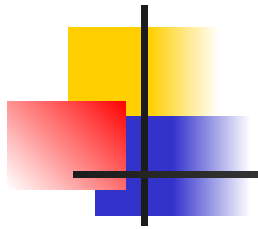


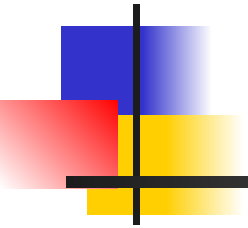
Programming Languages and Compilers (CS 421)

Talia Ringer (they/them)

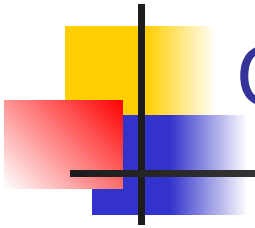
4218 SC, UIUC

https://courses.grainger.illinois.edu/cs421/fa2023/

Based heavily on slides by Elsa Gunter, which were based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

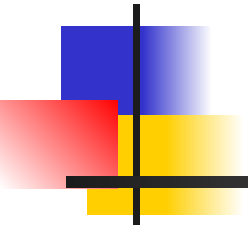


Midterm Post on Piazza

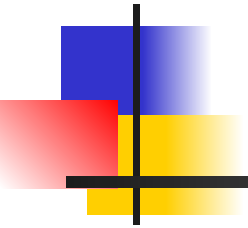


Objectives for Today

- Last week, we covered **type inference**
- There were a number of places where we mentioned **unification**, but we abstracted over how it actually works
- This week, we'll explain **how unification works**



Questions from last week?



Unification

Unification Problem

Given a set of pairs of terms ("equations")

$$\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$$

(the **unification problem**) does there exist a substitution σ (the **unification solution**) of terms for variables such that

$$\sigma(s_i) = \sigma(t_i),$$

for all $i = 1, \ldots, n$?

Unification Problem

Given a set of pairs of terms ("equations")

$$\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$$

(the **unification problem**) does there exist a substitution σ (the **unification solution**) of terms for variables such that

$$\sigma(s_i) = \sigma(t_i),$$

for all $i = 1, \ldots, n$?

Background for Unification

- **Terms** made from **constructors** and **variables** (for the simple first order case)
- Constructors may be **applied** to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (**arity**) considered different
- **Substitution** of terms for variables

Background for Unification

- **Terms** made from **constructors** and **variables** (for the simple first order case)
- Constructors may be **applied** to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (**arity**) considered different
- **Substitution** of terms for variables

Background for Unification

- **Terms** made from **constructors** and **variables** (for the simple first order case)
- Constructors may be **applied** to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (**arity**) considered different
- **Substitution** of terms for variables

Background for Unification

- **Terms** made from **constructors** and **variables** (for the simple first order case)
- Constructors may be **applied** to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (**arity**) considered different
- **Substitution** of terms for variables

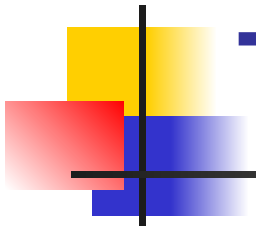
Background for Unification

- **Terms** made from **constructors** and **variables** (for the simple first order case)
- Constructors may be **applied** to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (**arity**) considered different
- **Substitution** of terms for variables

Background for Unification

- **Terms** made from **constructors** and **variables** (for the simple first order case)
- Constructors may be **applied** to arguments (other terms) to make new terms
- Variables and constructors with no arguments are base cases
- Constructors applied to different number of arguments (**arity**) considered different
- **Substitution** of terms for variables

Substituting a term t' for a variable x inside of another term t is often written t [t' / x]. For example, (y + 2)[3 / y] is 3 + 2.



Terms and Substitution

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name ->
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name ->
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name ->
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name ->
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name ->
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name ->
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name -> (* x [t' / x] is t' *)
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name -> (* x [t' / x] is t'; y [t' / x] is y *)
      if var_name = name then residue else term
  | Const (c, tys) ->
      Const (c, List.map (subst var_name residue) tys)
```

Terms and **Substitution**

```
type term = Var of string | Const of (string * term list)
let x = Var "'a" (* example variable *)
let tm = Const ("2",[]) (* example constructor *)

