<u>Functional Programming – Part 3: Semantics</u>

- We will give a precise, formal description of the behaviour of functional programs.

- 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 five its semantics using the structural operational semantics approach.

EVALUATION STRATEGIES

We can classify functional languages according to the evaluation strategy they implement.

- Call by name is less efficient because it can include repetition when evaluating
- Call by value have a risk of non-termination. If argument is not terminating, then entire function is non-terminating.

- *Call-by-value*: argument expressions are evaluated before applying function definitions.

- *Call*-by-name: function definitions are applied before evaluating argument expressions

First, we will give a semantics which correspond 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.

$$\frac{}{n \Downarrow_P n}\ (\mathrm{n}) \qquad \frac{}{b \Downarrow_P b}\ (\mathrm{b})$$

$$\frac{t_1 \Downarrow_P n_1 \quad t_2 \Downarrow_P n_2}{t_1\ op\ t_2 \Downarrow_P n}\ (\mathrm{op})\ \textit{if}\ n_1\ op\ n_2 = n \qquad \frac{t_1 \Downarrow_P n_1 \quad t_2 \Downarrow_P n_2}{t_1\ bop\ t_2 \Downarrow_P b}\ (\mathrm{bop})\ \textit{if}\ n_1\ bop\ n_2 = b$$

$$\frac{t_1 \Downarrow_P b_1 \quad t_2 \Downarrow_P b_2}{t_1 \wedge t_2 \Downarrow_P b}\ (\mathrm{and})\ \textit{if}\ b_1 \wedge b_2 = b \qquad \frac{t \Downarrow_P b_1}{\neg t \Downarrow_P b}\ (\mathrm{not})\ \textit{if}\ b = \neg b_1$$

$$\frac{t_0 \Downarrow_P \mathit{True} \quad t_1 \Downarrow_P v_1}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \Downarrow_P v_1}\ (\mathrm{if_t}) \qquad \frac{t_0 \Downarrow_P \mathit{False} \quad t_2 \Downarrow_P v_2}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \Downarrow_P v_2}\ (\mathrm{if_f})$$

$$\frac{t_1 \Downarrow_P v_1 \quad \dots \quad t_{(f_i)} \Downarrow_P v_{(f_i)} \quad d_i\{x_1 \mapsto v_1, \dots, x_{(f_i)} \mapsto v_{(f_i)}\} \Downarrow_P v}{f_i(t_1, \dots, t_{(f_i)}) \Downarrow_P v}\ (\mathrm{fn_V})$$

SYNTAX OF SFUN

- **ARITY** – number of arguments taken by a function

Let $\mathcal{V}$ be a set of variables $\{x, y, z, \ldots\}$ and let $\mathcal{F}$ be a set of function names $\{f_1, f_2, \ldots f_i, \ldots\}$, each with a fixed arity $\langle f_i \rangle$.

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

$$
\begin{aligned}
op &::= + \mid - \mid * \mid / \qquad bop ::= \; > \mid < \mid = \\
t &::= n \mid b \mid x \mid t_1 \; op \; t_2 \mid t_1 \; bop \; t_2 \mid \neg t_1 \mid t_1 \wedge t_2 \\
&\mid \text{if } t_0 \text{ then } t_1 \text{ else } t_2 \mid f(t_1, \ldots, t_{\langle f \rangle})
\end{aligned}
$$

- where n represents an integer value, $n \in \mathbb{Z}$
- and b represents a Boolean value, $b \in \{True, False\}$

In the case of the application of a function f such that $\langle f \rangle = 0$, then the empty parentheses may be omitted and the term written as simply f. *Division in* SFUN *is floor division.*

- Remember that SFUN can use only Integer and Boolean so all divisions are floor divisions.
- IN SFUN WE HAVE THE PARENTHESIS AROUND ARGUMENTS < NOT LIKE IN HASKELL

- T :: n = Integer, b = Boolean, x = variable, t1 op t2 = arithmetic, t1 bop t2 = comparison, negation, conjunction, two-way conditional and function application = taking the same number of parameters as ARITY <T of F> shows.

