Part III: Semantics and type systems of programming languages

Small-step semantics

- abstract model of program execution
- abstract machine:
  - states $s \in S$
  - $s \rightarrow s'$ reduction relation
  - if deterministic, a (partial) function
- calculus: states are language terms $t \in \mathcal{T}$
  - values $v \in Val \subseteq \mathcal{T}$
  - a term t is a normal form if $\nexists t'.t \rightarrow t'$ (shortly $t \not\rightarrow$)

Introductory example: calculus $\mathcal{E}$

boolean and natural expressions

$$
\begin{array}{lcl}
t & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \mid \texttt{succ}\ t \\
 & & \mid \texttt{pred}\ t \mid \texttt{0} \mid \texttt{iszero}\ t \\
v & ::= & \texttt{true} \mid \texttt{false} \mid n \\
n & ::= & \texttt{0} \mid \texttt{succ}\ n
\end{array}
$$

Reduction rules

Inductive definition of $t \to t'$

(IF) $$\frac{t \to t'}{\texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \to \texttt{if}\ t'\ \texttt{then}\ t_1\ \texttt{else}\ t_2}$$

(IFTRUE) $$\frac{}{\texttt{if true then}\ t_1\ \texttt{else}\ t_2 \to t_1}$$

(IFFALSE) $$\frac{}{\texttt{if false then}\ t_1\ \texttt{else}\ t_2 \to t_2}$$

computational rules, congruence (propagation) rules

Reduction rules

(Succ) $$\frac{t \to t'}{\texttt{succ}\ t \to \texttt{succ}\ t'}$$

(Pred) $$\frac{t \to t'}{\texttt{pred}\ t \to \texttt{pred}\ t'}$$

(PredZero) $$\frac{}{\texttt{pred}\ 0 \to 0}$$

(PredSucc) $$\frac{}{\texttt{pred}\ \texttt{succ}\ n \to n}$$

(IsZeroZero) $$\frac{}{\texttt{iszero}\ 0 \to \texttt{true}}$$

(IsZeroSucc) $$\frac{}{\texttt{iszero}\ \texttt{succ}\ n \to \texttt{false}}$$

(IsZero) $$\frac{t \to t'}{\texttt{iszero}\ t \to \texttt{iszero}\ t'}$$

Example of reduction with proof trees

$$(\text{If})\ \dfrac{(\text{IsZero})\ \dfrac{(\text{PredSucc})\ \dfrac{}{\texttt{pred succ}\ 0 \to 0}}{\texttt{iszero pred succ}\ 0 \to \texttt{iszero}\ 0}}{\texttt{if iszero pred succ}\ 0\ \texttt{then}\ 0\ \texttt{else succ}\ 0 \to \texttt{if iszero}\ 0\ \texttt{then}\ 0\ \texttt{else succ}\ 0}$$

$$(\text{If})\ \dfrac{(\text{IsZeroZero})\ \dfrac{}{\texttt{iszero}\ 0 \to \texttt{true}}}{\texttt{if iszero}\ 0\ \texttt{then}\ 0\ \texttt{else succ}\ 0 \to \texttt{if true then}\ 0\ \texttt{else succ}\ 0}$$

Properties of $\mathcal{E}$

- any value is a normal form
  - the converse does not hold: e.g., `succ true`
  - stuck terms are normal forms but not values
- reduction is deterministic, that is, for all t there exists at most one t' s.t. $t \to t'$ (exercise)
- reduction is terminating, that is, any reduction sequence is finite
- hence, any term has a unique normal form

Big-step semantics

Inductive definition of $t \Downarrow v$

(Big-Val) $\dfrac{}{v \Downarrow v}$

(Big-IfTrue) $\dfrac{t \Downarrow \texttt{true} \quad t_1 \Downarrow v}{\texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow v}$

(Big-IfFalse) $\dfrac{t \Downarrow \texttt{false} \quad t_2 \Downarrow v}{\texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow v}$

