Extending The Type Expression Language

$Type \rightarrow BasicType \mid ConstructedType \mid TypeName \mid QuantifiedType$

$BasicType \rightarrow$ **int** | **char** | **void** | ...

$ConstructedType \rightarrow$ **Array**($Type$) | **Prod**($Type$, $Type$) | **Func**($Type$, $Type$) | ...

$TypeName \rightarrow$ **TypeDef id** = $Type$

$QuantifiedType \rightarrow \forall A.\ Type \mid$	[Universal quantification]
$\exists A.\ Type \mid$	[Existential quantification]
$\forall A \subseteq Type.\ Type \mid \exists A \subseteq Type.\ Type$	[Bounded quantification]

- Universal quantification leads to parametric polymorphism.
- Existential quantification enables abstract data types with hidden representation.
- Bounded quantification provides a conceptual model for inheritance in object-oriented languages.

Example: Map-Reduce

- Consider the following ML-style function definitions:
  - `fun map f nil = nil`
    `| map f (a::r) = ((f a)::(map f r))`
  - `fun reduce g id nil = id`
    `| reduce g id (a::r) = g a (reduce g id r))`

Example: Map-Reduce

- Consider the following ML-style function definitions:
  - ```
    fun map f nil = nil
      | map f (a::r) = ((f a)::(map f r))
    ```
  - ```
    fun reduce g id nil = id
      | reduce g id (a::r) = g a (reduce g id r))
    ```
- Question 1: What are the polytypes of the functions `map` and `reduce` as defined above?
  - The infix `::` operator is the polymorphic list constructor, with polytype $\alpha \times \alpha\ \mathrm{List} \rightarrow \alpha\ \mathrm{List}$.
  - The name `nil` is the polymorphic empty list, with polytype $\alpha\ \mathrm{List}$.

Example: Map-Reduce

- Consider the following ML-style function definitions:
  - ```
    fun map f nil = nil
      | map f (a::r) = ((f a)::(map f r))
    ```
  - ```
    fun reduce g id nil = id
      | reduce g id (a::r) = g a (reduce g id r))
    ```
- Question 1: What are the polytypes of the functions `map` and `reduce` as defined above?
  - The infix `::` operator is the polymorphic list constructor, with polytype $\alpha \times \alpha\ \mathrm{List} \rightarrow \alpha\ \mathrm{List}$.
  - The name `nil` is the polymorphic empty list, with polytype $\alpha\ \mathrm{List}$.
- Question 2: What are the monotypes of the uses of `map` and `reduce` in the following expression?
  - `(reduce plus 0.0 (map sqrt (1::4::9::16::nil)))`
  - The function identifier `sqrt` has monotype $\mathbf{int} \rightarrow \mathbf{real}$.
  - The function identifier `plus` has monotype $\mathbf{real} \times \mathbf{real} \rightarrow \mathbf{real}$.

Answering Question 1

- Consider the following ML-style function definitions:
  - ```
    fun map f nil = nil
      | map f (a::r) = ((f a)::(map f r))
    ```
  - ```
    fun reduce g id nil = id
      | reduce g id (a::r) = g a (reduce g id r))
    ```
- Question 1: What are the polytypes of the functions `map` and `reduce`?
  - The infix `::` operator is the polymorphic list constructor, with polytype $\alpha \times \alpha\ \text{List} \rightarrow \alpha\ \text{List}$.
  - The name `nil` is the polymorphic empty list, with polytype $\alpha\ \text{List}$.

Answering Question 1

- Consider the following ML-style function definitions:
  - ```
    fun map f nil = nil
      | map f (a::r) = ((f a)::(map f r))
    ```
  - ```
    fun reduce g id nil = id
      | reduce g id (a::r) = g a (reduce g id r))
    ```
- Question 1: What are the polytypes of the functions `map` and `reduce`?
  - The infix `::` operator is the polymorphic list constructor, with polytype $\alpha \times \alpha \text{ List} \rightarrow \alpha \text{ List}$.
  - The name `nil` is the polymorphic empty list, with polytype α List.
- Strategy
  - Form a system of equations among unknown *type variables*.
    - The known information about `::` and `nil` are the "constants".
    - Multiple uses of the same identifier must have must be assigned the same type.
    - The conditional expression (|) has the same type as that of its two arms.
  - Solve this system of equations for the unknown type variables.

Forming (and Solving) Equations for `map`

```
fun map f nil = nil
  | map f (a::r) = ((f a)::(map f r))
```

- Let $\tau_{\mathbf{id}}$ be the type variable assigned to identifier **id**.

Forming (and Solving) Equations for `map`

```
fun map f nil = nil
  | map f (a::r) = ((f a)::(map f r))
```

- Let $\tau_{\mathbf{id}}$ be the type variable assigned to identifier $\mathbf{id}$.