- If function is a constant, we drop the brackets.

VARIABLES

*The notation **vars(t)** is used to denote the variables that occur in the term **t.***

EXAMPLE:

- **Vars(x) = {x}**
- **Vars(f(y, z)) = {y, z} –** if it's a function application with two variables it will be those two variables.

*A **closed term** is a term such that **vars(t)** = ∅*, that is a term that contains no variable

PROGRAMS IN SFUN

- $d_1, \dots, d_k$ are terms,

- A program in **SFUN** is a set of recursive equations:
- Like in Haskell program is a list of equations in SFUN you can only have one equation for each function symbol, so there is no definition by guarder or pattern matching, every function is defined by **single equation**.

$$\begin{aligned} f_1(x_1, \dots, x_{\langle f_1 \rangle}) &= d_1 \\ &\vdots \\ f_k(x_1, \dots, x_{\langle f_k \rangle}) &= d_k \end{aligned}$$

- this defines that all variables in given term **di** must be in subset of all argument variables that are on the left side.

- $vars(d_i) \subseteq \{x_1, \dots, x_{\langle f_i \rangle}\}, \forall i. 1 \leq i \leq k,$

- there is only one equation for each function name f_i.

Equations are recursive, thus the terms d_i may contain occurrences of $f_1, \dots, f_k$.

EXAMPLE

$$\begin{aligned} max(x, y) &= \text{if } x \geq y \text{ then } x \text{ else } y \\ fact(x) &= \text{if } x \leq 0 \text{ then } 1 \text{ else } x * fact(x-1) \end{aligned}$$

$$\begin{aligned} square(x) &= x * x \\ quadratic(x, a, b, c) &= a * square(x) + b * x + c \end{aligned}$$

$$\begin{aligned} mod(x, y) &= \text{if } x - y < 0 \text{ then } x \text{ else } mod(x - y, y) \\ even(x) &= mod(x, 2) = 0 \\ collatz(x) &= \text{if } x = 1 \text{ then } 1 \text{ else if } even(x) \text{ then } x/2 \text{ else } 3 * x + 1 \end{aligned}$$

OPERATIONAL Semantics of SFUN

*We will not give the semantics of SFUN in the structural operational semantics style. We assume that programs are **well-typed***

- *We will define the evaluation relation (**a big-step semantics)** for terms in the context of the program P using a transition system where configurations are just terms. (NO MEMORY STATES)*
- *The values of the system are **integer** and **Boolean** values.*
- *The evaluation relation relates closed **SFUN** terms to values in the context of **P**, and is denoted by* ⇓P.

- *The evaluation strategy that we will model first is a **call-by-value** strategy.*

The term $fortytwo(0)$ has the value 42, that is $fortytwo(0) \Downarrow_P 42$.

$$\dfrac{\dfrac{}{0 \Downarrow_P 0}\,(\mathrm{n}) \qquad \dfrac{}{42\{x \mapsto 0\} \Downarrow_P 42}\,(\mathrm{n})}{fortytwo(0) \Downarrow_P 42}\,(\mathrm{fn_V})$$

CALL-BY-VALUE evaluation of SFUN

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

$$\frac{t_1 \Downarrow_P n_1 \quad t_2 \Downarrow_P n_2}{t_1\ op\ t_2 \Downarrow_P n}\,(\mathsf{op})\ \textit{if}\ n_1\ op\ n_2 = n \qquad \frac{t_1 \Downarrow_P n_1 \quad t_2 \Downarrow_P n_2}{t_1\ bop\ t_2 \Downarrow_P b}\,(\mathsf{bop})\ \textit{if}\ n_1\ bop\ n_2 = b$$