let rec subst var_name residue term = (* t [t' / x] *)
  match term with
  | Var name -> (* x [t' / x] is t'; y [t' / x] is y *)
      if var_name = name then residue else term
  | Const (c, tys) -> (* c(x, ...)[t' / x] = c(x [t' / x], ... *)
      Const (c, List.map (subst var_name residue) tys)
```

Unification Problem

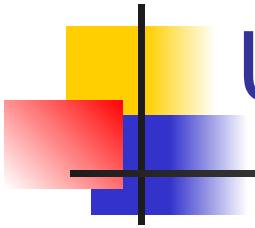
Given a set of pairs of terms ("equations")

$$\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$$

(the **unification problem**) does there exist a substitution σ (the **unification solution**) of terms for variables such that

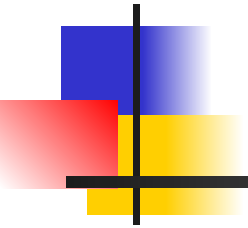
$$\sigma(s_i) = \sigma(t_i),$$

for all $i = 1, \ldots, n$?

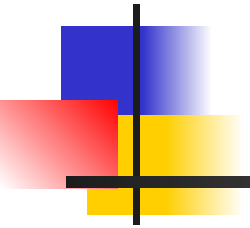


Uses for Unification

- **Type inference** and **type checking**
- **Pattern matching** as in OCaml
  - Can use a simplified version of algorithm
- **Logic programming** (e.g., Prolog)
- Simple **parsing**
- With fancy types: used in **proof synthesis/repair**



Questions so far?



Algorithm Overview

Unification Algorithm

Let $S = \{(s_1 = t_1), \ldots, (s_n = t_n)\}$ be a unification problem.

Solve by cases:

- **Case S = { } : Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case S = {(s, t)} ∪ S' ->**

  Four main steps:

  - Delete
  - Decompose
  - Orient
  - Eliminate

Algorithm

Unification Algorithm

Let $S = \{(s_1 = t_1), \ldots, (s_n = t_n)\}$ be a unification problem.

Solve by cases:

- **Case S = { } : Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case S = {(s, t)} ∪ S' ->**

  Four main steps:

  - Delete
  - Decompose
  - Orient
  - Eliminate

Unification Algorithm

Let $S = \{(s_1 = t_1), \ldots, (s_n = t_n)\}$ be a unification problem.

Solve by cases:

- **Case S = { } : Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case S = {(s, t)} ∪ S' ->**

  Four main steps:

  - Delete
  - Decompose
  - Orient
  - Eliminate

Unification Algorithm

Let $S = \{(s_1 = t_1), \ldots, (s_n = t_n)\}$ be a unification problem.

Solve by cases:

- **Case $S = \{\ \}$: Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case $S = \{(s, t)\} \cup S'$ ->**

  Four main steps:

  - Delete
  - Decompose
  - Orient
  - Eliminate

For nonempty S written

$\{(s_1 = t_1), \ldots, (s_n = t_n)\}$,

choose a pair $(s_i = t_i)$. Then we can write S as

$\{s_i = t_i\} \cup (\{(s_1 = t_1), \ldots, (s_n = t_n)\} - (s_i = t_i))$.

Let (s, t) be (s_i, t_i), and let S′ be

$\{(s_1 = t_1), \ldots, (s_n = t_n)\} - (s_i = t_i)$.

Then $S = \{(s, t)\} \cup S'$.

Algorithm

Unification Algorithm

Let $S = \{(s_1 = t_1), \ldots, (s_n = t_n)\}$ be a unification problem.

Solve by cases:

- **Case $S = \{\ \}$: Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case $S = \{(s, t)\} \cup S'$ ->**