(Big-Succ) $\dfrac{t \Downarrow n}{\texttt{succ}\ t \Downarrow \texttt{succ}\ n}$

(Big-PredZero) $\dfrac{t \Downarrow 0}{\texttt{pred}\ t \Downarrow 0}$

(Big-PredSucc) $\dfrac{t \Downarrow \texttt{succ}\ n}{\texttt{pred}\ t \Downarrow n}$

(Big-IsZeroZero) $\dfrac{t \Downarrow 0}{\texttt{iszero}\ t \Downarrow \texttt{true}}$

(Big-IsZeroSucc) $\dfrac{t \Downarrow \texttt{succ}\ n}{\texttt{iszero}\ t \Downarrow \texttt{false}}$

Proof of equivalence

$t \Downarrow v \Rightarrow t \to^\star v$

By induction on the definition of $\Downarrow$, that is:

> *for each (meta)rule defining $\Downarrow$, we prove that, if the property holds for the premises, then it holds for the consequence*

(BIG-VAL) Trivially $v \to^\star v$ (in zero steps).

(BIG-IFTRUE) We have to prove that $\texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \to^\star v$.
By inductive hypothesis, $t \to^\star \texttt{true}$. Then, by applying (IF) as many times as the number of steps in $t \to^\star \texttt{true}$, we get:

$$\texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \to^\star \texttt{if}\ \texttt{true}\ \texttt{then}\ t_1\ \texttt{else}\ t_2$$

Now, by applying (IFTRUE), we get

$$\texttt{if}\ \texttt{true}\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \to^\star t_1$$

and we conclude, since by inductive hypothesis $t_1 \to^\star v$.

Proof of equivalence

$t \to^\star v \Rightarrow t \Downarrow v$

By arithmetic induction on the length of the reduction sequence.

$t \to^0 v$ Then t coincides with v, and we get the thesis.

$t \to^{n+1} v$ Then $t \to t' \to^n v$. By inductive hypothesis, $t' \Downarrow v$.

We prove, by induction on the definition of $\to$, that
$t \to t'$ and $t' \Downarrow v$ imply $t \Downarrow v$.

Proof of equivalence

$t \rightarrow t'$ and $t' \Downarrow v$ imply $t \Downarrow v$

(IFTRUE) We have to prove that $t_1 \Downarrow v$ implies
`if true then` t_1 `else` $t_2 \Downarrow v$.
We get the thesis by applying rules (BIG-VAL) and (BIG-IFTRUE).

(IF) We have to prove that `if` t' `then` t_1 `else` $t_2 \Downarrow v$ implies
`if` t `then` t_1 `else` $t_2 \Downarrow v$.
We derived `if` t' `then` t_1 `else` $t_2 \Downarrow v$ by applying (BIG-IFTRUE) or (BIG-IFFALSE). Consider, e.g, the first case.
Then, we know that premises $t' \Downarrow$ `true` and $t_1 \Downarrow v$ hold.
From the first premise and $t \rightarrow t'$, by inductive hypothesis, we get $t \Downarrow$ `true`.
By applying (BIG-IFTRUE) with premises $t \Downarrow$ `true` e $t_1 \Downarrow v$ we get the thesis.

Lambda-calculus

- introduced by Alonzo Church in the 1930s as part of an investigation into the foundations of mathematics
- Turing-complete formalism, can be considered “the smallest programming language”
- hence, studied as paradigmatic model of programming languages, which can all be encoded
- functional languages are more directly based on it

Basic idea

- calculus of functions
- basic constructs: function definition and application
- in function definition, the "name" is not relevant: $f(x) = x + 3$ and $g(x) = x + 3$ define the same function, also sometimes denoted by $x \mapsto x + 3$
- in the lambda-calculus we write $\lambda \mathtt{x}.\mathtt{x} + 3$, or, by using the operators of $\mathcal{E}$:

  $$\lambda x.succ\ succ\ succ\ x$$

- meta-level abbreviation *add3* $= \lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x}$