$$\frac{t_1 \Downarrow_P b_1 \quad t_2 \Downarrow_P b_2}{t_1 \wedge t_2 \Downarrow_P b}\,(\mathsf{and})\ \textit{if}\ b_1 \wedge b_2 = b \qquad \frac{t \Downarrow_P b_1}{\neg t \Downarrow_P b}\,(\mathsf{not})\ \textit{if}\ b = \neg b_1$$

$$\frac{t_0 \Downarrow_P \textit{True} \quad t_1 \Downarrow_P v_1}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \Downarrow_P v_1}\,(\mathsf{if_t}) \qquad \frac{t_0 \Downarrow_P \textit{False} \quad t_2 \Downarrow_P v_2}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \Downarrow_P v_2}\,(\mathsf{if_f})$$

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

EXAMPLE

- The term **square(2 + 1)** has the value 9, **square(2 + 1)** ⇓ 9

$$\frac{\dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n}) \quad \dfrac{}{1 \Downarrow_P 1}\,(\mathsf{n})}{2+1 \Downarrow_P 3}\,(\mathsf{op}) \qquad \dfrac{\dfrac{}{3 \Downarrow_P 3}\,(\mathsf{n}) \quad \dfrac{}{3 \Downarrow_P 3}\,(\mathsf{n})}{(x * x)\{x \mapsto 3\} \Downarrow_P 9}\,(\mathsf{op})}{\textit{square}(2+1) \Downarrow_P 9}\,(\mathsf{fn_V})$$

- The term **max(3, square(2))** ⇓ 4

$$\frac{\dfrac{}{3 \Downarrow_P 3}\,(\mathsf{n}) \quad \dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n}) \quad \dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n}) \quad \dfrac{}{2 \Downarrow_P 2}\,(\mathsf{n})}{(x * x)\{x \mapsto 2\} \Downarrow_P 4}\,(\mathsf{op})}{\textit{square}(2) \Downarrow_P 4}\,(\mathsf{fn}) \quad \dfrac{\dfrac{\dfrac{}{3 \Downarrow_P 3}\,(\mathsf{n}) \quad \dfrac{}{4 \Downarrow_P 4}\,(\mathsf{n})}{3 \geq 4 \Downarrow_P \textit{False}}\,(\mathsf{bop}) \quad \dfrac{}{4 \Downarrow_P 4}\,(\mathsf{n})}{\mathsf{if}\ x \geq y\ \mathsf{then}\ x\ \mathsf{else}\ y\{x \mapsto 3,\ y \mapsto 4\} \Downarrow_P 4}\,(\mathsf{if_f})}{\textit{max}(3, \textit{square}(2)) \Downarrow_P 4}\,(\mathsf{fn})$$

Example

The term *fortytwo*(*infinity*) does not have a value, because the evaluation of the argument *infinity* gives no value. A derivation for *infinity* cannot be constructed because it recurses infinitely on the rule ($\mathsf{fn_V}$).

PROOF OF UNICITY OF NORMAL FORMS

If the evaluation of an expression using these axiom and rules terminates, then the value reached is unique.

Theorem
For any closed term t, if $t \Downarrow_P v_1$ and $t \Downarrow_P v_2$, then $v_1 = v_2$.

Proof.
By rule induction.

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

Base cases (axioms):

- If t is a integer n then $n \Downarrow_P n$ using the axiom (n)
- If t is a Boolean b then $b \Downarrow_P b$ using the axiom (b)

Therefore there is only one value in both cases.

There are no more base cases, because t is closed and thus cannot contain variables.

Inductive cases (rules):

- Assume t is the term $f(t_1, \ldots, t_{\langle f \rangle})$.
- Then, using the rule (fn_V), $f_i(t_1, \ldots, t_{\langle f_i \rangle}) \Downarrow_P v$ if and only if $t_1 \Downarrow_P v_1, \ldots, t_{\langle f_i \rangle} \Downarrow_P v_{\langle f_i \rangle}$, and $d_i\{x_1 \mapsto v_1, \ldots, x_{\langle f_i \rangle} \mapsto v_{\langle f_i \rangle}\} \Downarrow_P v$
- By the induction hypothesis, there is at most one value for each term $t_1, \ldots, t_n$ and $d_i\{x_1 \mapsto v_1, \ldots, x_{\langle f_i \rangle} \mapsto v_{\langle f_i \rangle}\}$.
- Therefore v is unique.