- Solving these, we get $\tau_{\mathbf{map}} = (\gamma \rightarrow \delta) \times \gamma \text{ List} \rightarrow \delta \text{ List}$.
  - This is shorthand for $\tau_{\mathbf{map}} = \forall\, \gamma \forall\, \delta. (\gamma \rightarrow \delta) \times \gamma \text{ List} \rightarrow \delta \text{ List}$.

Forming (and Solving) Equations for `reduce`

```
fun reduce g id nil = id
  | reduce g id (a::r) = g a (reduce g id r))
```

- Let $\tau_{\mathbf{id}}$ be the type variable assigned to identifier $\mathbf{id}$.

- Solving these, we get $\tau_{\mathbf{reduce}} = (\alpha \times \alpha \to \alpha) \times \alpha \times \alpha \text{ List} \to \alpha$.
  - This is shorthand for $\tau_{\mathbf{reduce}} = \forall\, \alpha. (\alpha \times \alpha \to \alpha) \times \alpha \times \alpha \text{ List} \to \alpha$.

Answering Question 2

- What are the monotypes of the uses of `map` and `reduce` in the following expression?
  - `(reduce plus 0.0 (map sqrt (1::4::9::16::nil)))`
- What do we know?
  - The function identifier `sqrt` has monotype $\mathbf{int} \rightarrow \mathbf{real}$.
  - The function identifier `plus` has monotype $\mathbf{real} \times \mathbf{real} \rightarrow \mathbf{real}$.

Answering Question 2

- What are the monotypes of the uses of `map` and `reduce` in the following expression?
  - `(reduce plus 0.0 (map sqrt (1::4::9::16::nil)))`
- What do we know?
  - The function identifier `sqrt` has monotype $\mathbf{int} \rightarrow \mathbf{real}$.
  - The function identifier `plus` has monotype $\mathbf{real} \times \mathbf{real} \rightarrow \mathbf{real}$.
  - The function identifier `map` has polytype $\forall\, \gamma \forall\, \delta. (\gamma \rightarrow \delta) \times \gamma \text{ List} \rightarrow \delta \text{ List}$.
  - The function identifier `reduce` has polytype $\forall\, \alpha. (\alpha \times \alpha \rightarrow \alpha) \times \alpha \times \alpha \text{ List} \rightarrow \alpha$.

Answering Question 2

- What are the monotypes of the uses of `map` and `reduce` in the following expression?
    - `(reduce plus 0.0 (map sqrt (1::4::9::16::nil)))`
- What do we know?
    - The function identifier `sqrt` has monotype $\mathbf{int} \rightarrow \mathbf{real}$.
    - The function identifier `plus` has monotype $\mathbf{real} \times \mathbf{real} \rightarrow \mathbf{real}$.
    - The function identifier `map` has polytype $\forall\, \gamma \forall\, \delta. (\gamma \rightarrow \delta) \times \gamma \text{ List} \rightarrow \delta \text{ List}$.
    - The function identifier `reduce` has polytype $\forall\, \alpha. (\alpha \times \alpha \rightarrow \alpha) \times \alpha \times \alpha \text{ List} \rightarrow \alpha$.
    - The literal 0.0 has type $\mathbf{real}$.
    - The expression (1::4::9::16::nil) has type $\mathbf{int}$ List.

Answering Question 2

- What are the monotypes of the uses of `map` and `reduce` in the following expression?
  - `(reduce plus 0.0 (map sqrt (1::4::9::16::nil)))`
- What do we know?
  - The function identifier `sqrt` has monotype $\mathbf{int} \to \mathbf{real}$.
  - The function identifier `plus` has monotype $\mathbf{real} \times \mathbf{real} \to \mathbf{real}$.
  - The function identifier `map` has polytype $\forall\, \gamma \forall\, \delta.\, (\gamma \to \delta) \times \gamma\ \mathrm{List} \to \delta\ \mathrm{List}$.
  - The function identifier `reduce` has polytype $\forall\, \alpha.\, (\alpha \times \alpha \to \alpha) \times \alpha \times \alpha\ \mathrm{List} \to \alpha$.
  - The literal 0.0 has type $\mathbf{real}$.
  - The expression (1::4::9::16::nil) has type $\mathbf{int}\ \mathrm{List}$.
- So the uses have the following monotypes.
  - For map: $(\mathbf{int} \to \mathbf{real}) \times \mathbf{int}\ \mathrm{List} \to \mathbf{real}\ \mathrm{List}$.
  - For reduce: $(\mathbf{real} \times \mathbf{real} \to \mathbf{real}) \times \mathbf{real} \times \mathbf{real}\ \mathrm{List} \to \mathbf{real}$.