))} : \underline{a_{xs}\to a_{r1}}\end{array}}\,(\textsc{Lam})$$

$$\dfrac{C, \Gamma_0 \vdash \texttt{(lambda (xs) (foldr (lambda (x l) (+ 1 l)) 0 xs))} : \underline{a_{xs}\to a_{r1}} \qquad C \text{ solved by } \theta \qquad \texttt{generalize}(\theta(\underline{a_{xs}\to a_{r1}}), \emptyset) = \underline{\forall\alpha\, .\, \alpha\ \mathsf{list}\to\mathsf{int}}}{\langle\Gamma_0, \texttt{(val length (lambda (xs) (foldr (lambda (x l) (+ 1 l)) 0 xs)))}\rangle \to \Gamma_0\{\texttt{length}\mapsto\underline{\forall\alpha\, .\, \alpha\ \mathsf{list}\to\mathsf{int}}\}}\,(\textsc{Val})$$

$$\begin{array}{rcl}
\Gamma_0 & = & \{\texttt{+}\mapsto\forall\, .\, \mathsf{int}\times\mathsf{int}\to\mathsf{int}, \texttt{foldr}\mapsto\forall\alpha,\beta\, .\, (\alpha\times\beta\to\beta)\times\beta\times\alpha\ \mathsf{list}\to\beta, \ldots\} \\
\Gamma_1 & = & \underline{\Gamma_0\{\texttt{xs}\mapsto\forall\, .\, a_{xs}\}} \\
\Gamma_2 & = & \underline{\Gamma_1\{\texttt{x}\mapsto\forall\, .\, a_x, \texttt{l}\mapsto\forall\, .\, a_l\}} \\
C & = & \underline{\mathrm{T}\wedge(\mathrm{T}\wedge\mathrm{T}\wedge\mathrm{T}\wedge\mathsf{int}\times\mathsf{int}\to\mathsf{int}\sim\mathsf{int}\times a_l\to a_{r2})\wedge\mathrm{T}\wedge\mathrm{T}\wedge(a\times b\to b)\times b\times a\ \mathsf{list}\to b\sim(a_x\times a_l\to a_{r2})\times\mathsf{int}\times a_{xs}\to a_{r1}} \\
\theta & = & \underline{\{a_l\mapsto\mathsf{int}, a_{r2}\mapsto\mathsf{int}, a\mapsto a_x, b\mapsto\mathsf{int}, a_{xs}\mapsto a_x\ \mathsf{list}\}} \\
\theta(\underline{a_{xs}\to a_{r1}}) & = & \underline{a_x\ \mathsf{list}\to\mathsf{int}}
\end{array}$$

Type Errors with Type Inference

- How do we report errors when doing type inference?
- Note: $C, \Gamma \vdash e : \tau$ typeof: exp * env → ty * con

  never fails - it always generates some constraints

  ↳ (almost never. unbound vars are an error here)
- Only solving $\tau_1 \sim \tau_2$ constraints can fail
- Simple approach: $\tau_1 \sim_{loc} \tau_2$

  ↑ when generating an equality, annotate w/ source location of expression.