  Four main steps:

  - Delete
  - Decompose
  - Orient
  - Eliminate

For nonempty S written

$\{(s_1 = t_1), \ldots, (s_n = t_n)\}$,

choose a pair $(s_i = t_i)$. Then we can write S as

$\{s_i = t_i\} \cup (\{(s_1 = t_1), \ldots, (s_n = t_n)\} - (s_i = t_i))$.

Let (s, t) be (s_i, t_i), and let S' be

$\{(s_1 = t_1), \ldots, (s_n = t_n)\} - (s_i = t_i)$.

Then $S = \{(s, t)\} \cup S'$.

Algorithm

Unification Algorithm

Let $S = \{(s_1 = t_1), \ldots, (s_n = t_n)\}$ be a unification problem.

Solve by cases:

- **Case S = { } : Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case S = {(s, t)} ∪ S' ->**

  Four main steps:

  - Delete
  - Decompose
  - Orient
  - Eliminate

Algorithm

Unification Algorithm

Let S = {(s_1 = t_1), …, (s_n = t_n)} be a unification problem.
Solve by cases:

- **Case S = { } : Unif(S) ->**

  Identity function (i.e., no substitution).

- **Case S = {(s, t)} ∪ S' ->**

  Four main steps:
  - Delete
  - Decompose
  - Orient
  - Eliminate

Algorithm

Unification Algorithm

Case S = {(s, t)} ∪ S' ->

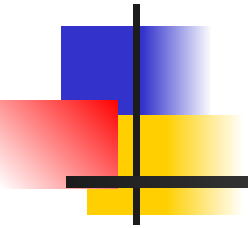
- **Delete:** if s = t (same term), consider just S'
- **Decompose:** if s and t apply the same function to the same number of arguments, consider just the pairs of arguments

Four main steps:

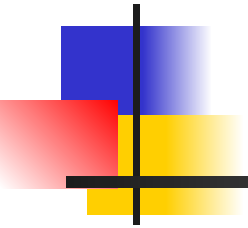
- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if t = x is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

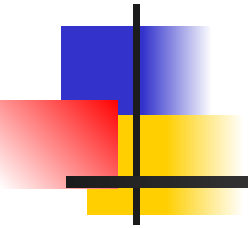
Algorithm



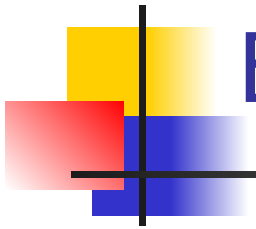
Questions so far?



Will make this more formal later.



But first, an example.



Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} = ?

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} = ?

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), **(g(y, y) = x)**} = ?

- Pick a pair: **(g(y,y) = x)**

Example

Example: Non-empty S, Orient

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), **(g(y, y) = x)**} = ?

- Pick a pair: **(g(y,y) = x)**
- **Orient**: **(x = g(y,y))**

Example

Example: Non-empty S, Orient

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), **(g(y, y) = x)**} =

Unify {(f(x) = f(g(f(z),y))), **(x = g(y,y))**} (Orient)

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

?

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), **(x = g(y,y))**} (Orient) =

?

- Pick a pair: **(x = g(y,y))**

Example: Non-empty S, Eliminate

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), **(x = g(y,y))**} (Orient) =

?

- Pick a pair: **(x = g(y,y))**
- **Eliminate** x with substitution {x→ g(y,y)}
  - Check: x not in g(y,y)

Example

Example: Non-empty S, Eliminate

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(**x**) = f(g(f(z),y))), **(x = g(y,y))**} (Orient) = ?

