Functional Languages: Formal Description

We will give a precise, formal description of the behaviour of functional programs using *transition systems*.

The language that we consider is a small, first-order, functional programming language working on integers and booleans, called SFUN.

We will define the syntax of SFUN, then give its operational semantics using the Structural Operational Semantics approach.

We can classify functional languages according to the evaluation strategy they implement. There are two main families of functional languages:

- *call-by-value*, and
- *call-by-name.*

First we will give a semantics of SFUN which corresponds to a call-by-value evaluation strategy, then we will show how the semantics has to be modified to model a call-by-name strategy.

Syntax of SFUN

Let $\mathcal{V}$ be a set of variables $\{x, y, z, x_1, x_2, \ldots\}$.
Let $\mathcal{F}$ be a set of functions $\{f_1, \ldots, f_k\}$, each with a fixed arity $ar(f_i) = a_i$.

The *terms* of the language SFUN are defined by the grammar:

$$t ::= n \mid b \mid x \mid t_1\ op\ t_2 \mid t_1\ bop\ t_2 \mid \neg t_1 \mid t_1 \wedge t_2 \mid$$
$$\texttt{if}\ t_0\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \mid f_i(t_1, \ldots, t_{a_i})$$

where

- n represents an integer number (a constant)
- b represents a boolean (True, False)
- $x \in \mathcal{V}$
- $op ::= + \mid - \mid * \mid /$
- $bop ::= \ > \ \mid \ < \ \mid \ =$

The notation $Var(t)$ is used to denote the variables that occur in the term t.
For instance, $Var(x) = \{x\}$, $Var(f(y, z)) = \{y, z\}$.

Closed Terms: $Var(t) = \emptyset$ (no variables).

Program:

A program in SFUN is a set of recursive equations:

$$\begin{aligned} f_1(x_1, \ldots, x_{a_1}) &= t_1 \\ &\vdots \\ f_k(x_1, \ldots, x_{a_k}) &= t_k \end{aligned}$$

such that

- $t_1, \ldots, t_k$ are terms,
- $Var(t_i) \subseteq \{x_1, \ldots, x_{a_i}\}$, $\forall 1 \leq i \leq k$,
- there is only one equation for each f_i.

Since equations are recursive, the terms t_i might contain occurrences of $f_1, \ldots, f_k$.

Example:

$square(x) = x * x$
$fact(x) = \texttt{if } x \leq 0 \texttt{ then } 1 \texttt{ else } x * fact(x - 1)$

Operational Semantics of SFUN

We will give the semantics of SFUN using the Structural Operational Semantics approach. We assume that programs are well-typed.

Let P be a program:

$$\begin{aligned} f_1(x_1, \ldots, x_{a_1}) &= d_1 \\ &\vdots \\ f_k(x_1, \ldots, x_{a_k}) &= d_k \end{aligned}$$

We will define the *evaluation relation* (big-step semantics) for terms in the context of the program P using a *transition system* where *configurations* are just terms.

The evaluation relation (relating closed SFUN terms and values) is denoted by $\Downarrow_P$.

Values are constants (numbers or booleans).

The evaluation strategy that we will model is *call-by-value* (also called *applicative order* of reduction).

The following is an inductive definition (we give a set of axioms and rules) of the evaluation relation $\Downarrow_P$.

Call-by-Value Evaluation. of SFUN

$$\frac{}{n \Downarrow_P n}\,(\mathsf{n}) \qquad \frac{}{b \Downarrow_P b}\,(\mathsf{b})$$

$$\frac{t_1 \Downarrow_P n_1 \quad t_2 \Downarrow_P n_2}{t_1\ op\ t_2 \Downarrow_P n}\,(\mathsf{op})\ \texttt{if}\ n_1\ op\ n_2 = n$$

$$\frac{t_1 \Downarrow_P n_1 \quad t_2 \Downarrow_P n_2}{t_1\ bop\ t_2 \Downarrow_P b}\,(\mathsf{bop})\ \texttt{if}\ n_1\ bop\ n_2 = b$$

$$\frac{t_1 \Downarrow_P b_1 \quad t_2 \Downarrow_P b_2}{t_1\ \wedge\ t_2 \Downarrow_P b}\,(\mathsf{and})\ \texttt{if}\ b_1 \wedge b_2 = b$$

$$\frac{t \Downarrow_P b}{\neg t \Downarrow_P b'}\,(\mathsf{not})\ \texttt{if}\ b' = \neg b$$

$$\frac{t_0 \Downarrow_P True \quad t_1 \Downarrow_P v_1}{\texttt{if}\ t_0\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_P v_1}\,(\mathsf{If_T})$$

$$\frac{t_0 \Downarrow_P False \quad t_2 \Downarrow_P v_2}{\texttt{if}\ t_0\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_P v_2}\,(\mathsf{If_F})$$

$$\frac{t_1 \Downarrow_P v_1 \quad \ldots \quad t_{a_i} \Downarrow_P v_{a_i} \quad d_i\{x_1 \mapsto v_1, \ldots, x_{a_i} \mapsto v_{a_i}\} \Downarrow_P v}{f_i(t_1, \ldots, t_{a_i}) \Downarrow_P v}\,(\mathsf{fn})$$

