CS440: Programming Languages and Translators

Lecture 14: Type Checking and Unification
Spring 2023

Stefan Muller
Type checking isn’t too hard

let rec sum (l: int list) : int =
    match (l: int list) with
    | ([]) : int list) -> (0 : int)
    | (h::t : int list) -> ((h: int) + (sum t: int): int)
Type checking isn’t too hard... even if we only have inputs

let rec sum (l: int list) : int =
  match (l: int list) with
  | ([]: int list) -> (0 : int)
  | (h::t : int list) -> ((h: int) + (sum t: int): int)
Type *inference* is a little harder

```ocaml
let rec sum l =
    match l with
    | [] -> 0
    | h::t -> h + sum t
```
Type *inference* is a little harder

```
let rec sum (l: ?? ) =
  match l with
  | [] -> 0
  | h::t -> h + sum t
```
Type *inference* is a little harder

```ocaml
let rec sum (l: ?? ) = 
match (l: ?? ) with 
| [] -> 0 
| h::t -> h + sum t
```
Type *inference* is a little harder

```ocaml
let rec sum (l: ??? ) = 
  match (l: ??? ) with 
  | [] -> 0 
  | h::t -> h + sum t
```
Type *inference* is a little harder

```ocaml
let rec sum (l: int list) : int =
  match (l: int list) with
  | [] -> (0 : int)
  | h::t -> ((h: int) + (sum (t: int list) : int): int)
```
Unification variables help us keep track of what we still have to figure out

• We’ll use ?1, ?2, ?3, etc.
• Need to fill in the same type everywhere ?1 appears
• NOT the same as type variables ‘a, ‘b, etc., but difference is subtle
When we see something whose type we don’t know, add a unif. var.

```ml
let rec sum (l: ?1) : ?2 =
    match l with
    | [] -> 0
    | h::t -> h + sum t
```
Keep unification variables consistent

let rec sum (l: ?1) : ?2 =
    match (l: ?1) with
    | [] -> 0
    | h::t -> h + sum t
We can refine unification variables when we get more information

let rec sum (l: ?3 list) : ?2 =
  match (l: ?3 list) with
  | [] -> 0
  | h::t -> h + sum t
We can refine unification variables when we get more information

let rec sum (l: ?3 list) : ?2 =
    match (l: ?3 list) with
    | [] -> (0: ?2)
We can refine unification variables when we get more information

let rec sum (l: ?3 list) : int =
   match (l: ?3 list) with
   | [] -> (0: int)
   | h::t -> ((h: ?3) + (sum (t: ?3 list): int): int)
We can refine unification variables when we get more information

let rec sum (l: int list) : int =
    match (l: int list) with
    | [] -> (0: int)
    | h::t -> ((h: int) + (sum (t: int list): int): int)
Unification

- Making types “look like” each other
- e.g., unify(?1, ?3 list)
- e.g., unify(?3, int)
What if you can’t unify?

let rec sum_bad (l: ?1) : ?2 =
    match (l: ?1) with
    | [] -> 0
    | h::t -> h +. sum t
What if you can’t unify?

let rec sum_bad (l: ?3 list) : int =
    match (l: ?3 list) with
    | [] -> (0: int)
    | h::t -> (h +. sum t: int)
What if you can’t unify?

let rec sum_bad (l: ?3 list) : int =
  match (l: ?3 list) with
  | [] -> (0: int)
  | h::t -> ((h: ?3) +. (sum (t: ?3 list) : int): int)
What if you can’t unify?

let rec sum_bad (l: float list) : int =
    match (l: float list) with
    | [] -> (0: int)
    | h::t -> ((h: float) +. (sum (t: float list) : int): int)
What if you can’t unify?

let rec sum_bad (l: float list) : int =
    match (l: float list) with
    | [] -> (0: int)
    | h::t -> ((h: float) +. (sum (t: float list) : int): int)

unify(float, int)

A: Type error
The goal of unification is to produce a substitution $\sigma$

- A mapping from unification variables to types
- e.g., [$?1 \rightarrow ?3\text{ list}, ?2 \rightarrow \text{int}, ?3 \rightarrow \text{int}$]
Unification example

