04—Typing and Denotational Semantics of simPL

CS4215: Programming Language Implementation

Martin Henz

February 3, 2012

Generated on Thursday 2 February, 2012, 18:17

Example

Is `x + 3` well-typed?

Type Environments

A *Type environment*, denoted by Γ, keeps track of the type of indentifiers appearing in the expression.

$\Gamma(x)$ returns the type that is known by environment Γ for the identifier x.

Environment Extension

If $\Gamma[x \leftarrow t]\Gamma'$, then Γ' behaves like Γ, except that the type of x is t.

Example

Let $\Gamma = \emptyset$.

$\emptyset[\texttt{AboutPi} \leftarrow \texttt{int}]\Gamma'$

$\Gamma'(\texttt{AboutPi}) = \texttt{int}$

$\Gamma'[\texttt{Square} \leftarrow \texttt{int -> int}]\Gamma''$

$dom(\Gamma'') = \{\texttt{AboutPi}, \texttt{Square}\}$

Typing Relation

The set of well-typed expressions is defined by the ternary *typing relation*, written $\Gamma \vdash E : t$, where Γ is a type environment such that $E \bowtie X$ and $X \subseteq dom(\Gamma)$.

"The expression E has type t, under the assumption that its free identifiers have the types given by Γ."

Examples

- $\Gamma' \vdash \texttt{AboutPi} * 2 : \texttt{int}$
- $\Gamma'' \vdash \texttt{fun}\{\texttt{int -> int}\}\ \texttt{x -> AboutPi * (Square 2) end} : \texttt{int -> int}$

Examples

$\Gamma' \vdash$ `fun {int -> int} x -> AboutPi * (Square 2) end` : `int -> int`

does not hold, because `Square` occurs free in the expression, but the type environment Γ' to the left of the $\vdash$ symbol is not defined for `Square`.

Definition of Typing Relation

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ \textbf{[VarT]}$$

If $\Gamma(x)$ is not defined, then this rule is not applicable. In this case, we say that there is no type for x derivable from the assumptions Γ.

Definition of Typing Relation (cont'd)

$$\frac{}{\Gamma \vdash n : \texttt{int}}\ \textbf{[NumT]}$$

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}}\ \textbf{[TrueT]}$$

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}\ \textbf{[FalseT]}$$

Definition of Typing Relation (cont'd)

For each primitive operation p that takes n arguments of types $t_1, ..., t_n$ and returns a value of type t, we have exactly one rule of the following form.

$$\frac{\Gamma \vdash E_1 : t_1 \qquad \cdots \qquad \Gamma \vdash E_n : t_n}{\Gamma \vdash p[E_1, \ldots, E_n] : t}\ \textbf{[PrimT]}$$

Definition of Typing Relation (cont'd)

p	t_1	t_2	t
+	int	int	int
-	int	int	int
*	int	int	int
/	int	int	int
&	bool	bool	bool
|	bool	bool	bool
\	bool		bool
=	int	int	bool
<	int	int	bool
>	int	int	bool

Definition of Typing Relation (cont'd)

$$\frac{\Gamma \vdash E : \texttt{bool} \qquad \Gamma \vdash E_1 : t \qquad \Gamma \vdash E_2 : t}{\Gamma \vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} : t}$$

Definition of Typing Relation (cont'd)

$$\frac{\Gamma_1[x_1 \leftarrow t_1]\Gamma_2 \ \cdots \Gamma_n[x_n \leftarrow t_n]\Gamma_{n+1} \qquad \Gamma_{n+1} \vdash E : t}{\Gamma_1 \vdash \texttt{fun}\ \{t_1 * \cdots * t_n \texttt{ -> } t\}\ x_1\ \ldots x_n \texttt{ -> } E\ \texttt{end} : t_1 * \cdots * t_n \texttt{ -> } t}$$

Definition of Typing Relation (cont'd)

$$\frac{\Gamma[f \leftarrow t_1 * \cdots * t_n \texttt{ -> } t]\Gamma_1 \quad \Gamma_1[x_1 \leftarrow t_1]\Gamma_2 \ \cdots \Gamma_n[x_n \leftarrow t_n]\Gamma_{n+1} \quad \Gamma_{n+1} \vdash E : t}{\Gamma \vdash \texttt{recfun } f \ \{t_1 * \cdots * t_n \texttt{ -> } t\} \ x_1 \ldots x_n \texttt{ -> } E \texttt{ end} : t_1 * \cdots t_n \texttt{ -> } t}$$