The cases corresponding to the other rules are similar.

The rules are deterministic you can only get one answer out applying this rule.

Call-by-name evaluation of SFUN

In order to model the call-by-name strategy we need to change the rule that defines the behaviour of application.

We replace it with the following rule:

$$\frac{d_i\{x_1 \mapsto t_1, \ldots, x_{\langle f_i \rangle} \mapsto t_{\langle f_i \rangle}\} \Downarrow_P v}{f_i(t_1, \ldots, t_{\langle f_i \rangle}) \Downarrow_P v} \ (\text{fn}_N)$$

The reduction system still has unique values.

Theorem
For any closed term t, if $t \Downarrow_P v_1$ and $t \Downarrow_P v_2$ then $v_1 = v_2$.

Proof.
By rule induction. □

CHANGE STRATEGY: CALL-BY NAME

Call by name and call by value have the same rules and axioms, the one is different is the function application.

CALL BY VALUE

$$\frac{t_1 \Downarrow_P v_1 \quad \dots \quad t_{\langle f_i \rangle} \Downarrow_P v_{\langle f_i \rangle} \quad d_i\{x_1 \mapsto v_1, \dots, x_{\langle f_i \rangle} \mapsto v_{\langle f_i \rangle}\} \Downarrow_P v}{f_i(t_1, \dots, t_{\langle f_i \rangle}) \Downarrow_P v}\,(\mathrm{fn_V})$$

CALL BY NAME

$$\frac{d_i\{x_1 \mapsto t_1, \dots, x_{\langle f_i \rangle} \mapsto t_{\langle f_i \rangle}\} \Downarrow_P v}{f_i(t_1, \dots, t_{\langle f_i \rangle}) \Downarrow_P v}\,(\mathrm{fn_N})$$

Recall the program P:

$$\begin{aligned} &\mathit{infinity} = \mathit{infinity} + 1 \\ &\mathit{fortytwo}(x) = 42 \\ &\mathit{square}(x) = x * x \end{aligned}$$

Because we did not evaluate the parameter which is non-terminal, when evaluating by name of function first we disregarded arguments because this function was returning a constant.

Example

Using the call-by-name semantics the term $\mathit{fortytwo}(0)$ also has the value 42, $\mathit{fortytwo}(0) \Downarrow_P 42$.

Example

However in contrast to call-by-value, $\mathit{fortytwo}(\mathit{infinity}) \Downarrow_P 42$, because the argument $\mathit{infinity}$ is discarded without being evaluated.

$$\frac{\dfrac{}{42\{x \mapsto \mathit{infinity}\} \Downarrow_P 42}\,(\mathrm{n})}{\mathit{fortytwo}(\mathit{infinity}) \Downarrow_P 42}\,(\mathrm{fn_N})$$

$$\frac{\dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathrm{n}) \quad \dfrac{}{1 \Downarrow_P 1}\,(\mathrm{n})}{2 + 1 \Downarrow_P 3}\,(\mathrm{op}) \qquad \dfrac{\dfrac{}{3 \Downarrow_P 3}\,(\mathrm{n}) \quad \dfrac{}{3 \Downarrow_P 3}\,(\mathrm{n})}{(x * x)\{x \mapsto 3\} \Downarrow_P 9}\,(\mathrm{op})}{\mathit{square}(2 + 1) \Downarrow_P 9}\,(\mathrm{fn_V})$$

In this case we evaluated 2 + 1 only once because we evaluate left branch to the axioms and then right value, so in this case 2 + 1 was already computed before evaluation of square method. On the other hand, call by Name is repetitive therefore less efficient because it computes 2 + 1 twice.