- Pick a pair: **(x = g(y,y))**
- **Eliminate** x with substitution **{x→ g(y,y)}**
  - Check: x not in g(y,y)

Example

Example: Non-empty S, Eliminate

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(**x**) = f(g(f(z),y))), **(x = g(y,y))**} (Orient) =

Unify {(f(**g(y,y)**) = f(g(f(z),y)))} ∘ {**x**→**g(y,y)**} (Elim)

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

?

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(**f**(**g**(**y**,**y**)) = **f**(**g**(**f**(**z**),**y**)))} ∘ {x→g(y,y)} (Elim) =

?

- Pick a pair: **(f(g(y,y)) = f(g(f(z),y)))**

Example

Example: Non-empty S, Decompose

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(**f**(g(y,y)) = **f**(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

?

- Pick a pair: **(f(g(y,y)) = f(g(f(z),y)))**
- **Decompose:**
  (**f**(g(y,y)) = **f**(g(f(z),y))) becomes
  {(g(y,y) = g(f(z),y))}

Example

Example: Non-empty S, Decompose

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(**f**(g(y,y)) = **f**(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp)

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

?

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(**g**(**y**,**y**) = **g**(**f**(**z**),**y**))} ∘ {x→g(y,y)} (Decomp) =

?

- Pick a pair: **(g(y,y) = g(f(z),y))**

Example

Example: Non-empty S, Decompose

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(**g**(y,y) = **g**(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

?

- Pick a pair: **(g(y,y) = g(f(z),y))**
- **Decompose:**

  (**g**(y,y) = **g**(f(z),y)) becomes

  {(y = f(z)); (y = y)}

Example

Example: Non-empty S, Decompose

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(**g**(y,y) = **g**(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp)

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

?

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(**y = f(z)**); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

?

- Pick a pair: **y = f(z)**

Example

Example: Non-empty S, Eliminate

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(**y = f(z)**); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

?

- Pick a pair: **y = f(z)**
- **Eliminate** y with substitution {y→ f(z)}
  - Check: y not in f(z)

Example

Example: Non-empty S, Eliminate

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(**y** = f(z)); (**y** = **y**)} ∘ {x→ g(**y**,**y**)} (Decomp) =

?

- Pick a pair: **y = f(z)**
- **Eliminate** y with substitution **{y→ f(z)}**
  - Check: y not in f(z)

Example

Example: Non-empty S, Eliminate

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(**y** = f(z)); (**y** = **y**)} ∘ {x→ g(**y**,**y**)} (Decomp) =

Unify {(**f**(**z**) = f(z))} ∘ {**y**→**f**(**z**); x→g(**f**(**z**),**f**(**z**))} (Elim)