Definition of Typing Relation (cont'd)

$$\frac{\Gamma \vdash E : t_1 * \cdots * t_n \text{ -> } t \qquad \Gamma \vdash E_1 : t_1 \quad \cdots \quad \Gamma \vdash E_n : t_n}{\Gamma \vdash (E\ E_1 \cdots E_n) : t}$$

Well-Typedness

An expression E is well-typed, if there is a type t such that $E : t$.

Example Proof

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} : \texttt{int}} \quad \dfrac{}{\emptyset \vdash \texttt{3} : \texttt{int}}}{\emptyset \vdash \texttt{2*3} : \texttt{int}} \quad \dfrac{}{\emptyset \vdash \texttt{7} : \texttt{int}}}{\emptyset \vdash \texttt{2*3>7} : \texttt{bool}}$$

Example Proof

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \texttt{x} : \texttt{int}} \quad \dfrac{}{\Gamma \vdash \texttt{1} : \texttt{int}}}{\emptyset[\texttt{x} \leftarrow \texttt{int}]\Gamma \vdash \texttt{x+1} : \texttt{int}}}{\dfrac{\emptyset \vdash \texttt{fun } \{\texttt{int -> int}\} \texttt{ x -> x+1 end} : \texttt{int -> int} \quad \dfrac{}{\emptyset \vdash \texttt{2} : \texttt{int}}}{\emptyset \vdash (\texttt{fun } \{\texttt{int -> int}\} \texttt{ x -> x+1 end 2}) : \texttt{int}}}$$

Unique Type

Lemma

For every expression E and every type assignment Γ, there exists at most one type t such that $\Gamma \vdash E : t$.

More Properties of Typing Relation

Lemma

Typing is not affected by "junk" in the type assignment. If $\Gamma \vdash E : t$, *and* $\Gamma \subset \Gamma'$, *then* $\Gamma' \vdash E : t$.

Lemma

Substituting an identifier by an expression of the same type does not affect typing. If $\Gamma[x \leftarrow t']\Gamma'$, $\Gamma' \vdash E : t$, *and* $\Gamma \vdash E' : t'$, *then* $\Gamma \vdash E'' : t$, *where* $E[x \leftarrow E']E''$.

Type Safety

Type safety is a property of a given language with a given static and dynamic semantics. It says that if a program of the languge is well-typed, certain problems are guaranteed not to occur at runtime.

What do we consider as "problems"?

Components of Type Safety

Progress. Well-typed expressions are values or can be further evaluated.

Preservation. Well-typed expressions do not change their type during evaluation.

Definition of Type Safety

A programming language with a given typing relation $\cdots \vdash \cdots : \cdots$ and one-step evaluation $\mapsto$ is called type-safe, if the following two conditions hold:

1. **Preservation.** If E is a well-typed program with respect to $\cdots \vdash \cdots : \cdots$ and $E \mapsto E'$, then E' is also a well-typed program with respect to $\vdash$.
2. **Progress.** If E is a well-typed program, then either E is a value or there exists a program E' such that $E \mapsto E'$.

Preservation in simPL

If for a simPL expression E and some type t holds $E : t$ and if $E \mapsto_{\text{simPL}} E'$, then $E' : t$.

Progress in simPL

Let simPL' be simPL without division.
If for a simPL' expression E holds $E : t$ for some type t, then either E is a value, or there exists an expression E' such that $E \mapsto_{\text{simPL'}} E'$.

Is perfect typing possible?

The type safety of simPL' ensures that evaluation of a well-typed simPL' expression does not get stuck.
Can we say the reverse by claiming that any expression for which the dynamic semantics produces a value is well-typed?

Is perfect typing possible?

Unfortunately, some expressions produce a value although they are not well-typed. Example:

```
if true then 1 else false end
```

Interesting to note: Perfect typing is not possible for languages such as simPL'.