Across
1. SB 104, SB 218E, e.g.
4. A functional language used at Jane Street Capital
6. [], h::t, (x, y), for example

Down
1. Maker of the Spectra 70 computer (abbr.)
2. _____ e with
3. _____ solver, which we hope we don’t need for type inference
4. int _____, the type of “Some 42”
5. Xavier _____, inventor of 4-Across
Unification example

Across
1. SB 104, SB 218E, e.g.
4. A functional language used at Jane Street Capital
5. Xavier ____, inventor of 4

Down
1. Maker of the Spectra 70 computer (abbr.)
2. ____ e with
3. ____ solver, which we hope we don’t need for type inference
4. int ____, the type of “Some 42”
6. [(), ::t, (x, y), for example

Lecture 0
24
Unification example

Across
1. SB 104, SB 218E, e.g.
4. A functional language used at Jane Street Capital
6. [], h::t, (x, y), for example

Down
1. Maker of the Spectra 70 computer (abbr.)
2. ____ e with
3. ____ solver, which we hope we don’t need for type inference
4. int ____ , the type of “Some 42”
5. Xavier ____ , inventor of 4-Across
Unification example

Across
1. SB 104, SB 218E, e.g.
4. A functional language used at Jane Street Capital
6. [], h::t, (x, y), for example

Down
1. Maker of the Spectra 70 computer (abbr.)
2. _____ e with
3. _____ solver, which we hope we don’t need for type inference
4. int _____, the type of “Some 42”
5. Xavier _____, inventor of 4-Across

Substituting types

• We write \([\sigma]\tau\) to mean “\(\tau\) with all of the substitutions in \(\sigma\)”

\[\text{[?1 -> int list, ?2 -> int]}(?1 -> ?2) = \text{int list -> int}\]

• “Simultaneous substitution”: keep substituting until things don’t change
  • \([?1 -> ?3 \text{ list, ?2 -> int, ?3 -> int]}(?1 -> ?2) = \text{int list -> int}\]
We build up a substitution as we unify

```ocaml
defined rec sum l =
    match l with
    | [] -> 0
    | h::t -> h + sum t
```
We build up a substitution as we unify

```ocaml
let rec sum (l: ?1) : ?2 =
  match l with
  | []  -> 0
  | h::t -> h + sum t
```

Lecture 0
We build up a substitution as we unify

```ocaml
let rec sum (l: ?1) : ?2 =
  match (l: ?1) with
  | [] -> 0
  | h::t -> h + sum t
```
We build up a substitution as we unify

```ocaml
let rec sum (l: ?1) : ?2 =
  match (l: ?1) with
  | [] -> 0
  | h::t -> h + sum t
```

?1 -> ?3 list,
We build up a substitution as we unify

let rec sum (l: ?1) : ?2 =
    match (l: ?1) with
    | []   -> (0: ?2)
    | h::t -> h + sum t
We build up a substitution as we unify

let rec sum (l: ?1) : ?2 =
  match (l: ?1) with
  | [] -> (0: ?2)   ?1 -> ?3 list, ?2 -> int
  | h::t -> (h: ?3) + (sum (t: ?3 list) : ?2) : ?2
We build up a substitution as we unify

let rec sum (l: ?1) : ?2 =
    match (l: ?1) with
    | [] -> (0: ?2)
    | h::t -> (h: ?3) + (sum (t: ?3 list) : ?2) : ?2
Q: What if we have leftover unification variables when we’re done?

```ocaml
let rec length (l: ?1) : ?2 =  
  match (l: ?1) with  
  | [] -> (0: ?2)  
  | h::t -> (1: ?4) + (length (t: ?3 list) : ?2) : ?2
```
Q: What if we have leftover unification variables when we’re done?

```
let rec length (l: ?1) : ?2 =
    match (l: ?1) with
    | [] -> (0: ?2)
    | h::t -> (1: ?4) + (length (t: ?3 list) : ?2) : ?2
```

A: They become type variables (but it’s a little complicated)