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(f(z) = f(z))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

?

Example

Example: Non-empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(**f**(**z**) = **f**(**z**))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

?

- Pick a pair: **f(z) = f(z)**

Example

Example: Non-empty S, Delete

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(**f(z)** = **f(z)**)} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

?

- Pick a pair: **f(z) = f(z)**
- **Delete**

Example

Example: Non-empty S, Delete

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(**f**(**z**) = **f**(**z**))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

Unify {} ∘ {y→f(z); x→g(f(z), f(z))} (Delete)

Example

Example: Empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(f(z) = f(z))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

Unify **{}** ∘ {y→f(z); x→g(f(z), f(z))} (Delete) =

?

Example

Example: Empty S

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(f(z) = f(z))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

Unify **{}** ∘ {y→f(z); x→g(f(z), f(z))} (Delete) =

{y→f(z); x→g(f(z), f(z))} (Identity)

Example

Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(f(z) = f(z))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

Unify {} ∘ {y→f(z); x→g(f(z), f(z))} (Delete) =

{y→f(z); x→g(f(z), f(z))} (Identity)

Example

Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

Unify {(f(x) = f(g(f(z),y))), (x = g(y,y))} (Orient) =

Unify {(f(g(y,y)) = f(g(f(z),y)))} ∘ {x→g(y,y)} (Elim) =

Unify {(g(y,y) = g(f(z),y))} ∘ {x→g(y,y)} (Decomp) =

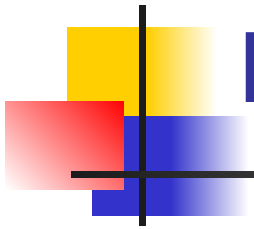
Unify {(y = f(z)); (y = y)} ∘ {x→ g(y,y)} (Decomp) =

Unify {(f(z) = f(z))} ∘ {y→f(z); x→g(f(z),f(z))} (Elim) =

Unify {} ∘ {y→f(z); x→g(f(z), f(z))} (Delete) =

{y→f(z); x→g(f(z), f(z))} (Identity)

Example

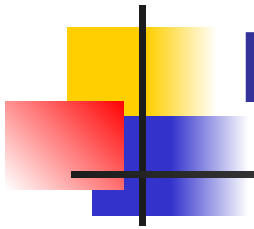


Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

{y→f(z); x→g(f(z), f(z))}



Example

x, y, z variables, f, g constructors

Unify {(**f(x) = f(g(f(z), y))**, (g(y, y) = x)} =

{y→f(z); x→g(f(z), f(z))}

From this, we can validate:

f(x) = f(g(f(z), y)) →

f(g(f(z), f(z))) = f(g(f(z), f(z)))

and:

g(y, y) = x →

g(f(z), f(z)) = g(f(z), f(z))

Example

Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

{**y**→**f(z)**; **x**→**g(f(z), f(z))**}

From this, we can validate:

f(**x**) = f(g(f(z), **y**)) →

f(**g(f(z), f(z))**) = f(g(f(z), **f(z)**))

and:

g(y, y) = x →

g(f(z), f(z)) = g(f(z), f(z))

Example

Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (**g(y, y) = x**)} =

{y→f(z); x→g(f(z), f(z))}

From this, we can validate:

f(x) = f(g(f(z), y)) →