Stepping back

Summary so far

- Typing allows us to focus on well-typed programs
- Well-typed programs "behave well" (progress, preservation)

Outlook

We will focus on well-typed programs and develop a semantics that eliminates many of the efficiency and engineering issues encountered with dynamic semantics.

A Critique of Current Approach

- Contraction relies substitution; mathematically rather complex.
- Primitive operations that are not total functions, such as division, make the evaluation process get stuck. We want a more explicit way of handling runtime errors.
- simPL contains many "unreasonable" programs, which complicates the definition of a dynamic semantics
- Dynamic semantics cannot be extended easily to other language paradigms such as imperative programming.

A Glimpse of Denotational Semantics

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{+[E_1, E_2] \rightarrowtail v_1 + v_2}$$

- syntactic domains
- semantic domains
- semantic functions

Sublanguages of simPL

- simPL0; integer and boolean expressions
- simPL1; add `let` and `if`
- simPL2; add division
- simPL3; add functions
- simPL4; add recursive functions

Syntactic Domain of simPL0

$$\frac{}{n} \qquad \frac{}{\texttt{true}} \qquad \frac{}{\texttt{false}}$$

$$\frac{E_1 \quad E_2}{p[E_1, E_2]}\ p \in \{\texttt{|},\texttt{\&},\texttt{+},\ \texttt{-},\texttt{*},\texttt{=},\texttt{>},\texttt{<}\} \qquad \frac{E}{p[E]}\ p \in \{\texttt{\textbackslash}\}$$

Semantic Domains

Domain name	Definition	Explanation
Bool	$\{true, false\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	**Bool** + **Int**	expressible values

Expressible values **EV** are values that are the result of evaluating an expression.

Semantic Function

The semantic function

$$\cdot \rightarrowtail \cdot : \textbf{simPL0} \rightarrow \textbf{EV}$$

expresses the meaning of elements of **simPL0**, by defining the value of each element.

Semantic Rules

$$\frac{}{\texttt{true} \rightarrowtail \mathit{true}} \qquad \frac{}{\texttt{false} \rightarrowtail \mathit{false}} \qquad \frac{n \rightarrowtail_{\mathbf{N}} i}{n \rightarrowtail i}$$

The function $\rightarrowtail_{\mathbf{N}}$ transforms the simPL0 integer syntax into an element of **Int**.

Semantic Rules (cont'd)

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 + E_2 \rightarrowtail v_1 + v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 - E_2 \rightarrowtail v_1 - v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 * E_2 \rightarrowtail v_1 \cdot v_2}$$

Semantic Rules (cont'd)

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 \& E_2 \rightarrowtail v_1 \wedge v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 \mid E_2 \rightarrowtail v_1 \vee v_2}$$

$$\frac{E \rightarrowtail v}{\backslash E \rightarrowtail \neg v}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 = E_2 \rightarrowtail v_1 \equiv v_2}$$

Semantic Rules (cont'd)

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 > E_2 \rightarrowtail v_1 > v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 < E_2 \rightarrowtail v_1 < v_2}$$

Example

$1 + 2 > 3 \rightarrowtail \mathit{false}$ holds because $1 + 2 \rightarrowtail 3$ and $3 > 3$ is *false*.

simPL1

Add the following to simPL0:

- conditionals,
- identifiers,
- let

Syntactic Domain of simPL1

$$\frac{E \quad E_1 \quad E_2}{\texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end}}$$

$$\frac{}{x}$$

$$\frac{E \quad E_1 \quad \ldots \quad E_n}{\texttt{let } x_1 = E_1 \cdots x_n = E_n \texttt{ in } E \texttt{ end}}$$

Motivation: Semantic Domains of simPL1

We need to introduce environments that allow us to keep track of the binding of identifiers. These environments map identifiers to *denotable values*.