Application

$(\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ \mathtt{succ\ 0}$

$(\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ \mathtt{succ\ 0} \rightarrow \mathtt{succ\ succ\ succ\ succ\ 0}$

$g = \lambda \mathtt{f}.\mathtt{f}\ (\mathtt{f}\ (\mathtt{succ\ 0}))$

$$\begin{array}{lll} g\ add3 & = & (\lambda \mathtt{f}.\mathtt{f}\ (\mathtt{f\ succ\ 0}))\ \lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x} \\ & \rightarrow & (\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})((\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ \mathtt{succ\ 0}) \\ & \rightarrow & (\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ \mathtt{succ\ succ\ succ\ succ\ 0} \\ & \rightarrow & \mathtt{succ\ succ\ succ\ succ\ succ\ succ\ succ\ 0} \end{array}$$

$double = \lambda \mathtt{f}.\lambda \mathtt{y}.\mathtt{f}\ (\mathtt{f\ y})$

$$\begin{array}{lll} double\ add3\ \mathtt{0} & = & (\lambda \mathtt{f}.\lambda \mathtt{y}.\mathtt{f}\ (\mathtt{f\ y}))(\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\mathtt{0} \\ & \rightarrow & (\lambda \mathtt{y}.(\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ ((\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ \mathtt{y}))\mathtt{0} \\ & \rightarrow & (\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ ((\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ \mathtt{0}) \\ & \rightarrow & (\lambda \mathtt{x}.\mathtt{succ\ succ\ succ\ x})\ (\mathtt{succ\ succ\ succ\ 0}) \\ & \rightarrow & \mathtt{succ\ succ\ succ\ succ\ succ\ succ\ 0} \end{array}$$

Syntax

$$\begin{array}{lll} t & ::= & x \mid \lambda x.t \mid t_1\ t_2 \mid \dots \\ x & ::= & \mathtt{x} \mid \mathtt{y} \mid \mathtt{f} \mid \dots \end{array}$$

e.g., + $\mathcal{E}$

- Conventions
  - $t_1\ t_2\ t_3 = (t_1\ t_2)\ t_3$
  - $\lambda x.t_1\, t_2 = \lambda x.(t_1\ t_2)$
- Binding, bound, free variables

$$\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}\ \mathtt{y}\ \mathtt{z}$$
$$\lambda \mathtt{x}.(\lambda \mathtt{y}.\mathtt{z}\ \mathtt{y})\ \mathtt{y}$$

- Exercise: formally define the set $FV(t)$ of the free variables of t, and $dim(t)$ the dimension of t, and prove that, for all t, $\mid FV(t) \mid \leq dim(t)$

Small step reduction rules

$$v \quad ::= \quad \lambda x.t$$

$$(\text{APP1})\ \frac{t_1 \rightarrow t_1'}{t_1\ t_2 \rightarrow t_1'\ t_2} \qquad (\text{APP2})\ \frac{t_2 \rightarrow t_2'}{v\ t_2 \rightarrow v\ t_2'}$$

$$(\text{APPABS}^{v})\ \frac{}{(\lambda x.t)\ v \rightarrow t[v/x]}$$

Call-by-value strategy

- corresponds to what usually happens in programming languages
- (AppAbs^v) is a restricted version of β-rule:

$$\text{(AppAbs)} \quad \frac{}{(\lambda x.t_1)\ t_2 \to t_1[t_2/x]}$$

- $t_1[t_2/x]$ is the term obtained by replacing all free occurrences of x in t_1 by t_2

Other strategies

- $(\lambda x.t_1)\ t_2$ is a redex
- full-beta reduction (any redex can be reduced in a non-deterministic way)
- normal order (leftmost outermost redex)
- call-by-name (as above, but no reduction inside a lambda-abstraction)

Consider $\mathit{id}\,(\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z})$ with $\mathit{id} = \lambda \mathrm{x}.\mathrm{x}$