f(g(f(z), f(z))) = f(g(f(z), f(z)))

and:

g(y, y) = x →

g(f(z), f(z)) = g(f(z), f(z))

Example

Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

{**y**→**f(z)**; **x**→**g(f(z), f(z))**}

From this, we can validate:

f(x) = f(g(f(z), y)) →

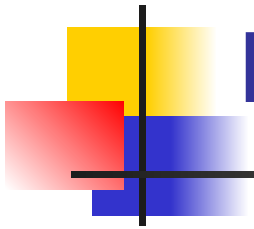
f(g(f(z), f(z))) = f(g(f(z), f(z)))

and:

g(**y**, **y**) = **x** →

g(**f(z)**, **f(z)**) = **g(f(z), f(z))**

Example



Example

x, y, z variables, f, g constructors

Unify {(f(x) = f(g(f(z), y))), (g(y, y) = x)} =

{y→f(z); x→g(f(z), f(z))}

From this, we can validate:

f(x) = f(g(f(z), y)) →

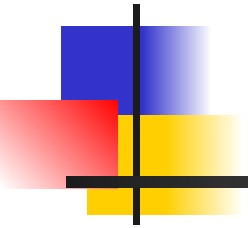
f(g(f(z), f(z))) = f(g(f(z), f(z)))

and:

g(y, y) = x →

g(f(z), f(z)) = g(f(z), f(z))

Example



Questions so far?

Example

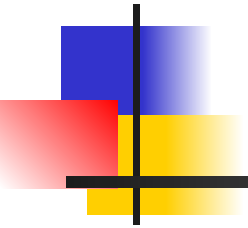
Example of Failure

Unify{(f(x, g(x)) = f(h(x), x))} =
Unify {(x = h(x)), (g(x) = x)} (Decomp) =
Unify {(x = h(x)), (x = g(x))} (Orient) =
?

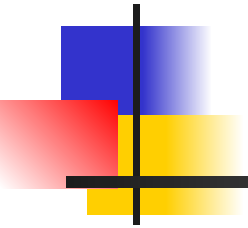
Example of Failure: Occurs Check

Unify{(f(x, g(x)) = f(h(x), x))} =
Unify {(x = h(x)), (g(x) = x)} (Decomp) =
Unify {(**x** = h(**x**)), (**x** = g(**x**))} (Orient) =
?

- No rules apply. x is in h(x), and x is in g(x).



Questions so far?



Algorithm, Revisited

Unification Algorithm

Case S = {(s, t)} ∪ S' ->

- **Delete:** if s = t (same term), consider just S'
- **Decompose:** if s and t apply the same function to the same number of arguments, consider just the pairs of arguments

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if t = x is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case S = {(s, t)} ∪ S' ->

- **Delete:** if s = t (same term), consider just S'
- **Decompose:** if s and t apply the same function to the same number of arguments, consider just the pairs of arguments

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if t = x is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case S = {(s, t)} ∪ S' ->

- **Delete:** if s = t (same term) then Unif(S) = Unif(S')
- **Decompose:** if s and t apply the same function to the same number of arguments, consider just the pairs of arguments

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if t = x is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case S = {(s, t)} ∪ S' ->

- **Delete:** if s = t (same term) then Unif(S) = Unif(S')
- **Decompose:** if s and t apply the same function to the same number of arguments, consider just the pairs of arguments

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if t = x is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case $S = \{(s, t)\} \cup S'$ ->

- **Delete:** if $s = t$ (same term) then $Unif(S) = Unif(S')$
