# Funcons

## Small Step Interpretation

L. Thomas van Binsbergen

Royal Holloway, University of London

7 March, 2015

Section 1

# Semantics of Programming Languages

# Formal Definition of Programming Languages

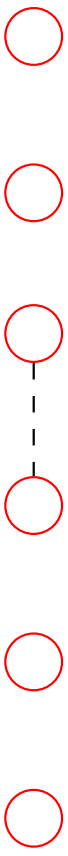
- Regular expressions are used to define lexical syntax formally.
- BNF is used to define context-free syntax formally.
- What is the semantics of a programming language?
- We distinguish between:
  - *Static* semantics;
    Analyses at compile-time, to rule out nonsensical programs.
  - *Dynamic* semantics;
    What is the precise run-time behaviour of a program?
- How to define the semantics of a language formally?
- Both semantics require context-sensitive analysis.

# Formalisms for Formal Semantics

- Attribute Grammars (AGs)
- Structural Operational Semantics (SOS)
  - Big step (natural semantics)
  - Small step
- Others: Denotational Semantics, Term Rewriting, ...
- PLanCompS:
  Modular variant of small-step SOS defines *reusable* programming constructs called *Funcons*.

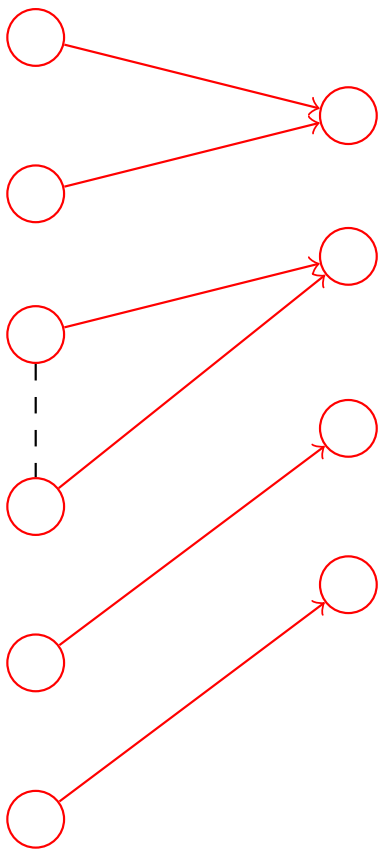
# Reusable Components: Funcons

Java

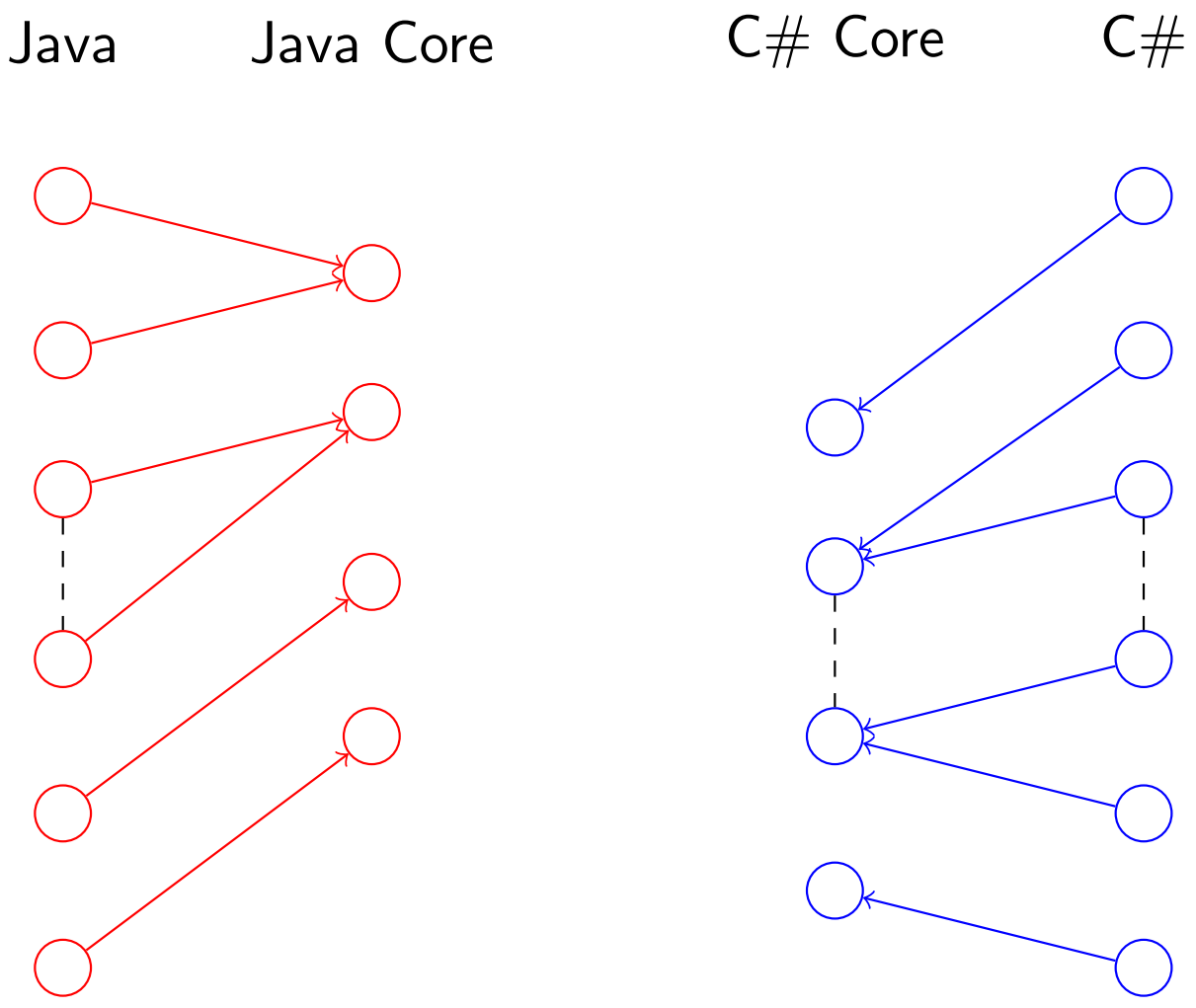


# Reusable Components: Funcons

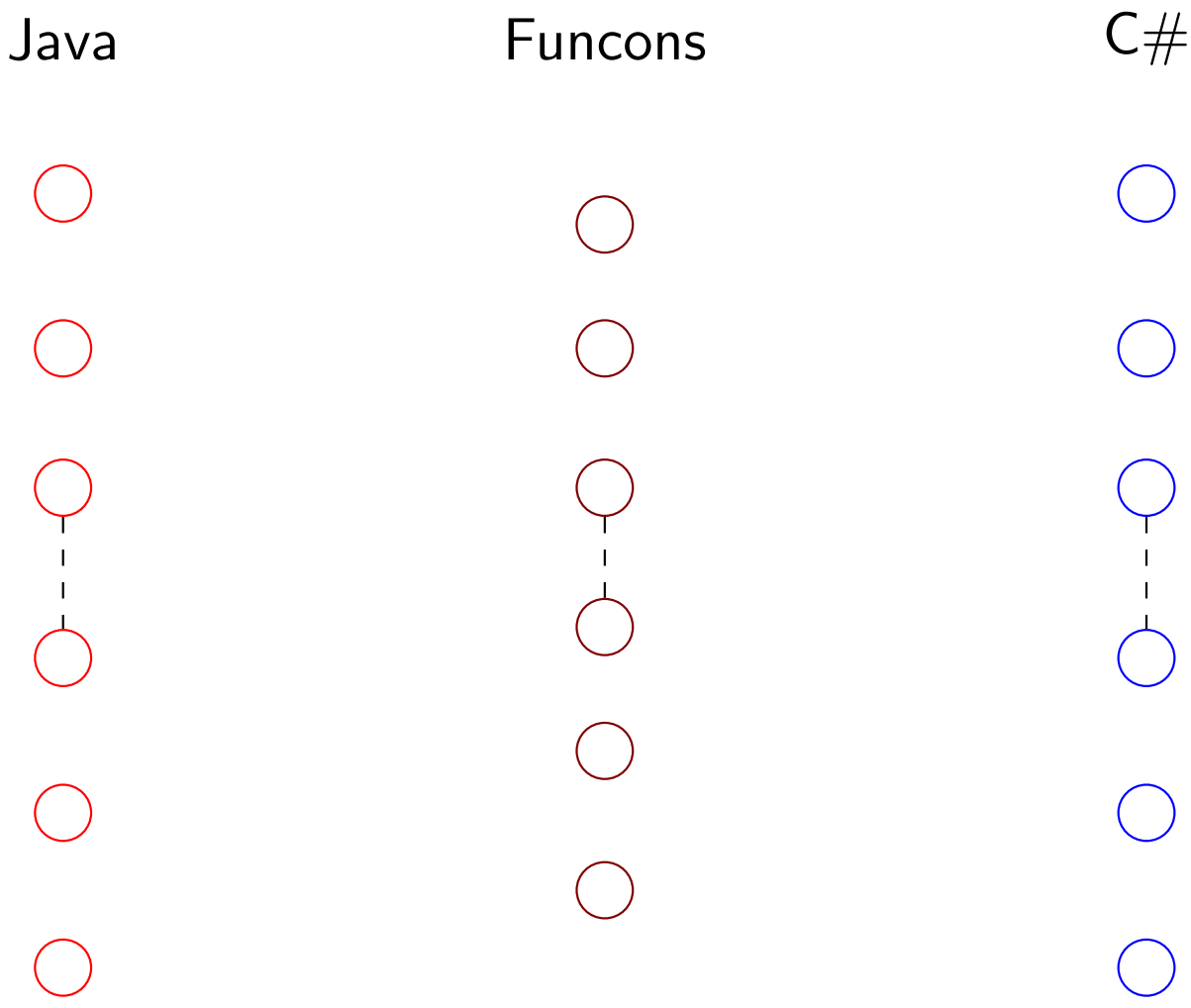
Java Java Core



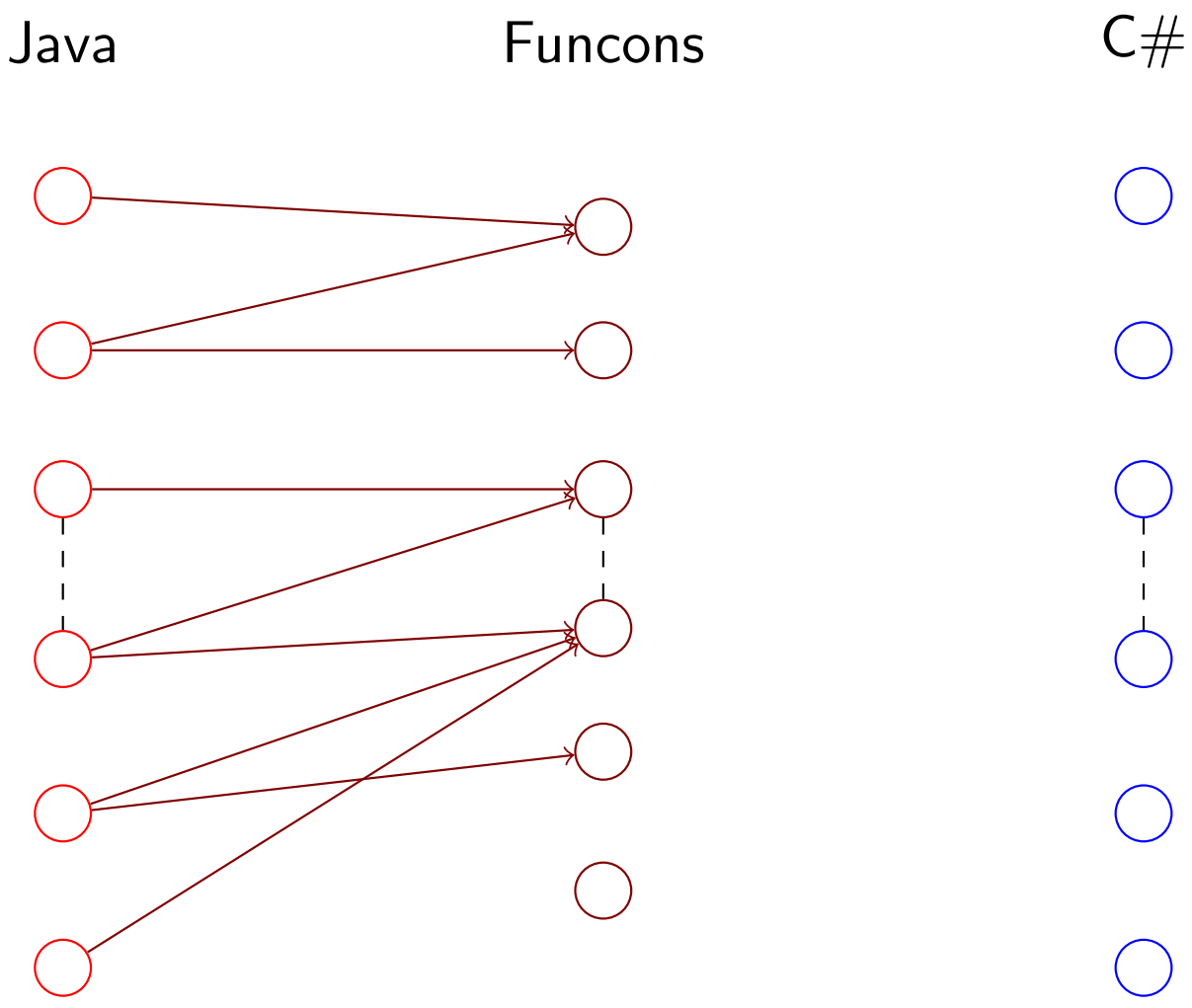
# Reusable Components: Funcons



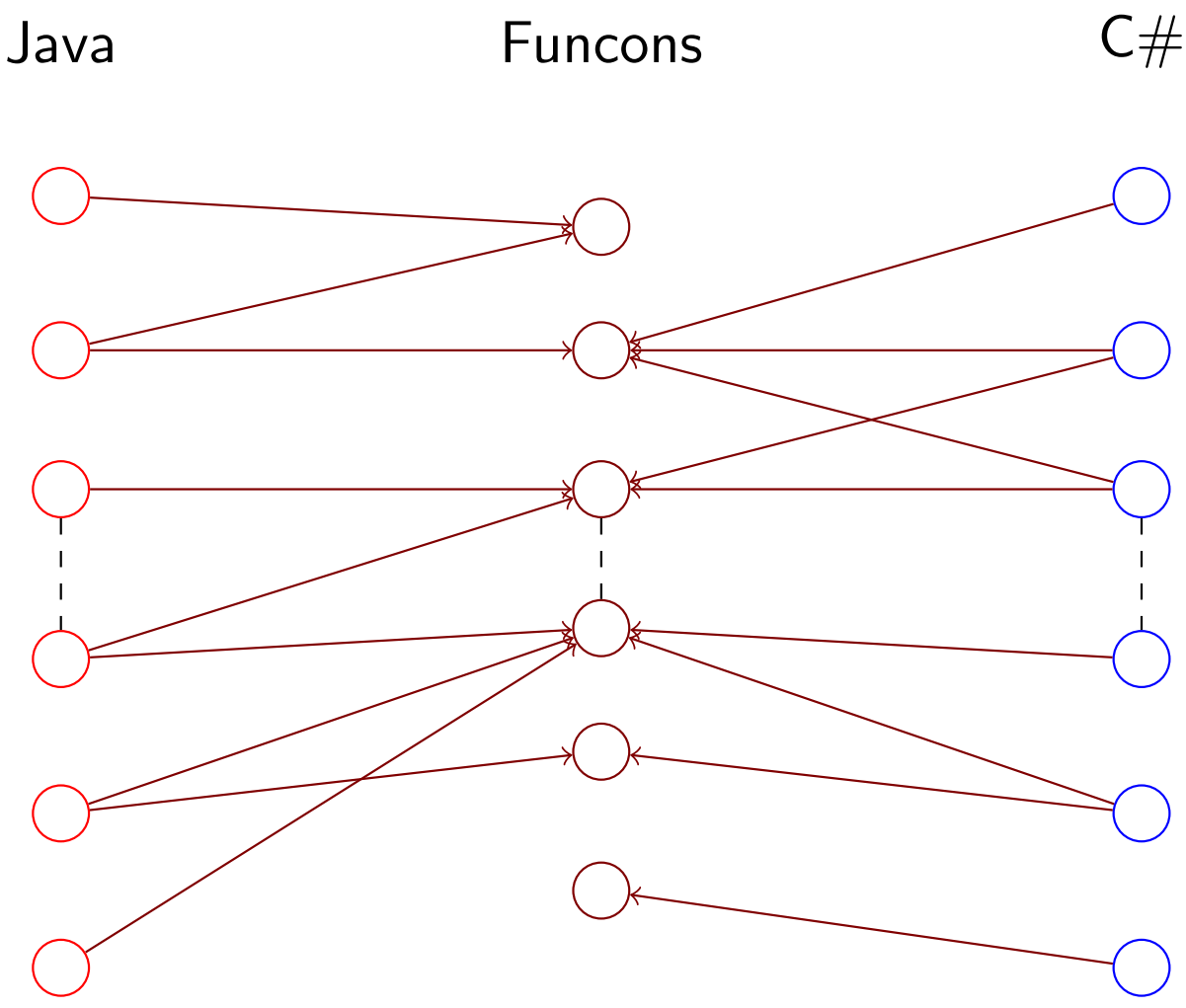