$$\frac{\dfrac{\dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathrm{n}) \quad \dfrac{}{1 \Downarrow_P 1}\,(\mathrm{n})}{2 + 1 \Downarrow_P 3}\,(\mathrm{op}) \qquad \dfrac{\dfrac{}{2 \Downarrow_P 2}\,(\mathrm{n}) \quad \dfrac{}{1 \Downarrow_P 1}\,(\mathrm{n})}{2 + 1 \Downarrow_P 3}\,(\mathrm{op})}{(x * x)\{x \mapsto 2 + 1\} \Downarrow_P 9}\,(\mathrm{op})}{\mathit{square}(2 + 1) \Downarrow_P 9}\,(\mathrm{fn_N})$$

Remembering to always finish one derivation of subtree before evaluating answer so the tree goes up and then when it evaluated to axiom then it can go down and write output.

TYPES 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 *base types*, β, and the *types* τ of SFUN are defined as follows:

$$\beta ::= \mathsf{int} \mid \mathsf{bool}$$
$$\tau ::= \beta \mid (\beta_1, \ldots, \beta_n) \to \beta$$

In the case of a function type τ such that $n = 0$ (that is, a function that takes no arguments), τ may be written simply as β.

WELL-TYPED TERMS IN SFUN

$$\Gamma \vdash_\varepsilon t : \tau$$

- Γ (GAMMA) is a variable environment, or simply environment, given as a finite partial function from variables to base types.
- ε (EPSILON) is a function environment assigning to each function name a type respecting its arity: that is, if hf i = 0 then ε(f) = β and if hf i = n where n ≥ 1, then ε(f) = (β1, . . . , βn) → β
- t is a SFUN term
- τ is a type

The relation $\Gamma \vdash_\varepsilon t : \tau$ can be read as: "If variables x, y, …… have types Γ(x), Γ(y), …. And functions *f1, f2*, … have types ε*(f1),* ε(f2), … then the term t has type τ .

The well-typedness relation is inductively defined by the following system of axioms and rules. Note that in the case of the application of a function f, if $\langle f \rangle = 0$ then the rule reduces to an axiom, because there are no arguments to check.

$$\frac{}{\Gamma \vdash_\varepsilon b : \mathsf{bool}}\text{(b)} \qquad \frac{}{\Gamma \vdash_\varepsilon n : \mathsf{int}}\text{(n)} \qquad \frac{}{\Gamma \vdash_\varepsilon x : \beta}\text{(var) } \textit{if } \Gamma(x) = \beta$$

$$\frac{\Gamma \vdash_\varepsilon t_1 : \mathsf{int} \quad \Gamma \vdash_\varepsilon t_2 : \mathsf{int}}{\Gamma \vdash_\varepsilon t_1 \; op \; t_2 : \mathsf{int}}\text{(op)} \quad \frac{\Gamma \vdash_\varepsilon t_1 : \mathsf{int} \quad \Gamma \vdash_\varepsilon t_2 : \mathsf{int}}{\Gamma \vdash_\varepsilon t_1 \; bop \; t_2 : \mathsf{bool}}\text{(bop)} \quad \frac{\Gamma \vdash_\varepsilon t : \mathsf{bool}}{\Gamma \vdash_\varepsilon \neg t : \mathsf{bool}}\text{(not)}$$

$$\frac{\Gamma \vdash_\varepsilon t_1 : \mathsf{bool} \quad \Gamma \vdash_\varepsilon t_2 : \mathsf{bool}}{\Gamma \vdash_\varepsilon t_1 \wedge t_2 : \mathsf{bool}}\text{(and)} \qquad \frac{\Gamma \vdash_\varepsilon t_0 : \mathsf{bool} \quad \Gamma \vdash_\varepsilon t_1 : \tau \quad \Gamma \vdash_\varepsilon t_2 : \tau}{\Gamma \vdash_\varepsilon \textsf{if } t_0 \textsf{ then } t_1 \textsf{ else } t_2 : \tau}\text{(if)}$$