- **Decompose:** if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ then $Unif(S) = Unif(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if $t = x$ is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case $S = \{(s, t)\} \cup S'$ ->

- **Delete:** if $s = t$ (same term) then $Unif(S) = Unif(S')$
- **Decompose:** if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ then $Unif(S) = Unif(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if $t = x$ is a variable, and s is not a variable, flip it so we get a variable on the LHS
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case $S = \{(s, t)\} \cup S'$ ->

- **Delete:** if $s = t$ (same term) then $Unif(S) = Unif(S')$
- **Decompose:** if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ then $Unif(S) = Unif(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if $t = x$ is a variable, and s is not a variable, $Unif(S) = Unif(\{(x = s)\} \cup S')$
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

Unification Algorithm

Case S = {(s, t)} ∪ S' ->

- **Delete:** if s = t (same term) then Unif(S) = Unif(S')
- **Decompose:** if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ then $Unif(S) = Unif(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if t = x is a variable, and s is not a variable, Unif(S) = Unif ({(x = s)} ∪ S')
- **Eliminate**: If we have a variable on the LHS not in the RHS, substitute

Algorithm

*

Unification Algorithm

Case $S = \{(s, t)\} \cup S'$ ->

- **Delete:** if $s = t$ (same term) then $Unif(S) = Unif(S')$
- **Decompose:** if $s = f(q_1, \ldots, q_m)$ and $t = f(r_1, \ldots, r_m)$ then $Unif(S) = Unif(\{(q_1, r_1), \ldots, (q_m, r_m)\} \cup S')$

Four main steps:

- Delete
- Decompose
- Orient
- Eliminate

- **Orient**: if $t = x$ is a variable, and s is not a variable, $Unif(S) = Unif(\{(x = s)\} \cup S')$
- **Eliminate**: If we have a variable on the LHS not in the RHS, needs own slide …

Algorithm

*

Unification Algorithm

- **Eliminate:** if s = x is a variable, and x does not occur in t (the occurs check), then
  - Let $\phi = \{x \rightarrow t\}$
    - $Unif(S) = Unif(\phi(S')) \circ \{x \rightarrow t\}$
  - Let $\psi = Unif(\phi(S'))$
    - $Unif(S) = \{x \rightarrow \psi(t)\} \circ \psi$
  - Note:

    $\{x \rightarrow a\} \circ \{y \rightarrow b\} =$

    $\{y \rightarrow (\{x \rightarrow a\}(b))\} \circ \{x \rightarrow a\}$

    if y not in a

Algorithm

*

Unification Algorithm

- **Eliminate:** if $s = x$ is a variable, and x does not occur in t (the occurs check), then
  - Let $\phi = \{x \rightarrow t\}$
    - Unif(S) = Unif(**ϕ(S′)**) o $\{x \rightarrow t\}$
  - Let ψ = Unif(ϕ(S′))
    - Unif(S) = $\{x \rightarrow \psi(t)\}$ o ψ
  - Note:

    $\{x \rightarrow a\}$ o $\{y \rightarrow b\}$ =

    $\{y \rightarrow (\{x \rightarrow a\}(b))\}$ o $\{x \rightarrow a\}$

    if y not in a

Algorithm

