Evaluation Strategies

CS 4301/6371: Advanced Programming Languages

Kevin W. Hamlen

April 11–16, 2024

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\rightarrow}$
System F
OCaml
Haskell

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C ✗
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\rightarrow}$
System F
OCaml
Haskell

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓
$\lambda_{\rightarrow}$	✓
System F	
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	✓
Haskell	

First-class

Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	✓
Haskell	✓

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\to}$
System F
OCaml
Haskell

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C ✗
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda_{\to}$
- System F
- OCaml
- Haskell

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	
Java	
JavaScript	
Python	
$\lambda_\to$	
System F	
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	
JavaScript	
Python	
$\lambda_{\to}$	
System F	
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	
Python	
$\lambda_\to$	
System F	
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	
$\lambda_{\to}$	
System F	
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓(with functools)
$\lambda_{\to}$	
System F	
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \rightarrow \cdots \rightarrow \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓(with functools)
$\lambda_{\rightarrow}$	✓
System F	
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓(with functools)
$\lambda_\to$	✓
System F	✓
OCaml	
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓(with functools)
$\lambda_{\to}$	✓
System F	✓
OCaml	✓
Haskell	

Partial Evaluation

Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

C	✗
C++	✗
SIMPL	✓
Java	✗
JavaScript	✓
Python	✓(with functools)
$\lambda_{\to}$	✓
System F	✓
OCaml	✓
Haskell	✓

Eager Evaluation

Definition (Eager Semantics or Call-by-value): An *eager* or *call-by-value* language evaluates all function arguments before passing them as parameters.

Operational semantics look like this:

$$\frac{\forall i \in [1,n], \langle e_i, \sigma \rangle \Downarrow u_i \qquad \sigma(f)(u_1, \ldots, u_n) \Downarrow u'}{\langle f(e_1, \ldots, e_n), \sigma \rangle \Downarrow u'}$$

Lazy Evaluation

Definition (Lazy Semantics): A *lazy* language evaluates function arguments after the function body has started evaluating. There are two main varieties:

- **Call-by-name** languages (re)evaluate each argument expression each time the function uses it.
  - Can be formalized via capture-avoiding substitution
  - Disadvantage: usually inefficient
  - Advantage: sometimes highly efficient (e.g., unused arguments, highly parallelizable languages)
- **Call-by-need** languages evaluate each argument at first use, then memoize and reuse those values at subsequent uses.
  - Advantage: highest efficiency (usually)
  - Disadvantage: sometimes unintuitive!

Optional Exercises: Devise call-by-value and call-by-need operational semantics for λ-calculus

Call-by-reference

Definition (call-by-reference): Languages supporting *call-by-reference* allow callees to destructively modify the values of variables passed as arguments.

Example: Most object-oriented languages pass objects by reference, allowing callees to globally modify the object's fields instead of receiving a local copy of the object.

Note: Call-by-reference does not make sense for immutable variables.

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda_{\rightarrow}$
- System F
- OCaml
- Haskell

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	call-by-value, call-by-reference (objects)
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	call-by-value, call-by-reference (objects)
Python	call-by-value, call-by-reference (everything mutable!)
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	call-by-value, call-by-reference (objects)
Python	call-by-value, call-by-reference (everything mutable!)
$\lambda_{\rightarrow}$	call-by-name
System F	
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	call-by-value, call-by-reference (objects)
Python	call-by-value, call-by-reference (everything mutable!)
$\lambda_{\rightarrow}$	call-by-name
System F	call-by-name
OCaml	
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	call-by-value, call-by-reference (objects)
Python	call-by-value, call-by-reference (everything mutable!)
$\lambda_{\rightarrow}$	call-by-name
System F	call-by-name
OCaml	call-by-value, call-by-reference (objects)
Haskell	

Evaluation Strategies

Which evaluation strategies are supported by the following languages?

C	call-by-value
C++	call-by-value, call-by-reference
SIMPL	call-by-value
Java	call-by-value, call-by-reference (objects)
JavaScript	call-by-value, call-by-reference (objects)
Python	call-by-value, call-by-reference (everything mutable!)
$\lambda_{\rightarrow}$	call-by-name
System F	call-by-name
OCaml	call-by-value, call-by-reference (objects)
Haskell	call-by-need

Church-Rosser Property

Definition (Church-Rosser): Languages with the *Church-Rosser Property* are those in which the order of evaluation has no impact on the observable result. More technically, they are those languages whose small-step operational semantics are *confluent*.

Church-Rosser languages typically...

- have strictly immutable variables,
- are *pure* (i.e., free of side-effects).

Languages that are Church-Rosser can have unknown evaluation strategies (unobservable to the user), and offer compilers many optimization opportunities.

Static vs. Dynamic Typing

Definition (static/dynamic typing): A language is *(strictly) statically typed* if all types are erased during compilation. In contrast, a language is *dynamically typed* if types are available at runtime (usually attached to runtime values).

Advantages of strict static typing:

- space- and time-efficiency (no runtime storage or tracking of types)
- types facilitate static debugging
- types facilitate compile-time static code optimization
- types can be more universal (e.g., characterizing all possible executions)

Advantages of dynamic typing:

- type-tag values available at runtime (whether you need them or not)
- sometimes easier patching and bug mitigation
- opportunities for extra security sanity-checking

Strict Static Typing

Which of the following languages are strictly statically typed?

