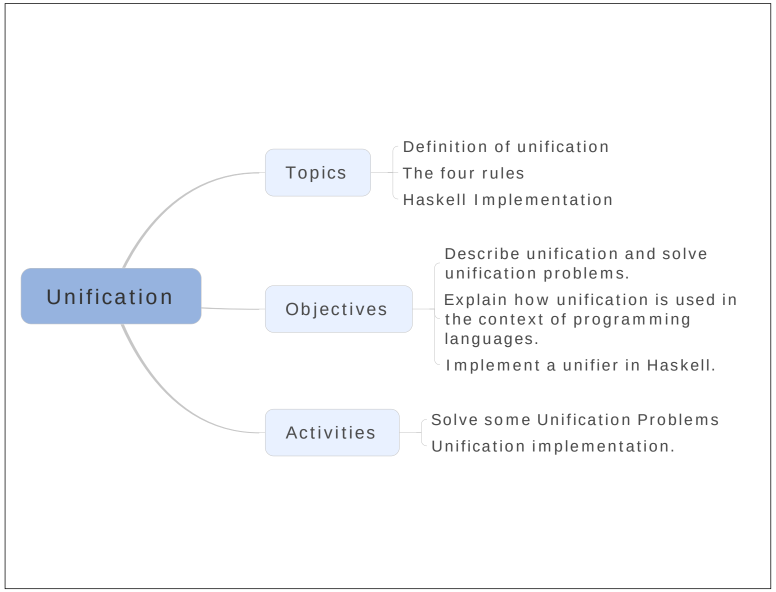
Unification

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Outline



Objectives

You should be able to...

Unification is a third major topic that will appear many times in this course. It is used in languages such as Haskell and Prolog, and also in theoretical discussions.

- Be able to describe the problem of unification.
- Be able to solve a unification problem.
- Be able to implement unification in Haskell.
- Know how to use unification to implement pattern matching.
- Know how to use unification to check types of functions.

The Domain

Terms Have *name* and *arity*

- The name will be in western alphabet
- Arity = "number of arguments" — may be zero
- Examples: `x`, `z`, `f(x,y)`, `x(y,f,z)`

Variables Written using Greek alphabet, may be subscripted

- Represent a target for substitution
- Examples: $\alpha, \beta_{12}, \gamma_7$

Substitutions Mappings from Variables to Terms

- Examples: $\sigma = \{\alpha \mapsto \mathtt{f}(\mathtt{x}, \beta), \beta \mapsto \mathtt{y}\}$
- Substitutions are *applied*: $\sigma(\mathtt{g}(\beta)) \to \mathtt{g}(\mathtt{y})$

Note: arguments to terms may have non-zero arity, or may be variables.

The Problem

- Given terms s and t, try to find a substitution σ such that $\sigma(s) = \sigma(t)$.
- If such a substitution exists, it is said that s and t unify.
- A *unification problem* is a set of equations $S = \{s_1 = t_1, s_2 = t_2, \ldots\}$.
- A unification problem $S = \{x_1 = t_1, x_2 = t_2, \ldots\}$ is in *solved form* if
  - the terms x_i are distinct variables
  - none of them occur in t_i.

Our approach: given a unification problem S, we want to find the most general unifier σ that solves it. We will do this by transforming the equations.

Four Operations

Start with a unification problem $S = \{s_1 = t_1, s_2 = t_2, \ldots\}$ and apply the following transformations as necessary:

Delete A trivial equation $t = t$ can be deleted.

Decompose An equation $f(\overline{t_n}) = f(\overline{u_n})$ can be replaced by the set $\{t_1 = u_1, \ldots, t_n = u_n\}$

Orient An equation $t = x$ can be replaced by $x = t$ if x is a variable and t is not.

Eliminate an equation $x = t$ can be used to substitute all occurrences of x in the remainder of S.

Example

(Stolen from "Term Rewriting and All That")
$\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}$

Example

(Stolen from "Term Rewriting and All That")
$\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}$
We can use the Eliminate method, replace α with $f(x)$ on the right sides of the equations.

Example

(Stolen from "Term Rewriting and All That")
$\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}$
We can use the Eliminate method, replace α with $f(x)$ on the right sides of the equations.
$\{\alpha = f(x),\ g(f(x), f(x)) = g(f(x), \beta)\}$
We can use the Decompose method, and get rid of the g functions.

Example

(Stolen from "Term Rewriting and All That")
$\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}$
We can use the Eliminate method, replace α with $f(x)$ on the right sides of the equations.
$\{\alpha = f(x),\ g(f(x), f(x)) = g(f(x), \beta)\}$
We can use the Decompose method, and get rid of the g functions.
$\{\alpha = f(x),\ f(x) = f(x), f(x) = \beta\}$
We can delete the $f(x) = f(x)$ equation.