# Reusable Components: Funcons



# Reusable Components: Funcons



# Reusable Components: Funcons



Subsection 1

# Structural Operational Semantics

# SOS Preliminaries

- Terms represent programs.
- A term is either a *value*, or requires *computation*.
- Values are the results of computations.
- The name of *meta-variables*, denoting terms, may indicate whether something is a value of a certain type.
- Typically $F, X, Y, T$ denote arbitrary terms, e.g. $F(X, Y)$.
- $V_1, V_2, \ldots$ denote values.
- $i_1, i_2, \ldots$ denote integers.
- $x_i, x_i, \ldots$ denote identifiers.

# Example Syntax

$$
\begin{aligned}
\mathit{Expr} ::= {} & i \\
\mid {} & x \\
\mid {} & \mathit{add}(\mathit{Expr}, \mathit{Expr})
\end{aligned}
$$

# Big Step SOS

$$\frac{X \Rightarrow i_1 \qquad Y \Rightarrow i_2}{add(X, Y) \Rightarrow i_1 + i_2} \tag{1}$$

$$i \Rightarrow i \tag{2}$$

$$x \Rightarrow ??? \tag{3}$$

# Big Step Semantics with Context

$$\frac{\Gamma \vdash X \Rightarrow i_1 \qquad \Gamma \vdash Y \Rightarrow i_2}{\Gamma \vdash add(X, Y) \Rightarrow i_1 + i_2} \tag{1}$$

$$\Gamma \vdash i \Rightarrow i \tag{2}$$

$$\frac{(x \mapsto V) \in \Gamma}{\Gamma \vdash x \Rightarrow V} \tag{3}$$

- Prove $add(3, add(2, x)) \Rightarrow 6$ assuming $\Gamma = \{x \mapsto 1\}$

# Small Step SOS

$$\frac{X \to X'}{add(X, Y) \to add(X', Y)} \quad (1)$$

$$\frac{Y \to Y'}{add(i_1, Y) \to add(i_1, Y')} \quad (2)$$

$$add(i_1, i_2) \to i_1 + i_2 \quad (3)$$

- We cannot prove: $add(3, add(2, 1)) \to 6$.
- Instead we prove: $add(3, add(2, 1)) \to add(3, 3) \to 6$.
- Or equivalently: $add(3, add(2, 1)) \to^* 6$.

# Mutable Entity

$$\frac{\langle X, \sigma \rangle \rightarrow \langle X', \sigma' \rangle}{\langle add(X, Y), \sigma \rangle \rightarrow \langle add(X', Y), \sigma' \rangle} \quad (1)$$

# Implicitly Modular SOS (I-MSOS)

## Traditional (small-step) SOS

$$\frac{\Gamma \vdash \langle X, \sigma \rangle \to \langle X', \sigma' \rangle}{\Gamma \vdash \langle add(X, Y), \sigma \rangle \to \langle add(X', Y), \sigma' \rangle} \quad (1)$$