$$\frac{\Gamma \vdash_\varepsilon t_1 : \beta_1 \quad \ldots \quad \Gamma \vdash_\varepsilon t_{\langle f \rangle} : \beta_{\langle f \rangle}}{\Gamma \vdash_\varepsilon f(t_1, \ldots, t_{\langle f \rangle}) : \beta}\text{(fn) } \textit{if } \varepsilon(f) = (\beta_1, \ldots, \beta_{\langle f \rangle}) \to \beta$$

EXAMPLE OF TYPING SFUN TERMS

1. GAMMA stores types of variables
2. Evaluate each subtree exhaustively
3. When doing evaluation, we consider only one branch of if conditional, however with TYPING we can’t infer which one will be chosen so, we must evaluate two branches.

Example

Given that $\Gamma(x) = \text{int}$ and $\varepsilon(\mathit{fact}) = (\text{int}) \to \text{int}$ give the derivation tree for $\Gamma \vdash_\varepsilon \text{if } x \leq 0 \text{ then } 1 \text{ else } x * \mathit{fact}(x-1) : \text{int}$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\varepsilon x : \text{int}} \quad \dfrac{}{\Gamma \vdash_\varepsilon 0 : \text{int}}}{\Gamma \vdash_\varepsilon x \leq 0 : \text{bool}} \quad \dfrac{}{\Gamma \vdash_\varepsilon 1 : \text{int}} \quad \dfrac{\dfrac{}{\Gamma \vdash_\varepsilon x : \text{int}} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\varepsilon x : \text{int}} \quad \dfrac{}{\Gamma \vdash_\varepsilon 1 : \text{int}}}{\Gamma \vdash_\varepsilon x - 1 : \text{int}}}{\Gamma \vdash_\varepsilon \mathit{fact}(x-1) : \text{int}}}{\Gamma \vdash_\varepsilon x * \mathit{fact}(x-1) : \text{int}}}{\Gamma \vdash_\varepsilon \text{if } x \leq 0 \text{ then } 1 \text{ else } x * \mathit{fact}(x-1) : \text{int}}$$

$$\dfrac{\dfrac{}{\Gamma_1 \vdash_\varepsilon x : \text{int}}\text{(var)} \quad \dfrac{}{\Gamma_1 \vdash_\varepsilon y : \text{int}}\text{(var)}}{\Gamma_1 \vdash_\varepsilon \mathit{mod}(x, y) : \text{int}}\text{(fn)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash_\varepsilon x : \text{int}} \quad \dfrac{}{\Gamma_1 \vdash_\varepsilon y : \text{int}}}{\Gamma_1 \vdash_\varepsilon x - y : \text{int}} \quad \dfrac{}{\Gamma_1 \vdash_\varepsilon 0 : \text{int}}}{\Gamma_1 \vdash_\varepsilon (x - y) < 0 : \text{bool}} \quad \dfrac{}{\Gamma_1 \vdash_\varepsilon x : \text{int}} \quad \dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash_\varepsilon x : \text{int}} \quad \dfrac{}{\Gamma_1 \vdash_\varepsilon y : \text{int}}}{\Gamma_1 \vdash_\varepsilon x - y : \text{int}} \quad \dfrac{}{\Gamma_1 \vdash_\varepsilon y : \text{int}}}{\Gamma_1 \vdash_\varepsilon \mathit{mod}(x - y, y) : \text{int}}}{\Gamma_1 \vdash_\varepsilon \text{if } (x - y) < 0 \text{ then } x \text{ else } \mathit{mod}(x - y, y) : \text{int}}$$

$$\dfrac{\dfrac{}{\Gamma_2 \vdash_\varepsilon x : \text{int}}\text{(var)}}{\Gamma_2 \vdash_\varepsilon \mathit{even}(x) : \text{bool}}\text{(fn)} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_2 \vdash_\varepsilon x : \text{int}}\text{(var)} \quad \dfrac{}{\Gamma_2 \vdash_\varepsilon 2 : \text{int}}\text{(n)}}{\Gamma_2 \vdash_\varepsilon \mathit{mod}(x, 2) : \text{int}}\text{(fn)} \quad \dfrac{}{\Gamma_2 \vdash_\varepsilon 0 : \text{int}}\text{(n)}}{\Gamma_2 \vdash_\varepsilon \mathit{mod}(x, 2) = 0 : \text{bool}}\text{(bop)}$$