Example

(Stolen from "Term Rewriting and All That")
$\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}$
We can use the Eliminate method, replace α with $f(x)$ on the right sides of the equations.
$\{\alpha = f(x),\ g(f(x), f(x)) = g(f(x), \beta)\}$
We can use the Decompose method, and get rid of the g functions.
$\{\alpha = f(x),\ f(x) = f(x), f(x) = \beta\}$
We can delete the $f(x) = f(x)$ equation.
$\{\alpha = f(x),\ f(x) = \beta\}$
Now we can reorient to make the variables show up on the left side.

Example

(Stolen from "Term Rewriting and All That")
$\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}$
We can use the Eliminate method, replace α with $f(x)$ on the right sides of the equations.
$\{\alpha = f(x),\ g(f(x), f(x)) = g(f(x), \beta)\}$
We can use the Decompose method, and get rid of the g functions.
$\{\alpha = f(x),\ f(x) = f(x), f(x) = \beta\}$
We can delete the $f(x) = f(x)$ equation.
$\{\alpha = f(x),\ f(x) = \beta\}$
Now we can reorient to make the variables show up on the left side.
$\{\alpha = f(x),\ \beta = f(x)\}$
Now we are done....
$S = \{\alpha \mapsto f(x),\ \beta \mapsto f(x)\}$

Example — Compatibility

- Your advisor wants you to take CS 440 and some theory class.
- Your mom wants you to take CS 536 and some languages class.
- Can both your advisor and your mom be happy?

This is a problem we can solve using unification:

- Let f be a "schedule function", the first argument is a language class, the second argument is a theory class.
- $s = f(cs440, \beta)$ (where β is a theory class)
- $t = f(\alpha, cs536)$ (where α is a language class)
- Let $\sigma = \{\alpha \mapsto cs440, \quad \beta \mapsto cs536\}$

Example — Types

Type checking is also a form of unification.

```
map :: (a -> b) -> [a] -> [b]
inc :: Int -> Int
foo :: [Int]
```

Will `map(inc)(foo)` work?

$$S = \{(\alpha \Rightarrow \beta) = (\texttt{Int} \Rightarrow \texttt{Int}), \quad \texttt{List}[\alpha] = \texttt{List}[\texttt{Int}]\}$$

Type Checking Solution

$$S = \{(\alpha \Rightarrow \beta) = (\texttt{Int} \Rightarrow \texttt{Int}), \quad \texttt{List}[\alpha] = \texttt{List}[\texttt{Int}]\}$$

- Decompose: $\{\alpha = \texttt{Int}, \quad \beta = \texttt{Int}, \quad \texttt{List}[\alpha] = \texttt{List}[\texttt{Int}]\}$
- Substitute: $\{\alpha = \texttt{Int}, \quad \beta = \texttt{Int}, \quad \texttt{List}[\texttt{Int}] = \texttt{List}[\texttt{Int}]\}$
- Delete: $\{\alpha = \texttt{Int}, \quad \beta = \texttt{Int}\}$

The original type of `map` was $(\alpha \Rightarrow \beta) \Rightarrow \texttt{List}[\alpha] \Rightarrow \texttt{List}[\beta]$
We can use our pattern to get the output type: $S(\texttt{List}[\beta]) \equiv \texttt{List}[\texttt{Int}]$

Example 2 — Types

Here's an example that fails. g

```
map :: (a->b) -> [a] -> [b]
inc : String -> Int
foo : [Int]
```

Will `map(inc)(foo)` work?

$$S = \{(\alpha \Rightarrow \beta) = (\texttt{String} \Rightarrow \texttt{Int}), \quad \texttt{List}[\alpha] = \texttt{List}[\texttt{Int}]\}$$

Type Checking 2 Solution

$$S = \{(\alpha \Rightarrow \beta) = (\texttt{String} \Rightarrow \texttt{Int}), \quad \texttt{List}[\alpha] = \texttt{List}[\texttt{Int}]\}$$

- Decompose: $\{\alpha = \texttt{String}, \quad \beta = \texttt{Int}, \quad \texttt{List}[\alpha] = \texttt{List}[\texttt{Int}]\}$
- Substitute: $\{\alpha = \texttt{string}, \quad \beta = \texttt{Int}, \quad \texttt{List}[\texttt{String}] = \texttt{List}[\texttt{Int}]\}$
- Error: $\texttt{List}[\texttt{string}] \neq \texttt{List}[\texttt{Int}]$!

Problem

Try the Unification Solving Activity

How to make this work in Haskell

To build a unifier, you need:

- a way to represent unification problems... i.e., a type,
- a way to decide which unification step is appropriate,
  - (and a way to tell when we are done)
- and functions to perform the various unification steps.

How should we represent things?

Strategy for Writing the Function

- You need three lists:
  - One is the list of solved-form equations.
  - Two form a queue of elements in progress.
- You need functions to perform the transformations we need.
  - Substitute
  - Deconstruct
  - Reorient (easy)
  - Drop (very easy)
- You may need a flag to indicate completion.

Time to start coding...!