Unification Algorithm

- **Eliminate:** if s = x is a variable, and x does not occur in t (the occurs check), then
  - Let $\phi = \{x \rightarrow t\}$
    - Unif(S) = Unif(**ϕ(S′)**) o $\{x \rightarrow t\}$
  - Let **ψ** = Unif(**ϕ(S′)**)
    - Unif(S) = $\{x \rightarrow$ **ψ(t)**$\}$ o **ψ**
  - Note:

    $\{x \rightarrow a\}$ o $\{y \rightarrow b\}$ =

    $\{y \rightarrow (\{x \rightarrow a\}(b))\}$ o $\{x \rightarrow a\}$

    if y not in a

Algorithm

Unification Algorithm

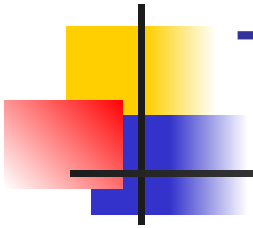
- **Eliminate:** if $s = x$ is a variable, and x does not occur in t (the occurs check), then
  - Let $\phi = \{x \rightarrow t\}$
    - $Unif(S) = Unif(\phi(S')) \circ \{x \rightarrow t\}$
  - Let $\psi = Unif(\phi(S'))$
    - $Unif(S) = \{x \rightarrow \psi(t)\} \circ \psi$
  - **Note**:

    $\{x \rightarrow a\} \circ \{y \rightarrow b\} =$

    $\{y \rightarrow (\{x \rightarrow a\}(b))\} \circ \{x \rightarrow a\}$

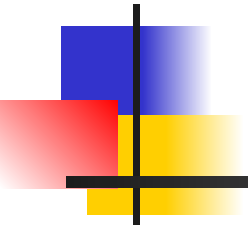
    if y not in a

Algorithm

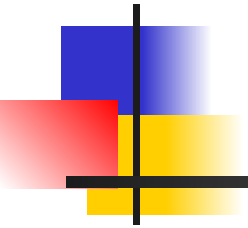


Tricks for Efficient Unification

- Don’t return substitution; rather, do it **incrementally**
- Make **substitution** be **constant time**
  - Requires implementation of terms to use **mutable** structures (or possibly **lazy** structures)
  - We won’t discuss these

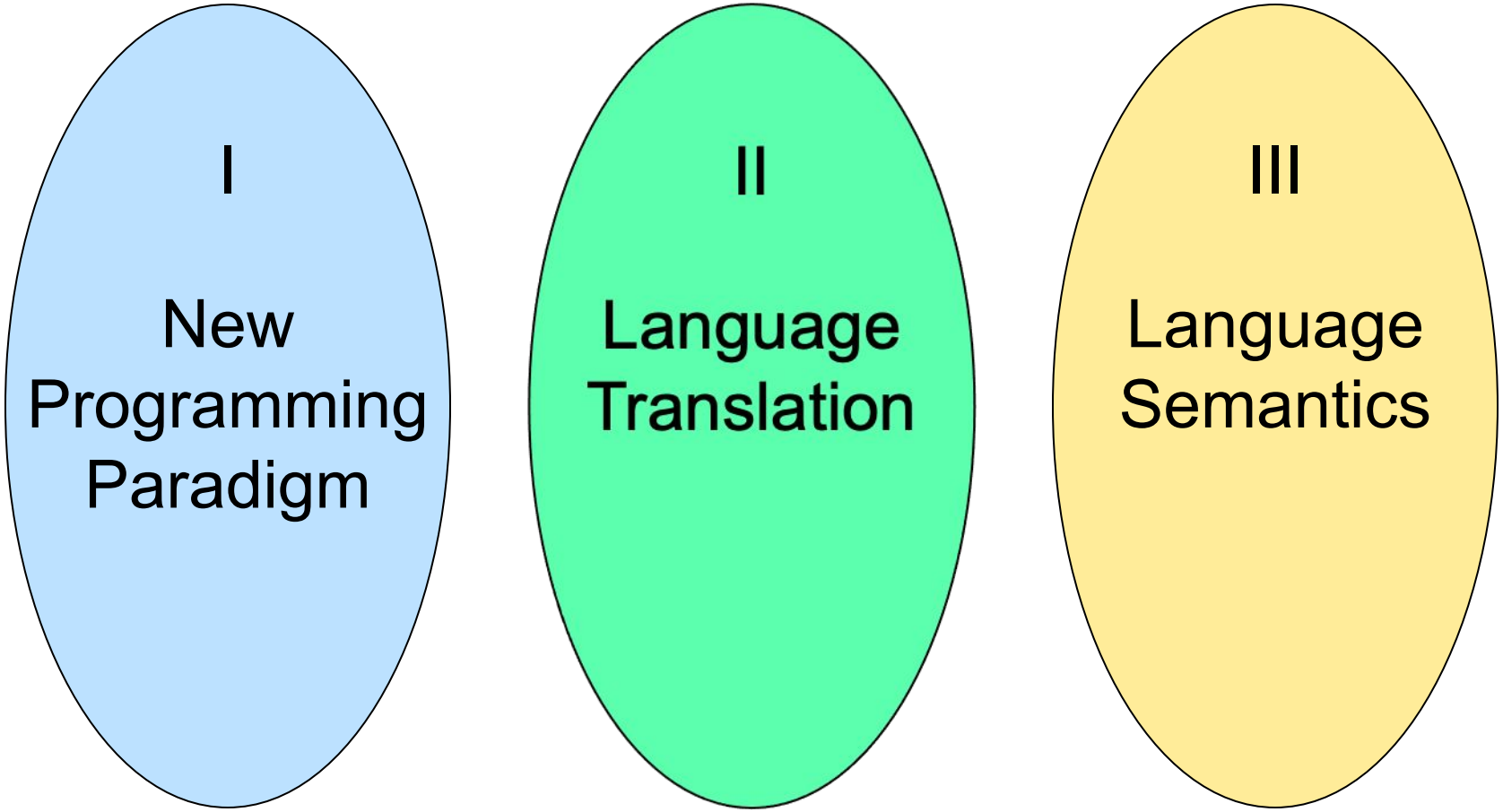


Questions?

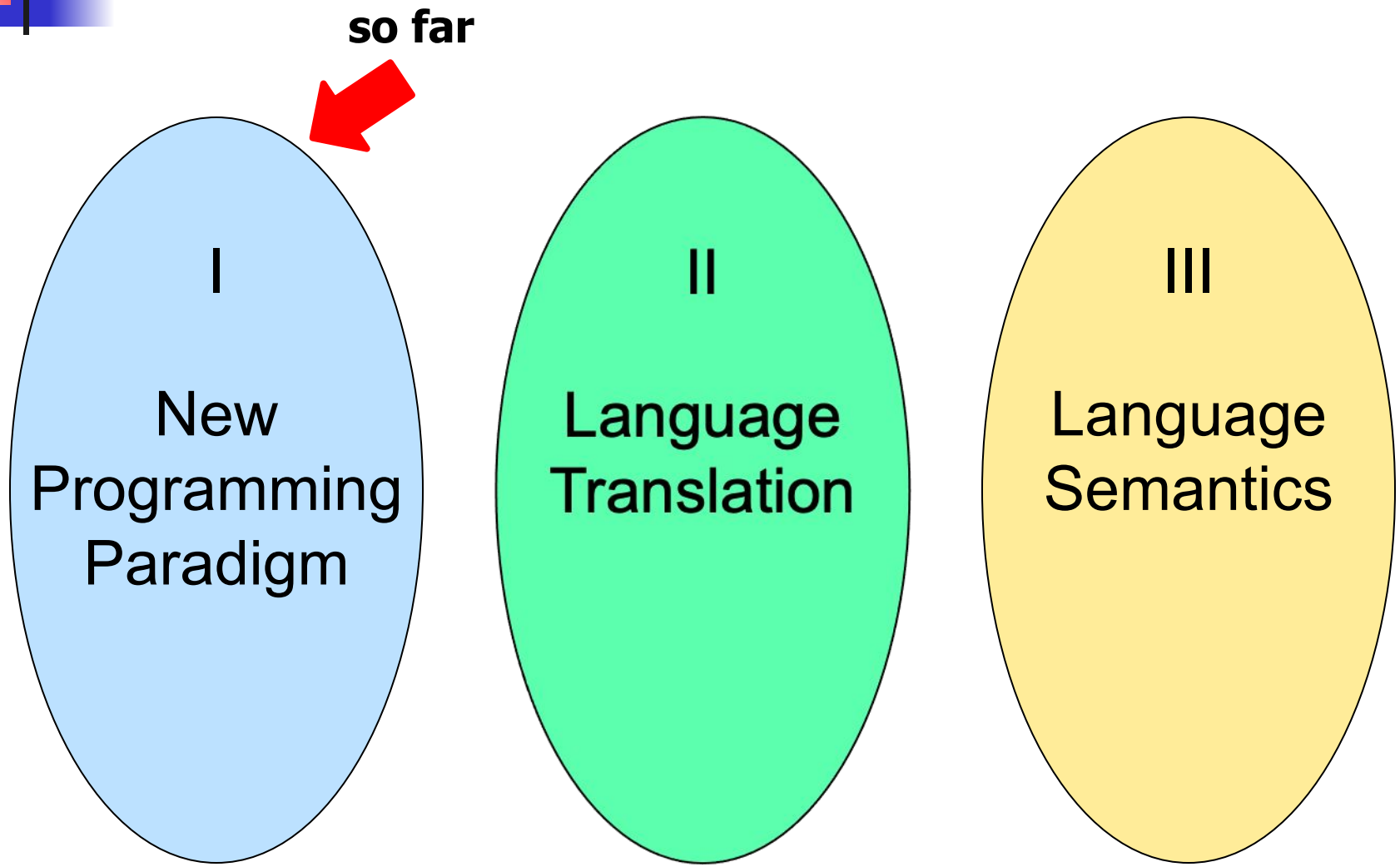


What's Next

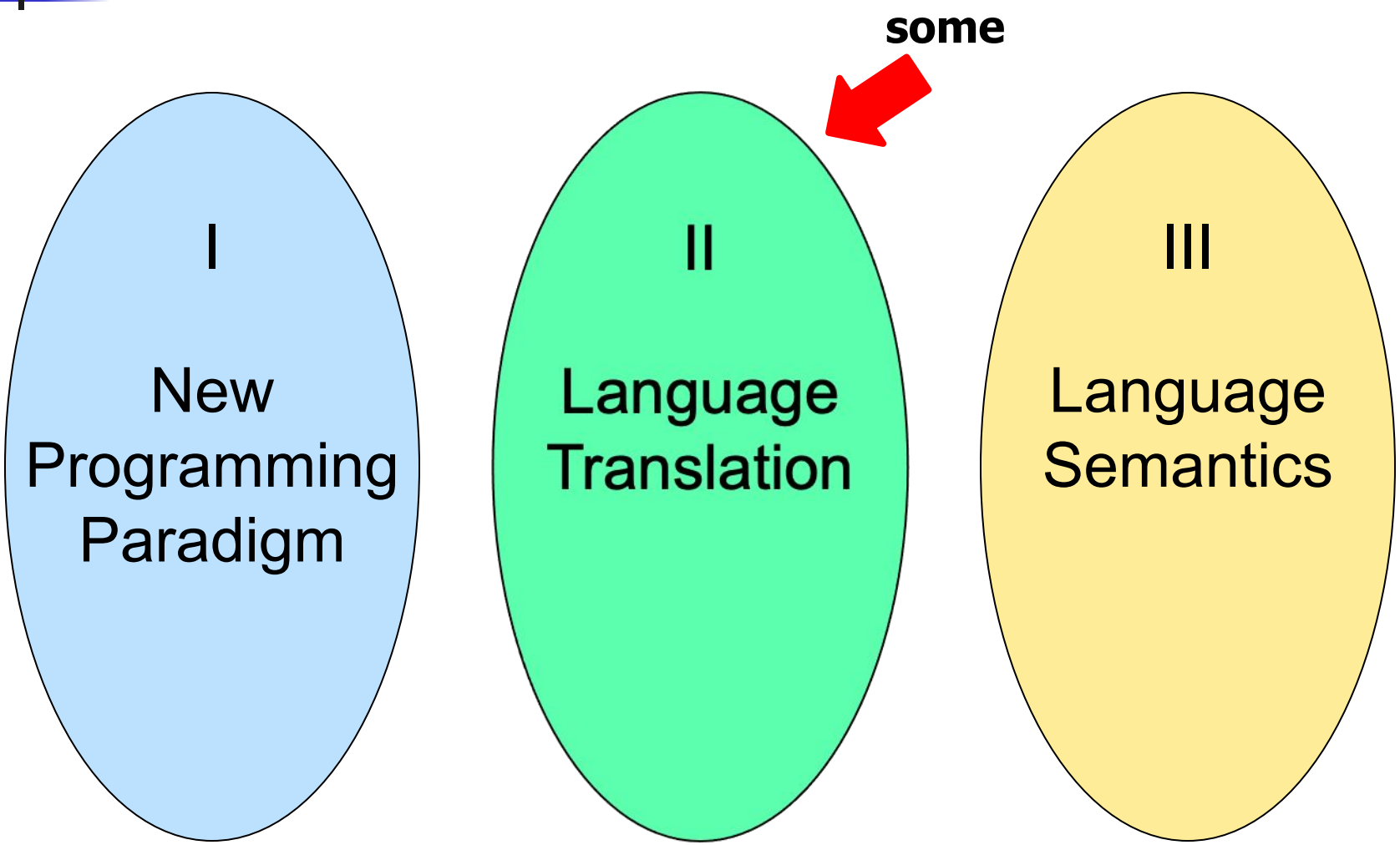
Three Main Topics



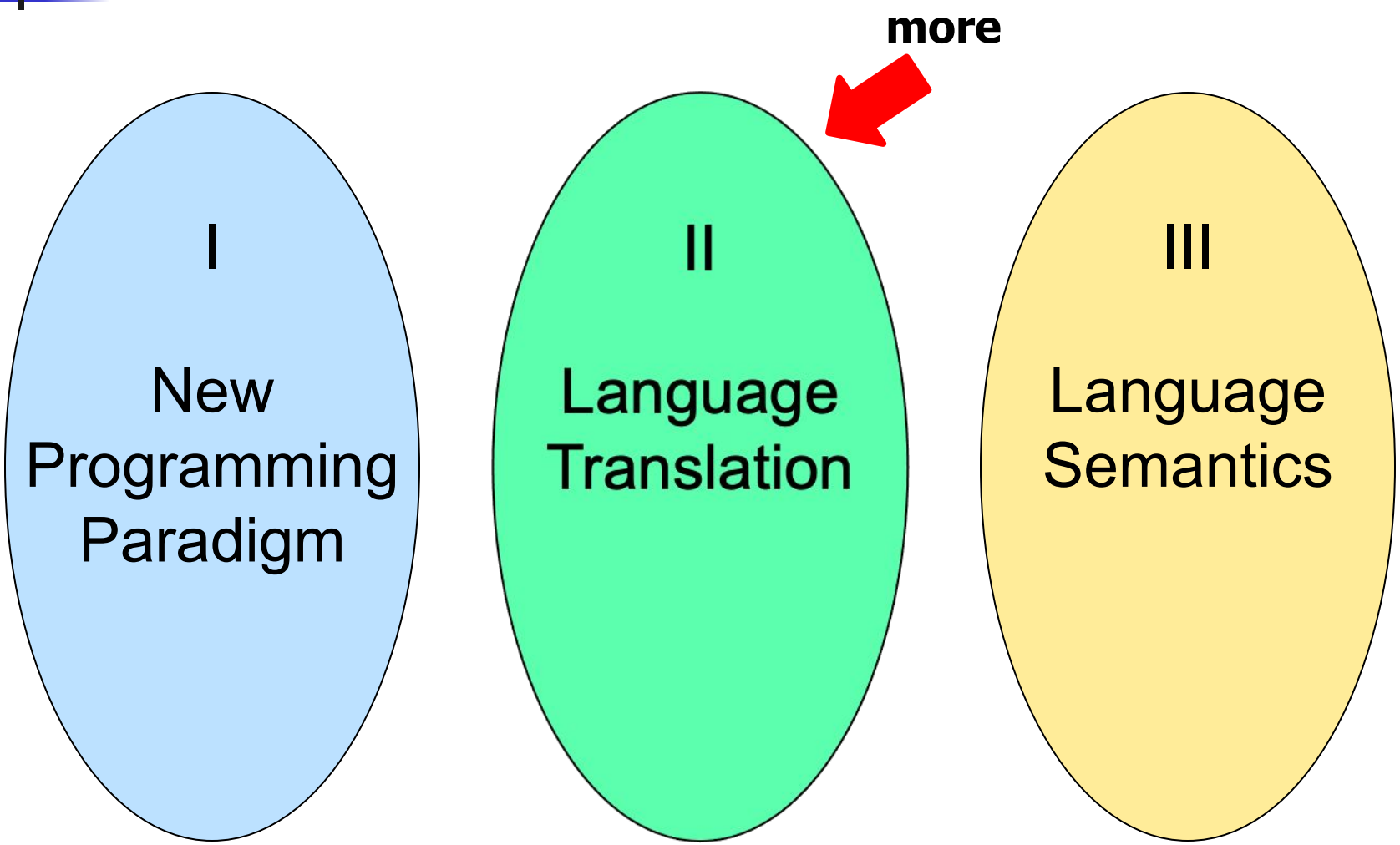
Three Main Topics



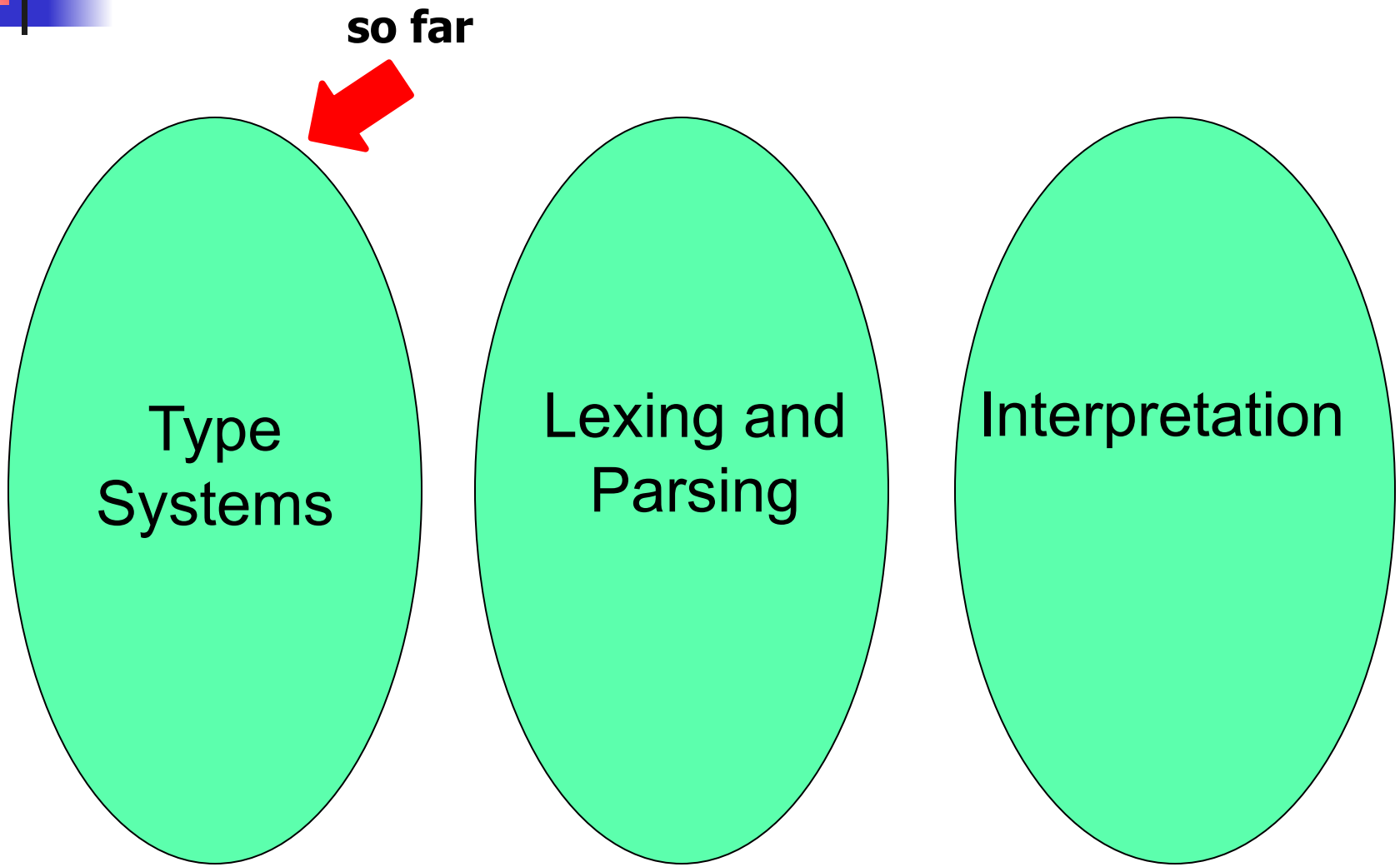
Three Main Topics



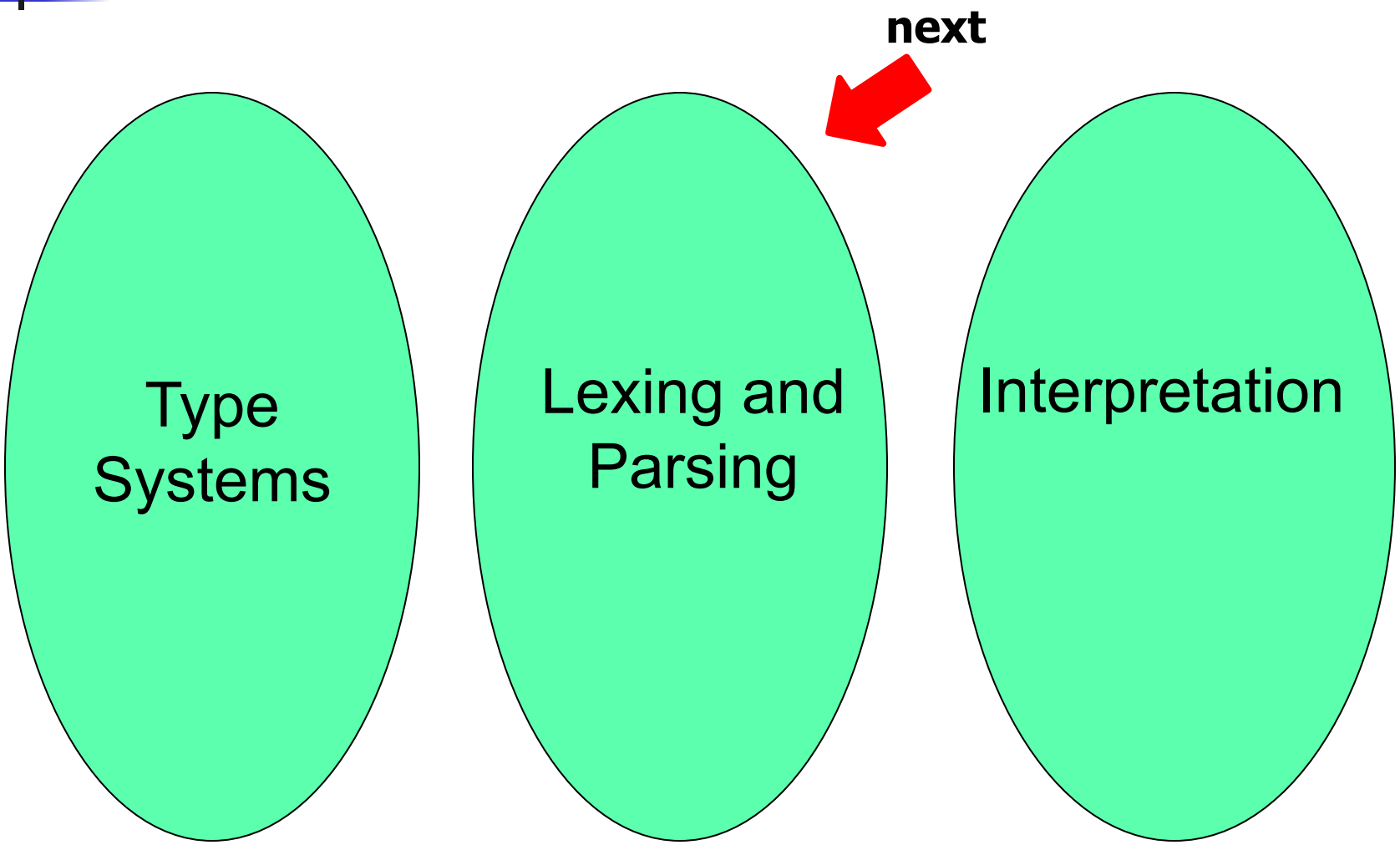
Three Main Topics



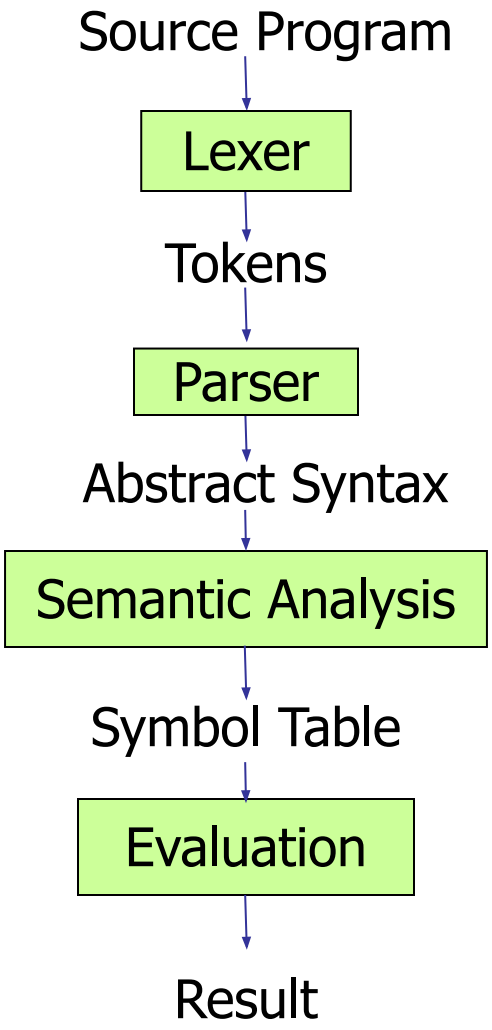
Language Translation



Language Translation

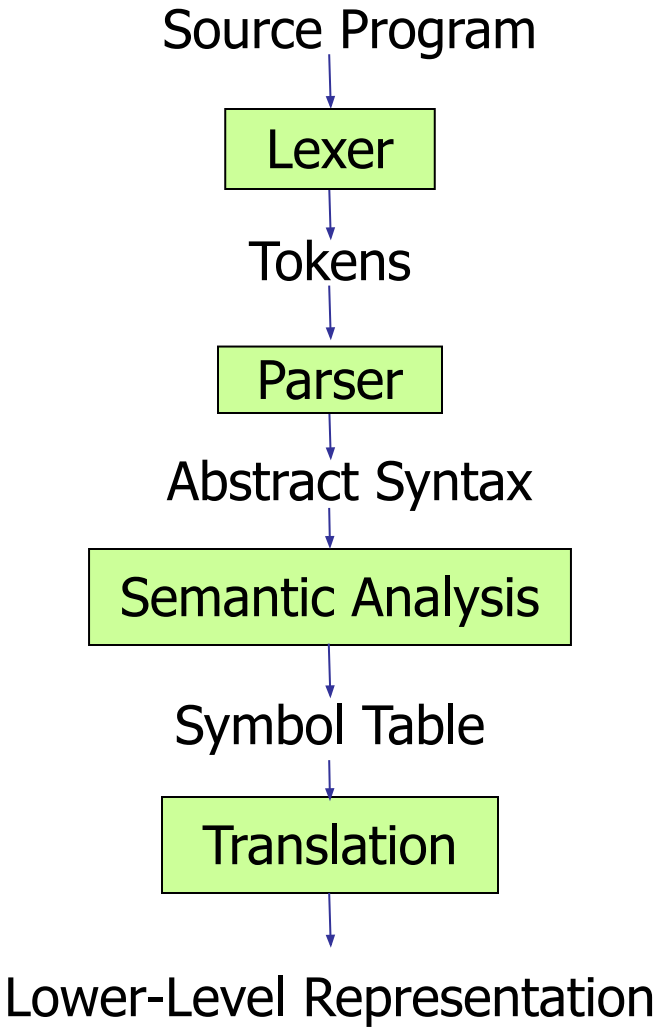


Every Interpreter Does This



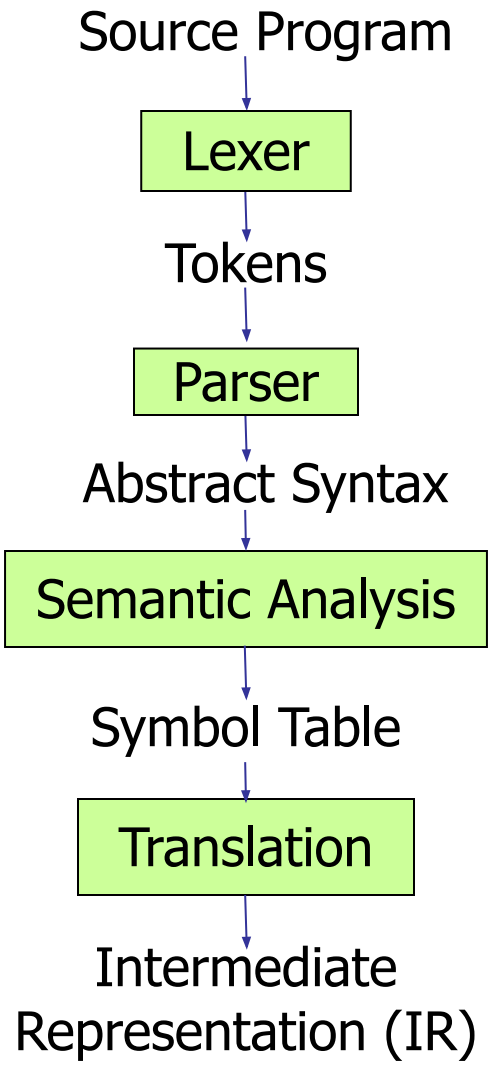
Modified from "Modern Compiler Implementation in ML" by Andrew Appel

Every Compiler Does This



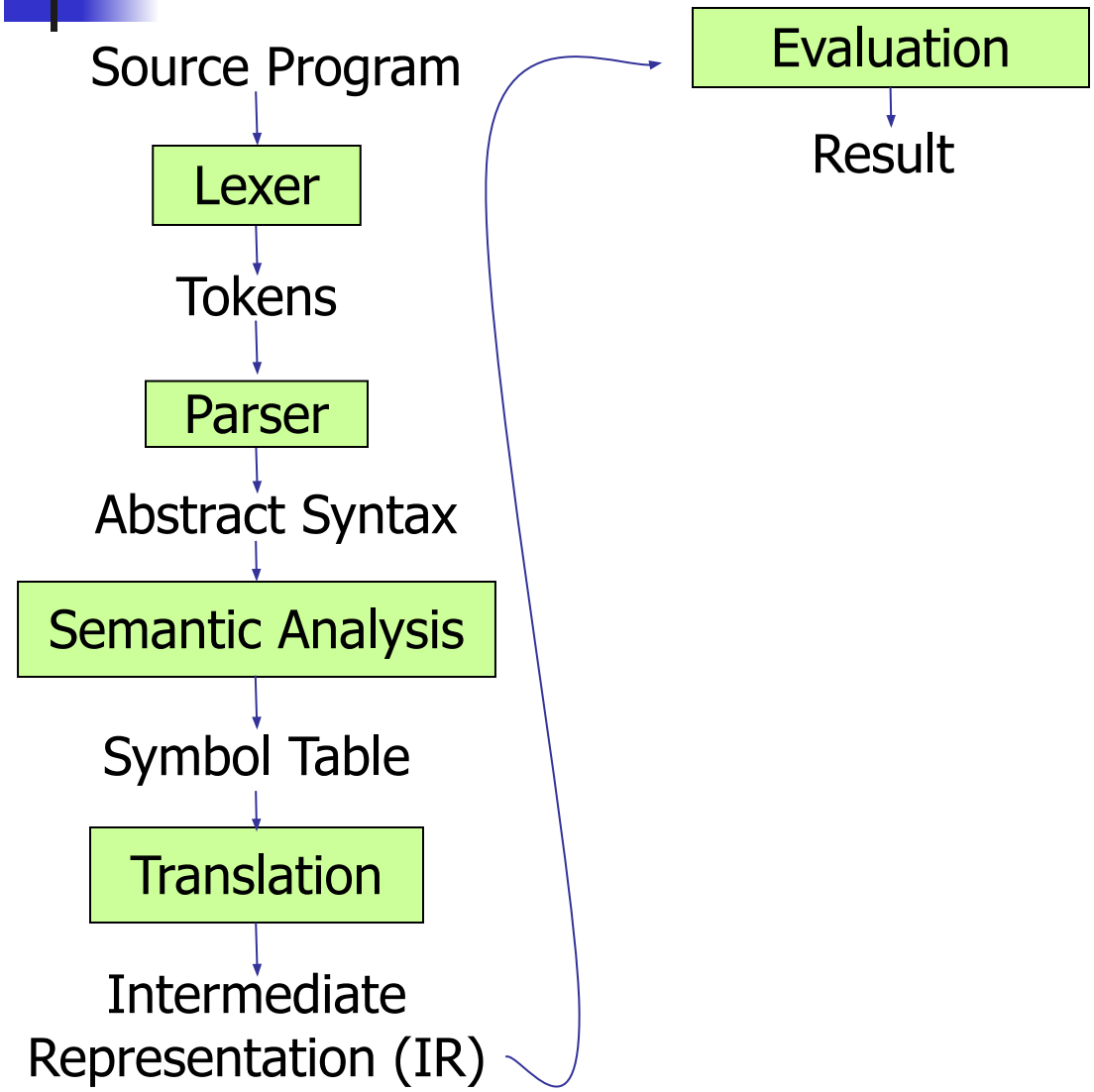
Modified from "Modern Compiler Implementation in ML" by Andrew Appel

Many Compilers Do This



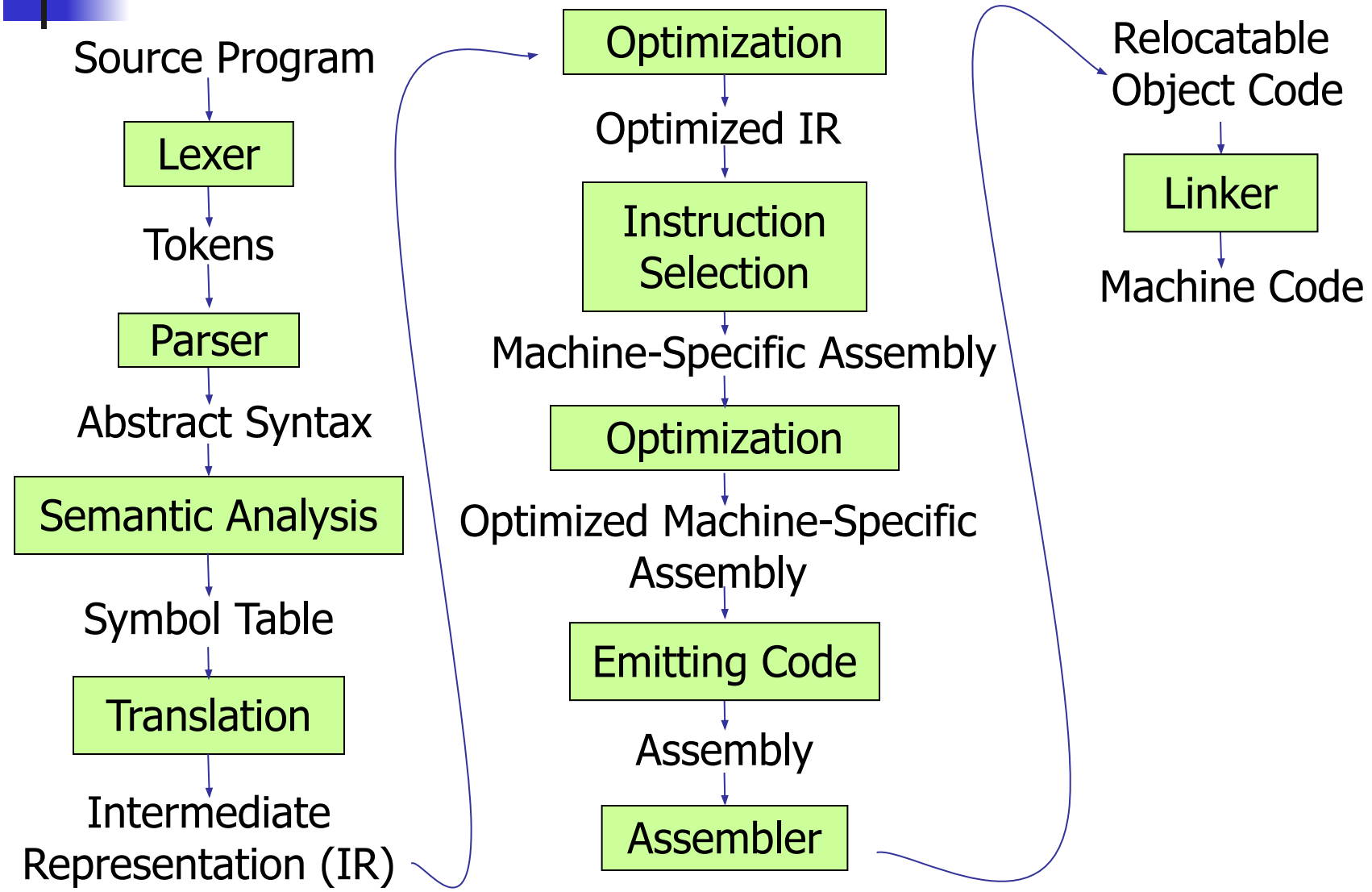
Modified from "Modern Compiler Implementation in ML" by Andrew Appel

Many Compilers Do This



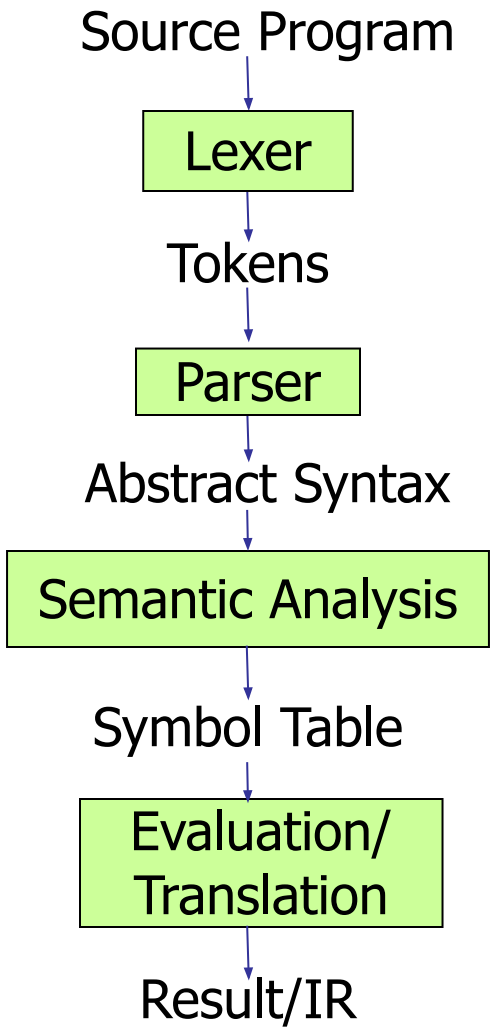
Modified from "Modern Compiler Implementation in ML" by Andrew Appel

Many Compilers Do This

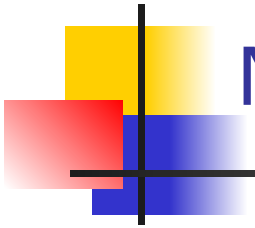


Modified from "Modern Compiler Implementation in ML" by Andrew Appel

We Will Stay Here



Modified from "Modern Compiler Implementation in ML" by Andrew Appel



Next Class

- **EC2 will be up**
- **WA6** due Thursday
- All deadlines can be found on **course website**
- Use **office hours** and **class forums** for help