## I-MSOS

$$\frac{\mathbf{env}(\Gamma) \vdash \langle X, \mathbf{store}(\sigma) \rangle \to \langle X', \mathbf{store}(\sigma') \rangle}{\mathbf{env}(\Gamma) \vdash \langle add(X, Y), \mathbf{store}(\sigma) \rangle \to \langle add(X', Y), \mathbf{store}(\sigma') \rangle} \quad (2)$$

# Implicitly Modular SOS (I-MSOS)

## Traditional (small-step) SOS

$$\frac{\Gamma \vdash \langle X, \sigma \rangle \to \langle X', \sigma' \rangle}{\Gamma \vdash \langle add(X, Y), \sigma \rangle \to \langle add(X', Y), \sigma' \rangle} \quad (1)$$

## I-MSOS

$$\frac{\mathbf{env}(\Gamma) \vdash X \to X'}{\mathbf{env}(\Gamma) \vdash add(X, Y) \to add(X', Y)} \quad (2)$$

# Implicitly Modular SOS (I-MSOS)

## Traditional (small-step) SOS

$$\frac{\Gamma \vdash \langle X, \sigma \rangle \to \langle X', \sigma' \rangle}{\Gamma \vdash \langle add(X, Y), \sigma \rangle \to \langle add(X', Y), \sigma' \rangle} \quad (1)$$

## I-MSOS

$$\frac{X \to X'}{add(X, Y) \to add(X', Y)} \quad (2)$$

Section 2

# Funcon interpretation

# Interpretation of Funcon Terms

- Funcons are defined with (small-step) SOS.
- The PLanCompS project has identified over 100 funcons.
- Examples:
  - **if-then-else**, **scope**, **assign**, **throw**, **print**
- Context is formed by *semantic entities*:
  - Inherited (like inherited attributes)
  - Mutable (like global mutable variables)
  - Control signals (signalling abnormal behaviour)
  - Output (extra results, e.g. printed values)
  - Input (e.g. input from standard-in or files)

# Interpretation of Funcons Terms (2)

- Complete and successful interpretation *yields* a value.
- Additional effects can be observed in the entities.
- Unsuccessful interpretation yields a stuck term and an optional signal.