Example
Call-by-Value

Let P be the program:
$f_1 = f_1 + 1$
$f_2(x) = 1$
$f_3(x) = x * x$

1. With this program, the term $f_2(0)$ has the value 1:
   $f_2(0) \Downarrow_P 1$

   Proof:

$$\dfrac{\dfrac{}{0 \Downarrow_P 0}\,(\mathsf{n}) \qquad \dfrac{}{1\{x \mapsto 0\} \Downarrow_P 1}\,(\mathsf{n})}{f_2(0) \Downarrow_P 1}\,(\mathsf{fn})$$

2. With the same program, the term $f_2(f_1)$ does not have a value:
   there is no v such that $f_2(f_1) \Downarrow_P v$
3. With the same program, $f_3(2+1)$ has the value 9:
   $f_3(2+1) \Downarrow_P 9$

   Proof:

$$\dfrac{\dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n}) \qquad \dfrac{}{1 \Downarrow_P 1}\,(\mathsf{n})}{2+1 \Downarrow_P 3}\,(\mathsf{op}) \qquad \dfrac{\dfrac{}{3 \Downarrow_P 3}\,(\mathsf{n}) \qquad \dfrac{}{3 \Downarrow_P 3}\,(\mathsf{n})}{x * x\{x \mapsto 3\} \Downarrow_P 9}\,(\mathsf{op})}{f_3(2+1) \Downarrow_P 9}\,(\mathsf{fn})$$

Property:

The system is deterministic:

For any term t, $t \Downarrow_P v_1$ and $t \Downarrow_P v_2$ implies $v_1 = v_2$.

Proof: By rule induction.

Base Cases (Axioms):

- If t is a number n then $n \Downarrow_P n$ using the axiom (n), and this is the only axiom or rule that applies.
- If t is a boolean b then $b \Downarrow_P b$ using the axiom (b), and this is the only axiom or rule that applies.

Therefore there is only one value in both cases. There are no more base cases (t is closed).

Induction Step:

We distinguish cases according to the rule that applies to t (for any term, there is only one rule that can be applied).

We will only show the case corresponding to a function application:

Assume t is the term $f(t_1, \ldots, t_{a_i})$. Then, using the rule (fn):

$f(t_1, \ldots, t_{a_i}) \Downarrow_P v$ if and only if

$t_1 \Downarrow_P v_1$, ..., $t_{a_i} \Downarrow_P v_{a_i}$, $d_i\{x_1 \mapsto v_1, \ldots, x_{a_i} \mapsto v_{a_i}\} \Downarrow_P v$

By induction hypothesis, there is at most one value for $t_1, \ldots, t_n$ and $d_i\{x_1 \mapsto v_1, \ldots, x_{a_i} \mapsto v_{a_i}\}$. Therefore v is uniquely determined.

The cases corresponding to the other rules are similar.

Call-by-Name Evaluation of SFUN

In order to model the call-by-name strategy (also called *normal order* of reduction), we need to change the rule that defines the behaviour of application.

We replace it by the following rule:

$$\frac{d_i\{x_1 \mapsto t_1, \ldots, x_{a_i} \mapsto t_{a_i}\} \Downarrow_P v}{f_i(t_1, \ldots, t_{a_i}) \Downarrow_P v} \ (\mathsf{fn_N})$$

The system is still deterministic:

Property:
For any term t: $t \Downarrow_P v_1$ and $t \Downarrow_P v_2$ implies $v_1 = v_2$.

Proof: By rule induction.
Exercise: Complete the details of the proof.

Example
Call-by-name

Let P be the program:
$f_1 = f_1 + 1$
$f_2(x) = 1$
$f_3(x) = x * x$

1. With this program, the term $f_2(0)$ has the value 1:
   $f_2(0) \Downarrow_P 1$
2. With the same program, $f_2(f_1)$ also has the value 1 (compare with the call-by-value semantics):
   $f_2(f_1) \Downarrow_P 1$

   This is proved as follows:

$$\dfrac{\dfrac{}{1\{x \mapsto f_1\} \Downarrow_P 1}\,(\mathsf{n})}{f_2(f_1) \Downarrow_P 1}\,(\mathsf{fn_N})$$

3. With the same program, the term $f_3(2+1)$ has the value 9 (compare with the call-by-value semantics):
   $f_3(2+1) \Downarrow_P 9$

   This is proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n}) \quad \dfrac{}{1 \Downarrow_P 1}\,(\mathsf{n})}{2+1 \Downarrow_P 3}\,(\mathsf{op}) \quad \dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n}) \quad \dfrac{}{1 \Downarrow_P 1}\,(\mathsf{n})}{2+1 \Downarrow_P 3}\,(\mathsf{op})}{x * x\{x \mapsto 2+1\} \Downarrow_P 9}\,(\mathsf{op})}{f_3(2+1) \Downarrow_P 9}\,(\mathsf{fn_N})$$

A Type System for SFUN

The grammar defining the syntax of SFUN allows us to build terms such as $1 + True$ which does not make sense.