C
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\rightarrow}$
System F
OCaml
Haskell

Strict Static Typing

Which of the following languages are strictly statically typed?

- C ✓
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda_{\rightarrow}$
- System F
- OCaml
- Haskell

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	✗
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✓
System F	
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	✓ (except for objects)
Haskell	

Strict Static Typing

Which of the following languages are strictly statically typed?

C	✓
C++	✓
SIMPL	✓
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	✓ (except for objects)
Haskell	✓

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\rightarrow}$
System F
OCaml
Haskell

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C ✗
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\rightarrow}$
System F
OCaml
Haskell

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	
JavaScript	
Python	
$\lambda_\rightarrow$	
System F	
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	☹ (stuckness formalized as exception)
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	☹ (stuckness formalized as exception)
Python	☹ (stuckness formalized as exception)
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	☹ (stuckness formalized as exception)
Python	☹ (stuckness formalized as exception)
$\lambda_{\rightarrow}$	✓
System F	
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	☹ (stuckness formalized as exception)
Python	☹ (stuckness formalized as exception)
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	☹ (stuckness formalized as exception)
Python	☹ (stuckness formalized as exception)
$\lambda_\rightarrow$	✓
System F	✓
OCaml	✓
Haskell	

Type-safety

Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C	✗
C++	✗
SIMPL	✓
Java	☹ (stuckness formalized as exception)
JavaScript	☹ (stuckness formalized as exception)
Python	☹ (stuckness formalized as exception)
$\lambda_{\rightarrow}$	✓
System F	✓
OCaml	✓
Haskell	✓

Polymorphism

Definition (polymorphism): A language is *polymorphic* if interfaces (e.g., functions) can accommodate entities (e.g., arguments) of multiple different types.

Three main varieties:

1. **Parametric Polymorphism:** type system has type-variables α
   - facilitates machine-checked code-reuse idioms
   - compatible with strictly static type-safety
2. **Subtyping Polymorphism:** object types arranged in a hierarchy
   - hallmark of object-oriented programming
   - static semantics usually characterized by a weakening rule:

     $$\frac{\Gamma \vdash e : \tau \qquad \tau \preceq \tau'}{\Gamma \vdash e : \tau'}$$

   - Warning: makes structural induction proofs much harder (Why?)
3. **Ad hoc Polymorphism:** conditionals can test types at runtime
   - opens the door for arbitrarily heterogeneous code blocks per type
   - antithesis of code-reuse (much harder to maintain and debug)

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda_{\rightarrow}$
- System F
- OCaml
- Haskell

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	subtyping, ad hoc
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	subtyping, ad hoc
Python	parametric (generics), subtyping, ad hoc
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	subtyping, ad hoc
Python	parametric (generics), subtyping, ad hoc
$\lambda_{\rightarrow}$	none
System F	
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	subtyping, ad hoc
Python	parametric (generics), subtyping, ad hoc
$\lambda_{\rightarrow}$	none
System F	parametric
OCaml	
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	subtyping, ad hoc
Python	parametric (generics), subtyping, ad hoc
$\lambda_{\rightarrow}$	none
System F	parametric
OCaml	parametric, subtyping
Haskell	

Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

C	none
C++	subtyping
SIMPL	none
Java	parametric (generics), subtyping, ad hoc
JavaScript	subtyping, ad hoc
Python	parametric (generics), subtyping, ad hoc
$\lambda_{\rightarrow}$	none
System F	parametric
OCaml	parametric, subtyping
Haskell	parametric

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C
C++
SIMPL
Java
JavaScript
Python
$\lambda_{\rightarrow}$
System F
OCaml
Haskell

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C ✗
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda_{\rightarrow}$
- System F
- OCaml
- Haskell

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	✗
Python	
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	
System F	
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✗
System F	
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✗
System F	✓
OCaml	
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✗
System F	✓
OCaml	✓(with `--rectypes`)
Haskell	

Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

C	✗
C++	✗
SIMPL	✗
Java	✗
JavaScript	✗
Python	✗
$\lambda_{\rightarrow}$	✗
System F	✓
OCaml	✓(with `--rectypes`)
Haskell	✓

Summary Table

	first-class functions	currying & partial evaluation	evaluation strategies	strict static typing	type safety	polymorphism	non-shallow types
C	✗	✗	Val	✓	✗	✗	✗
C++	✗	✗	Val,Ref	✓	✗	$\preceq$	✗
SIMPL	✓	✓	Val	✓	✓	✗	✗
Java	✗	✗	Val,Ref	✗	☹	α,$\preceq$,?:	✗
JavaScript	✓	✓	Val,Ref	✗	☹	$\preceq$,?:	✗
Python	✓	✓	Val,Ref	✗	☹	α,$\preceq$,?:	✗
$\lambda_\rightarrow$	✓	✓	Name	✓	✓	✗	✗
System F	✓	✓	Name	✓	✓	α	✓
OCaml	✓	✓	Val,Ref	✓	✓	α,$\preceq$	✓
Haskell	✓	✓	Need	✓	✓	α	✓