THIS IS HOW WE TYPE TERMS

THIS IS HOW WE TYPE PROGRAM

Given a program P in SFUN

$$\begin{array}{rcl} f_1(x_1,\ldots,x_{\langle f_1\rangle}) & = & t_1 \\ & \vdots & \\ f_k(x_1,\ldots,x_{\langle f_k\rangle}) & = & t_k \end{array}$$

and a function environment ε,
P is typeable, if for each equation $f_i(x_1,\ldots,x_{\langle f_i\rangle}) = t_i$,
there exists an environment Γ_i and a type τ_i, such that

- $\Gamma_i \vdash_\varepsilon f_i(x_1,\ldots,x_{\langle f_i\rangle}) : \tau_i$
- $\Gamma_i \vdash_\varepsilon t_i : \tau_i$

EXAMPLES OF TYPING SFUN PROGRAMS

Example

Given the following program and function environment show that the program is typable.

$$\begin{array}{rcl} \mathit{infinity} & = & \mathit{infinity}+1 \\ \mathit{fortytwo}(x) & = & 42 \\ \mathit{square}(x) & = & x*x \end{array}$$

$$\begin{array}{rcl} \varepsilon(\mathit{infinity}) & = & \mathsf{int} \\ \varepsilon(\mathit{fortytwo}) & = & (\mathsf{int}) \to \mathsf{int} \\ \varepsilon(\mathit{square}) & = & (\mathsf{int}) \to \mathsf{int} \end{array}$$

Both sides of each equation are typeable with type int,
the first in an empty variable environment
and the latter two using an environment Γ, such that $\Gamma(x) = \mathsf{int}$.

Example

Given the following program and function environment show that the program is typable.

$$\begin{array}{rcl} \mathit{mod}(x,y) & = & \text{if } x-y<0 \text{ then } x \text{ else } \mathit{mod}(x-y,y) \\ \mathit{even}(x) & = & \mathit{mod}(x,2)=0 \end{array}$$

$$\begin{array}{rcl} \varepsilon(\mathit{mod}) & = & (\mathsf{int},\mathsf{int}) \to \mathsf{int} \\ \varepsilon(\mathit{even}) & = & (\mathsf{int}) \to \mathsf{bool} \end{array}$$

The two sides of the equation for *mod* both have type int in a variable environment Γ_1, such that $\Gamma_1(x) = \mathsf{int}$ and $\Gamma_1(y) = \mathsf{int}$.
The two sides of the equation for *even* both have type bool using an environment Γ_2 where $\Gamma_2(x) = \mathsf{int}$.

Proving properties of SFUN programs

We can us proof by induction on SFUN programs directly.

$$\mathit{fact}(x) = \text{if } x \le 0 \text{ then } 1 \text{ else } x * \mathit{fact}(x-1)$$

For all natural numbers, n, prove that $\mathit{fact}(n) = n!$

Example

- Base case: $n = 0$
  by the left-hand side of the conditional, $\mathit{fact}(0) = 1 = 0!$
- Induction case: assume $\mathit{fact}(n) = n!$ and prove for $(n+1)$
  by the right-hand side, $\mathit{fact}(n+1) = (n+1) * \mathit{fact}((n+1)-1)$
  which equals $(n+1) * \mathit{fact}(n)$
  and by the inductive hypothesis equals $(n+1) * n! = (n+1)!$