CS440: Programming Languages and Translators

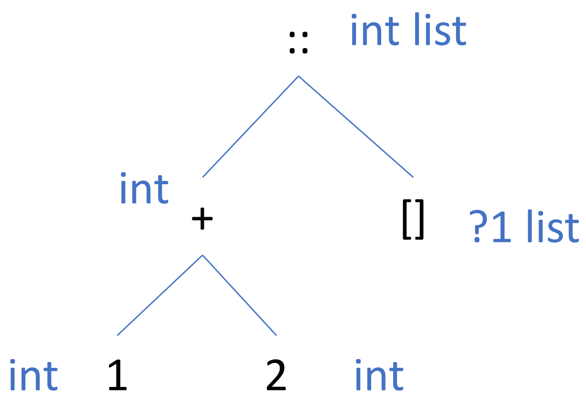
Lecture 16: Type Inference

Spring 2023

Stefan Muller

Type inference: basic idea

- infer(e: expression) : typ =
  - Call infer recursively on subexpressions of e
  - Figure out the type of e from the types of subexpressions
    - Use unification to enforce any constraints on types
    - If we ever don’t know the type of something, make a new unification variable



unify(?1, int)

We need to know the types of variables

```
let x = 1 in
let y = 2 in
x + y
```

A *context* keeps track of the types of variables

- infer(ctx: context, e: expression) : typ =
  - Call infer recursively on subexpressions of e
  - Figure out the type of e from the types of subexpressions
    - Use unification to enforce any constraints on types
    - If we see a variable, look it up in ctx (if not in ctx, it's unbound)
    - If we ever don't know the type of something, make a new unification variable

- How do we represent a context?
  - Map? Association list?

We need to know the types of variables

```
let x = 1 in
```
[x -> int]
```
let y = 2 in
```
[x -> int, y -> int]
```
x + y
```

Context **does not** store the values of variables! We’re not computing anything here!

How do we compute the type of e from the types of subexpressions?

- Depends on what e is.
- Example: e = e1 + e2
- Remember: e1 + e2 has type int if e1, e2 have type int
  - Let t1 = infer(ctx, e1)
  - Let t2 = infer(ctx, e2)
  - Unify(t1, int)
  - Unify(t2, int)
  - (If neither unification failed) return int

We also need to keep track of substitutions

- infer([x -> ?1], x + List.length x)
- infer([x -> ?1], x) = ?1
- infer([x -> ?1], List.length x) = int   unify(?1, ?2 list)
- unify(?1, int)
- unify(int, int)
- return int

- But ?1 can't be int and ?2 list!

Infer should also return a substitution

- infer([x -> ?1], x + List.length x)
- infer([x -> ?1], x) = (?1, [])
- infer([][x -> ?1], List.length x) = (int, [(?1, ?2 list)])

Apply current substitution to the context

- unify([[(?1, ?2 list)]]?1, [[(?1, ?2 list)]]int) = unify(?2 list, int) -> Shape Mismatch

Apply current substitution to the types in unification

Infer should also return a substitution

- infer([x -> ?1], (List.length x)::x)
- infer([x -> ?1], List.length x) = (int, [(?1, ?2 list)])
- infer([[(?1, ?2 list)]][x -> ?1], x)
  = infer([x -> ?2 list], x) = (?2 list, [])
- unify([[]]int list, ?2 list) = [(?2, int)]
- return (int list, [] @ [(?1, ?2 list)] @ [(?2, int)])
  = (int list, [(?1, ?2 list); (?2, int)])

Append all substitutions at the end

A *context* keeps track of the types of variables

- infer(ctx: context, e: expression) : typ * subst =
  - Call infer recursively on subexpressions of e
    - Need to apply previous substitutions to ctx
  - Figure out the type of e from the types of subexpressions
    - Use unification to enforce any constraints on types
    - If we see a variable, look it up in ctx (if not in ctx, it’s unbound)
    - If we ever don’t know the type of something, make a new unification variable

- How do we represent a context?
  - Map? Association list?