1. $\underline{\mathit{id}\,(\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z})}$
2. $\mathit{id}\,(\underline{\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z}})$
3. $\mathit{id}\,(\mathit{id}\,\lambda \mathrm{z}.\underline{\mathit{id}\,\mathrm{z}})$

call-by-value reduction

$\mathit{id}\,(\underline{\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z}}) \to \underline{\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z}} \to \lambda \mathrm{z}.\mathit{id}\,\mathrm{z}$

(another) full-beta-reduction

$\underline{\mathit{id}\,(\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z})} \to \underline{\mathit{id}\,\lambda \mathrm{z}.\mathit{id}\,\mathrm{z}} \to \lambda \mathrm{z}.\underline{\mathit{id}\,\mathrm{z}} \to \lambda \mathrm{z}.\mathrm{z}$

Call-by-value versus call-by-name

- consider $(\lambda x.0)\,t$: evaluation of t is useless, and can even lead to non termination
- consider $(\lambda x.x + x)\,t$: t can be evaluated only once
- Haskell uses an optimized version called call-by-need (the argument is evaluated if needed and only once)
- call-by-value strategy is strict (eager), call-by-name and call-by-need strategies are lazy
- exercise: formalize full-beta-reduction and call-by-name strategies

Which properties hold for the lambda-calculus?

- any value is a normal form
  - the converse does not hold, e.g., x
- the call by value strategy is deterministic, that is, for all t there exists at most one t' s.t. $t \rightarrow t'$ (exercise)
- reduction is non terminating, that is, there are infinite reduction sequences

Big-step semantics

(Big-Lambda) $$\frac{}{\lambda x.t \Downarrow \lambda x.t}$$

(Big-App) $$\frac{t_1 \Downarrow \lambda x.t \quad t_2 \Downarrow v' \quad t[v'/x] \Downarrow v}{t_1\ t_2 \Downarrow v}$$

Type systems

- aim: define a subset of the language terms, the well-typed terms, whose execution cannot get stuck
- this is obtained by classifying terms by different types
- language operators are applied coherently with such types

Introductory example: type system for $\mathcal{E}$

$$T ::= \texttt{Bool} \mid \texttt{Nat}$$

(T-TRUE) $$\frac{}{\texttt{true} : \texttt{Bool}}$$

(T-FALSE) $$\frac{}{\texttt{false} : \texttt{Bool}}$$

(T-IF) $$\frac{t : \texttt{Bool} \quad t_1 : T \quad t_2 : T}{\texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 : T}$$

(T-ZERO) $$\frac{}{\texttt{0} : \texttt{Nat}}$$

(T-SUCC) $$\frac{t : \texttt{Nat}}{\texttt{succ}\ t : \texttt{Nat}}$$

(T-PRED) $$\frac{t : \texttt{Nat}}{\texttt{pred}\ t : \texttt{Nat}}$$

(T-ISZERO) $$\frac{t : \texttt{Nat}}{\texttt{iszero}\ t : \texttt{Bool}}$$

Example of proof tree

$$(\text{T-If})\ \dfrac{(\text{T-IsZero})\ \dfrac{(\text{T-Zero})\ \dfrac{}{0 : \mathtt{Nat}}}{\mathtt{iszero}\ 0 : \mathtt{Bool}} \quad (\text{T-Zero})\ \dfrac{}{0 : \mathtt{Nat}} \quad (\text{T-Pred})\ \dfrac{(\text{T-Zero})\ \dfrac{}{0 : \mathtt{Nat}}}{\mathtt{pred}\ 0 : \mathtt{Nat}}}{\mathtt{if\ iszero\ 0\ then\ 0\ else\ pred\ 0} : \mathtt{Nat}}$$

- these metarules inductively define a relation $t : T$
- we can prove by structural induction that this relation is a partial function, that is,
  each term has at most one type
  not always true, e.g., in languages with subtyping
- the type system gives a conservative ("pessimistic") approximation of the execution, that is:
- well-typed programs do not get stuck, but the converse does not hold, e.g.,

$$\mathtt{if\ true\ then\ 0\ else\ false}$$

Theorem (Soundness)

If $t : T$ and $t \rightarrow^\star t'$, then t' is not stuck (that is, t' is a value or $t' \rightarrow$)

- soundness is usually proved by:

Theorem (Progress)

If $t : T$ then t is not stuck (that is, t is a value or $t \rightarrow$)

Theorem (Subject Reduction)

If $t : T$ and $t \rightarrow t'$ then $t' : T$

- in general the type could be not exactly the same, but, e.g., a subtype

Progress+Subject reduction $\Rightarrow$ Soundness

Proof: By arithmetic induction on the length of the reduction

$t \rightarrow^0 t'$ Then t coincides with t', and the thesis follows from Progress.