Semantic Domains of simPL1

Semantic domain	Definition	Explanation
Bool	$\{\mathit{true}, \mathit{false}\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	**Bool** + **Int**	expressible values
DV	**Bool** + **Int**	denotable values
Id	alphanumeric string	identifiers
Env	**Id** $\rightsquigarrow$ **DV**	environments

Operations on Environments

We introduce an operation $\Delta[x \leftarrow v]$,
which denotes an environment that works like Δ,
except that $\Delta[x \leftarrow v](x) = v$

Semantic Functions for simPL1

Define $\cdot \rightarrowtail \cdot$ using auxiliary semantic function $\cdot \Vdash \cdot \rightarrowtail \cdot$ that gets an environment as additional argument.

$$\cdot \rightarrowtail \cdot : \textbf{simPL1} \rightarrow \textbf{EV}$$

$$\frac{\emptyset \Vdash E \rightarrowtail v}{E \rightarrowtail v}$$

Auxiliary Semantic Function for simPL1

$$\cdot \Vdash \cdot \rightarrowtail \cdot : \textbf{Env} * \text{simPL1} \rightarrow \textbf{EV}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{}{\Delta \Vdash \texttt{true} \rightarrowtail \mathit{true}} \qquad \frac{}{\Delta \Vdash \texttt{false} \rightarrowtail \mathit{false}}$$

$$\frac{n \rightarrowtail_{\mathbf{N}} i}{\Delta \Vdash n \rightarrowtail i} \qquad \frac{}{\Delta \Vdash x \rightarrowtail \Delta(x)}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 + E_2 \rightarrowtail v_1 + v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 - E_2 \rightarrowtail v_1 - v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 * E_2 \rightarrowtail v_1 \cdot v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 \& E_2 \rightarrowtail v_1 \wedge v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 \mid E_2 \rightarrowtail v_1 \vee v_2}$$

$$\frac{\Delta \Vdash E \rightarrowtail v}{\Delta \Vdash \backslash E \rightarrowtail \neg v}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 = E_2 \rightarrowtail v_1 \equiv v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 > E_2 \rightarrowtail v_1 > v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 < E_2 \rightarrowtail v_1 < v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{true} \quad \Delta \Vdash E_1 \rightarrowtail v_1}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_1}$$

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{false} \quad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta[x_1 \leftarrow v_1]\cdots[x_n \leftarrow v_n] \Vdash E \rightarrowtail v \quad \Delta \Vdash E_1 \rightarrowtail v_1 \quad \cdots \quad \Delta \Vdash E_n \rightarrowtail v_n}{\Delta \Vdash \texttt{let } x_1 = E_1 \cdots x_n = E_n \texttt{ in } E \texttt{ end} \rightarrowtail v}$$

Example

$$\frac{\dfrac{\dfrac{\ldots}{\emptyset[\texttt{AboutPi} \leftarrow 3] \Vdash \texttt{AboutPi + 2} \rightarrowtail 5}}{\emptyset \Vdash \texttt{let AboutPi = 3 in AboutPi + 2 end} \rightarrowtail 5}}{\texttt{let AboutPi = 3 in AboutPi + 2 end} \rightarrowtail 5}$$

Syntactic Domain of simPL2

The language simPL2 adds division to simPL1.

$$\frac{E_1 \quad E_2}{E_1 / E_2}$$

The difficulty lies in the fact that division on integers is a partial function, not being defined for 0 as second argument.

Semantic Domains of simPL2

Domain name	Definition	Explanation
Bool	$\{\mathit{true}, \mathit{false}\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	$\mathbf{Bool} + \mathbf{Int} + \{\perp\}$	expressible values
DV	$\mathbf{Bool} + \mathbf{Int}$	denotable values
Id	alphanumeric string	identifiers
Env	$\mathbf{Id} \rightsquigarrow \mathbf{DV}$	environments

Modify Auxiliary Semantic Function

The semantic function $\cdot \Vdash \cdot \rightarrowtail \cdot$ is modified to take the occurrence of the error value $\perp$ into account.

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail \bot}{\Delta \Vdash E_1 + E_2 \rightarrowtail \bot} \qquad \frac{\Delta \Vdash E_2 \rightarrowtail \bot}{\Delta \Vdash E_1 + E_2 \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 + E_2 \rightarrowtail v_1 + v_2} \text{ if } v_1, v_2 \neq \bot$$

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail \bot}{\Delta \Vdash E_1 / E_2 \rightarrowtail \bot} \qquad \frac{\Delta \Vdash E_2 \rightarrowtail \bot}{\Delta \Vdash E_1 / E_2 \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E_2 \rightarrowtail 0}{\Delta \Vdash E_1 / E_2 \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 / E_2 \rightarrowtail v_1 / v_2} \text{ if } v_1, v_2 \neq \bot \text{ and } v_2 \neq 0$$