In the definition of the semantics of SFUN we only considered *well-typed* terms.

The set of well-typed terms can be defined using a relation:

$$\Gamma \vdash_{\varepsilon} M : \tau$$

where

- Γ is a *variable environment*, or simply *environment*, i.e. a finite partial function from variables to types.
- ε is a *function environment* assigning a type to each function, respecting its arity:
  If $arity(f_i) = a_i$ then $\varepsilon(f_i) = (\sigma_1, \ldots, \sigma_i) \to \sigma$.
- M is a SFUN term.
- τ is a *type*:

$$b ::= \mathsf{int} \mid \mathsf{bool}$$
$$\tau ::= (b_1, \ldots, b_n) \to b$$

The relation $\Gamma \vdash_{\varepsilon} M : \tau$ can be read as:

If the variable x has type $\Gamma(x)$ for each $x \in dom(\Gamma)$ and the functions $f_1, \ldots, f_k$ have types $\varepsilon(f_1), \ldots, \varepsilon(f_k)$ then the term M has type τ.

This is inductively defined by the following system of axioms and rules.

Typing SFUN Terms

Type System Axioms:

$$\frac{}{\Gamma \vdash_\varepsilon b{:}\mathsf{bool}} \qquad \frac{}{\Gamma \vdash_\varepsilon n{:}\mathsf{int}} \qquad \frac{}{\Gamma \vdash_\varepsilon x{:}\sigma}\ \texttt{if}\ \Gamma(x) = \sigma$$

Type System Rules:

$$\frac{\Gamma \vdash_\varepsilon M_1{:}\mathsf{int} \quad \Gamma \vdash_\varepsilon M_2{:}\mathsf{int}}{\Gamma \vdash_\varepsilon M_1\ op\ M_2{:}\mathsf{int}}$$

$$\frac{\Gamma \vdash_\varepsilon M_1{:}\mathsf{int} \quad \Gamma \vdash_\varepsilon M_2{:}\mathsf{int}}{\Gamma \vdash_\varepsilon M_1\ bop\ M_2{:}\mathsf{bool}}$$

$$\frac{\Gamma \vdash_\varepsilon M_1{:}\mathsf{bool} \quad \Gamma \vdash_\varepsilon M_2{:}\mathsf{bool}}{\Gamma \vdash_\varepsilon M_1 \wedge M_2{:}\mathsf{bool}} \qquad \frac{\Gamma \vdash_\varepsilon M{:}\mathsf{bool}}{\Gamma \vdash_\varepsilon \neg M{:}\mathsf{bool}}$$

$$\frac{\Gamma \vdash_\varepsilon B{:}\mathsf{bool} \quad \Gamma \vdash_\varepsilon M_1{:}\sigma \quad \Gamma \vdash_\varepsilon M_2{:}\sigma}{\Gamma \vdash_\varepsilon \texttt{if}\ B\ \texttt{then}\ M_1\ \texttt{else}\ M_2{:}\sigma}$$

$$\frac{\Gamma \vdash_\varepsilon t_1{:}\sigma_1 \quad \cdots \quad \Gamma \vdash_\varepsilon t_{a_i}{:}\sigma_{a_i}}{\Gamma \vdash_\varepsilon f_i(t_1, \ldots, t_{a_i}){:}\sigma}\ \texttt{if}\ \varepsilon(f_i) = (\sigma_1, \ldots, \sigma_{a_i}) \to \sigma$$

Typing SFUN Programs

A program P in SFUN:

$$\begin{aligned} f_1(x_1, \ldots, x_{a_1}) &= t_1 \\ &\vdots \\ f_k(x_1, \ldots, x_{a_k}) &= t_k \end{aligned}$$

is typeable if for each equation $f_i(x_1, \ldots, x_{a_i}) = t_i$, there is a type τ_i and an environment Γ_i such that $\Gamma_i \vdash_\varepsilon f_i(x_1, \ldots, x_{a_i}){:}\tau_i$ and $\Gamma_i \vdash_\varepsilon t_i{:}\tau_i$.

Example:

The program P of the previous examples:

$f_1 = f_1 + 1$
$f_2(x) = 1$
$f_3(x) = x * x$

is typeable in an environment ε where
$\varepsilon(f_1) = \mathsf{int}$
$\varepsilon(f_2) = \mathsf{int} \to \mathsf{int}$
$\varepsilon(f_3) = \mathsf{int} \to \mathsf{int}$

Question:

Is the type system defined for SFUN polymorphic or monomorphic?