# Small Step Interpretation

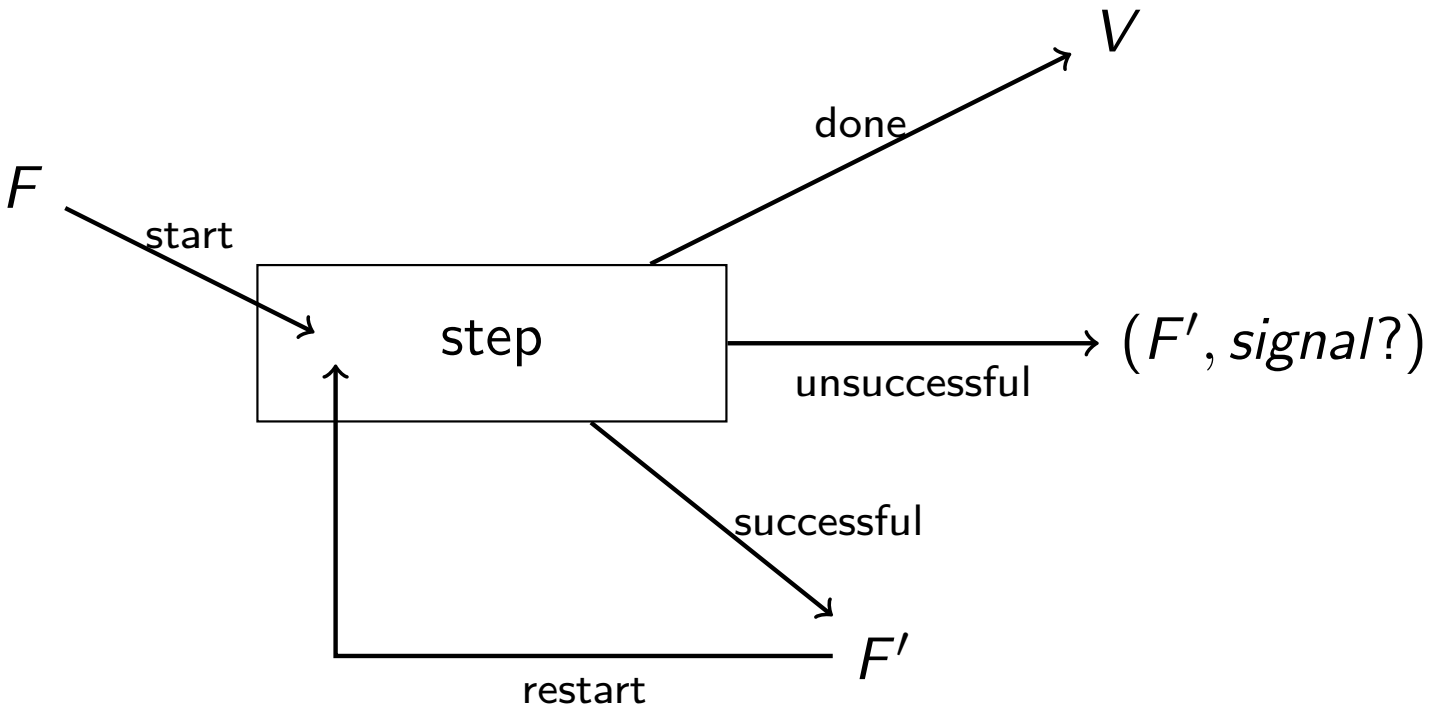



Figure : Diagram of small-step interpretation.

# Informal funcon specification

- The informal specification tells you for each funcon:
  - Whether the interpreter *descends* to some of its children, and in which order.
  - If not descending, by which term the subtree is replaced.
  - Which entities it accesses and whether propagation deviates.
  - The expected *types* of values yielded by the arguments.
  - The type of value the funcon yields itself.
- Enables us to draw diagrams to explain interpretation.
- Remember that we can never descend to a value!

# Pen and Paper Example

- **integer-add**($X$ : *integers*, $Y$ : *integers*) : *integers*
  - Descends $X$ then $Y$.
  - Is replaced by $X + Y$.

Subsection 1

# Mutable Entities

# Mutable Entities

- When descending, a *mutable* entity's value is copied.
- When ascending, a mutable entity's value is copied.
- A mutable entity's value is copied to the next restart.
- Only in the first step uses a default value.
- Examples:
  - **atom-generator** with default 0.
  - **store** with default $\{\}$ (empty map).

# Mutable Entities Example (1)

- Example 1 with **atom-generator**:
  **sequential**(**fresh-atom**, **fresh-atom**)
- **sequential**$(X : T_1, Y : T_2) : T_3$
  - Descends $X$ then $Y$.
  - Replaced by $(X, Y)$, or ...
- **fresh-atom** $() : T$
  - Reads and increments **atom-generator**.
  - Replaced by the new value of **atom-generator**.

Subsection 2

# Inherited Entities

# Inherited Entities

- When descending, an *inherited* entity's value is copied.
- When ascending, inherited entities are undefined
- In a restart, the default value of the entity is used.
- Examples:
  - **given-value** with default ().
  - **environment** with default {} (empty map).

# Inherited Entities Example (1)

- Example 1 with **given-value**
  **give**(3,**integer-add**(2,**given**))
- **give**($X$ : *values*, $Y$ : $T$) : $T$
  - Descends $X$ then $Y$.
  - Sets **given-value** of $Y$ to $X$.
  - Replaced by $Y$.
- **given**() : $T$
  - Reads and is replaced by **given-value**.

Subsection 3

# Output Entities

# Output Entities

- *Output* entities contain 'zero or more' values (in a list).
- When descending, *output* entities are undefined.
- When ascending, an output entity's values are copied.
- At a (re)start there are no values for output entities.
- The output of a sequence of steps ($\rightarrow^*$) is the concatenation of the output at each step.
- Examples: **standard-out**.

# Output Entities Example 1

- Example 1 with **standard-out**
  **sequential**(**print**(1),**print**(2))
- **print**($X : T$) : ()
  - Descends $X$.
  - Replaced by ().
  - Outputs $X$ on the **standard-out**.