$t \rightarrow^{n+1} t'$ Then $t \rightarrow t'' \rightarrow^n t'$. From Subject Reduction we have that $t'' : T$, hence by inductive hypothesis we get the thesis.

Simply-typed lambda-calculus (+ $\mathcal{E}$)

- explicitly typed approach (Church-style):
  - add type annotations when declaring variables

$$
\begin{array}{lll}
t & ::= & x \mid \lambda x : T.t \mid t_1\ t_2 \mid \texttt{true} \mid \texttt{false} \\
 & & \mid \texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \mid \ldots \\
v & ::= & \lambda x : T.t \mid \texttt{true} \mid \texttt{false} \mid \ldots \\
T & ::= & \texttt{Bool} \mid \texttt{Nat} \mid T_1 \to T_2
\end{array}
$$

  - there is an identity function for each type, e.g., $\lambda \mathtt{x} : \mathtt{Bool}.\mathtt{x}$, $\lambda \mathtt{x} : \mathtt{Nat}.\mathtt{x}$, ...
- alternative approach:
  - implicitly typed (Curry-style)

$$
\begin{array}{lll}
t & ::= & x \mid \lambda x.t \mid t_1\ t_2 \mid \texttt{true} \mid \texttt{false} \\
 & & \mid \texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \mid \ldots \\
v & ::= & \lambda x.t \mid \texttt{true} \mid \texttt{false} \mid \ldots \\
T & ::= & \texttt{Bool} \mid \texttt{Nat} \mid T_1 \to T_2 \mid \alpha \mid (\forall \alpha) T
\end{array}
$$

  - polymorphism: only one function $\lambda \mathtt{x}.\mathtt{x}$
  - most general type $(\forall \alpha)\alpha \to \alpha$

- typing relation $\Gamma \vdash t : T$ with Γ type context, needed to type free variables
- Γ is a partial function from variables to types
- $\Gamma[T/x]$ denotes the function which returns T on x, is equal to Γ otherwise

(T-TRUE) $$\frac{}{\Gamma \vdash \texttt{true} : \texttt{Bool}}$$

(T-FALSE) $$\frac{}{\Gamma \vdash \texttt{false} : \texttt{Bool}}$$

(T-IF) $$\frac{\Gamma \vdash t : \texttt{Bool} \quad \Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T}{\Gamma \vdash \texttt{if}\ t\ \texttt{then}\ t_1\ \texttt{else}\ t_2 : T}$$

(T-VAR) $$\frac{}{\Gamma \vdash x : T}\ \Gamma(x) = T$$

(T-ABS) $$\frac{\Gamma[T_1/x] \vdash t : T_2}{\Gamma \vdash \lambda x : T_1.t : T_1 \to T_2}$$

(T-APP) $$\frac{\Gamma \vdash t_1 : T_2 \to T \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1\ t_2 : T}$$

Soundness of the type system with simple types

Theorem (Soundness)

If $t : T$ *and* $t \rightarrow^\star t'$, *then* t' *is not stuck (that is,* t' *is a value or* $t' \rightarrow$*)*

Theorem (Progress)

If $t : T$, *then* t *is not stuck (that is,* t' *is a value or* $t' \rightarrow$*)*

Theorem (Subject reduction)

If $\Gamma \vdash t : T$ *and* $t \rightarrow t'$ *then* $\Gamma \vdash t' : T$.

- progress (and soundness) only holds for closed terms