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E \rightarrowtail \bot}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{true} \quad \Delta \Vdash E_1 \rightarrowtail v_1}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_1}$$

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{false} \quad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_2}$$

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E_i \rightarrowtail \bot}{\Delta \Vdash \texttt{let } x_1 \texttt{ = } E_1 \cdots x_n \texttt{ = } E_n \texttt{ in } E \texttt{ end} \rightarrowtail \bot} \text{ for } i, 1 \leq i \leq n$$

$$\frac{\Delta[x_1 \leftarrow v_1] \cdots [x_n \leftarrow v_n] \Vdash E \rightarrowtail v \quad \Delta \Vdash E_1 \rightarrowtail v_1 \cdots \Delta \Vdash E_n \rightarrowtail v_n}{\Delta \Vdash \texttt{let } x_1 \texttt{ = } E_1 \cdots x_n \texttt{ = } E_n \texttt{ in } E \texttt{ end} \rightarrowtail v}$$

otw'

Example

For any environment Δ,
$\Delta \Vdash 5+(3/0) \rightarrowtail \perp$, since
$\Delta \Vdash 3/0 \rightarrowtail \perp$.

Syntactic Domain of simPL3

Add (non-recursive) function definition and application.

$$\frac{E}{\texttt{fun}\ \{\ t\ \}\ x_1 \ldots x_n\ \texttt{->}\ E\ \texttt{end}} \qquad \frac{E \quad E_1 \ \cdots \ E_n}{(E\ E_1\ \ldots\ E_n)}$$

Semantic Domains of simPL3

Domain name	Definition	Explanation
Bool	$\{true, false\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	$\mathbf{Bool} + \mathbf{Int} + \{\perp\} + \mathbf{Fun}$	expressible values
DV	$\mathbf{Bool} + \mathbf{Int} + \mathbf{Fun}$	denotable values
Id	alphanumeric string	identifiers
Env	$\mathbf{Id} \rightsquigarrow \mathbf{DV}$	environments
Fun	$\mathbf{DV} * \cdots * \mathbf{DV} \rightsquigarrow \mathbf{EV}$	function values

Auxiliary Semantic Function (cont'd)

$$\Delta \Vdash \texttt{fun}\ \{\ t\ \}\ x_1 \ldots x_n\ \texttt{->}\ E\ \texttt{end} \rightarrowtail f$$

f is function such that
$f(y_1, \ldots, y_n) = v$, where
$\Delta[x_1 \leftarrow y_1] \cdots [x_n \leftarrow y_n] \Vdash E \rightarrowtail v$

Evaluation of Function Application

$$\frac{\Delta \Vdash E \rightarrowtail f \quad \Delta \Vdash E_1 \rightarrowtail v_1 \ \cdots \ \Delta \Vdash E_n \rightarrowtail v_n}{\Delta \Vdash (E \ E_1 \ \ldots \ E_n) \ \rightarrowtail f(v_1, \ldots, v_n)}$$

Syntactic Domain of simPL4

$$\frac{E}{\texttt{recfun}\ f\ \{\ t\ \}\ x_1 \ldots x_n\ \texttt{->}\ E\ \texttt{end}}$$

Discussion simPL4

We would like to add the following rule to our definition of $\rightarrowtail$.

$$\Delta \Vdash \texttt{recfun}\ g\ \{\ t\ \}\ x_1 \ldots x_n\ \texttt{->}\ E\ \texttt{end} \rightarrowtail f$$

f is function such that
$f(y_1, \ldots, y_n) = v$, where
$\Delta[x_1 \leftarrow y_1] \cdots [x_n \leftarrow y_n]$
$[g \leftarrow f] \Vdash E \rightarrowtail v$

Problem

The symbol f occurs on the right hand side of the definition of f. How do we know that such a function f exists? Some expressions have unique solutions for f, others have multiple solutions.
Example:

```
(recfun f {int -> int} x -> (f x) end 0)
```

Theory of fixpoints ...

Overview of Next Lecture

Friday 10/2:

- A Virtual Machine for simPL