Subsection 4

# Control Entities

# Control Entities

- *Control* entities are like output entities, except that:
  - They contain an optional value ('zero or one' values).
  - Values of control entities are called *signals*.
  - If a signal reaches the root of a term there will be no restart. (Unsuccessful interpretation)
  - Signals are *raised*, *caught* and *handled*.
- Examples:
  - **failed**
  - **thrown**

# Control Entities Example

- Example with **throw** and **handle-thrown**:
  **handle-thrown**(**throw**(3),**integer-add**(**given**,2))
- **throw**(X : V) : Stuck
  - Descends $X$.
  - Replaced by **stuck**.
  - Raises $X$ as **thrown** signal.
- **handle-thrown**(X : T, Y : T) : T
  - Descends $X$.
  - Replaced by $X$ if $X$ already a value.
  - Replaced by **give**(V,Y) iff desc. $X$ raises a **thrown** signal $V$.
  - In the latter case the signal is discarded.

# Small Step Interpretation with Entities

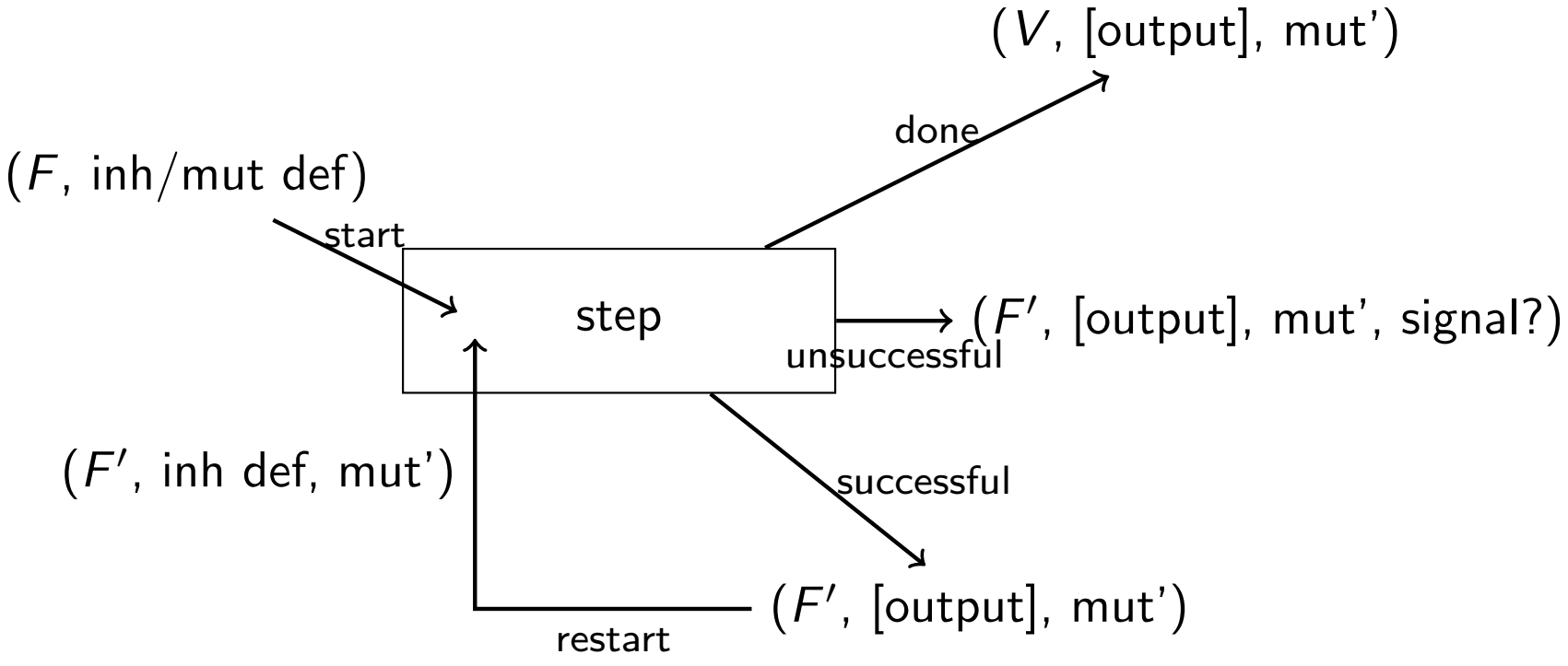



Figure : Diagram of small-step interpretation.