Our goal is to study programming language features using various semantic techniques. So far we have seen small-step and big-step operational semantics. However, there are other ways to specify meaning, and they can give useful insights that may not be apparent in the operational semantics.

A different way to give semantics is by defining a *translation* from the programming language to another language that is better understood (and typically simpler). This is essentially a process of *compilation*, in which a source language is converted to a target language. Later on we will see that the target language can even be mathematical structures, in which case we refer to the semantics as a *denotational semantics*. A third style of semantics is *axiomatic semantics*, which we will also discuss later in the course.

1 Translation

We map well-formed programs in the original language into items in a *meaning space*. These items may be

- programs in an another language (definitional translation);
- mathematical objects (denotational semantics); an example is taking $\lambda x : \mathsf{int}.\, x$ to $\{(0,0), (1,1), \ldots\}$.

Because they define the meaning of a program, these translations are also known as *meaning functions* or *semantic functions*. We usually denote the semantic function under consideration by $[\![\cdot]\!]$. An object e in the original language is mapped to an object $[\![e]\!]$ in the meaning space under the semantic function. We may occasionally add other decorations to distinguish between different semantic functions, as for example $[\![e]\!]_{\mathrm{cbn}}$ or $\mathcal{C}[\![e]\!]$.

2 Translating CBN λ-Calculus into CBV λ-Calculus

The call-by-name (lazy) λ-calculus was defined with the following reduction rule and evaluation contexts:

$$(\lambda x.\, e_1)\, e_2 \xrightarrow{1} e_1\{e_2/x\} \qquad\qquad E ::= [\cdot] \mid E\, e.$$

The call-by-value (eager) λ-calculus was similarly defined with

$$(\lambda x.\, e)\, v \xrightarrow{1} e\{v/x\} \qquad\qquad E ::= [\cdot] \mid E\, e \mid v\, E.$$

These are fine as operational semantics, but the CBN rules do not adequately capture why CBV is as expressive as CBN. We can see this more clearly by constructing a translation from CBN to CBV. That is, we treat the CBV calculus as the meaning space. This translation exposes some issues that need to be addressed when implementing a lazy language.

To translate from the CBN λ-calculus to the CBV λ-calculus, the key issue is how to make function application lazy in the arguments. CBV evaluation will eagerly evaluate all the argument expressions, so they need to be protected from evaluation. This is accomplished by wrapping the argument in a dummy λ-abstraction to delay its evaluation. Later, when the value of the argument is needed, the abstraction is applied to a dummy argument id to extract the body. The wrapped argument is sometimes called a *thunk*.

Formally, we define the semantic function $[\![\cdot]\!]$ by induction on the structure of the translated expression:

$$[\![x]\!] \triangleq x\ \mathsf{id} \qquad [\![\lambda x.\, e]\!] \triangleq \lambda x.\, [\![e]\!] \qquad [\![e_1\, e_2]\!] \triangleq [\![e_1]\!]\, (\lambda d.\, [\![e_2]\!]),\ d \notin FV([\![e_2]\!]), \tag{1}$$

where $\mathsf{id} = \lambda z.\, z$.

For an example, recall that we defined:

$$\mathsf{true} \triangleq \lambda xy.\, x \qquad \mathsf{false} \triangleq \lambda xy.\, y \qquad \mathsf{if} \triangleq \lambda xyz.\, xyz.$$

The problem with this construction in the CBV λ-calculus is that $\mathsf{if}\, b\, e_1\, e_2$ evaluates both e_1 and e_2, regardless of the truth value of b. The conversion above fixes this problem.

$$\begin{aligned}
[\![\mathsf{true}]\!] &= [\![\lambda xy.\, x]\!] = \lambda xy.\, [\![x]\!] = \lambda xy.\, x\ \mathsf{id} \\
[\![\mathsf{false}]\!] &= [\![\lambda xy.\, y]\!] = \lambda xy.\, [\![y]\!] = \lambda xy.\, y\ \mathsf{id} \\
[\![\mathsf{if}]\!] &= [\![\lambda xyz.\, xyz]\!] = \lambda xyz.\, [\![(xy)z]\!] = \lambda xyz.\, [\![xy]\!]\ (\lambda d.\, [\![z]\!]) \\
&= \lambda xyz.\, [\![x]\!]\ (\lambda d.\, [\![y]\!])\ (\lambda d.\, [\![z]\!]) \\
&= \lambda xyz.\, (x\ \mathsf{id})\ (\lambda d.\, y\ \mathsf{id})\ (\lambda d.\, z\ \mathsf{id}).
\end{aligned}$$

Now, translating $\mathsf{if}\, \mathsf{true}\, e_1\, e_2$ and evaluating under the CBV rules,

$$\begin{aligned}
[\![\mathsf{if}\, \mathsf{true}\, e_1\, e_2]\!] &= [\![\mathsf{if}]\!]\, (\lambda d.\, [\![\mathsf{true}]\!])\, (\lambda d.\, [\![e_1]\!])\, (\lambda d.\, [\![e_2]\!]) \\
&= (\lambda xyz.\, (x\, \mathsf{id})\, (\lambda d.\, y\, \mathsf{id})\, (\lambda d.\, z\, \mathsf{id}))\, (\lambda d.\, [\![\mathsf{true}]\!])\, (\lambda d.\, [\![e_1]\!])\, (\lambda d.\, [\![e_2]\!]) \\
&\xrightarrow{3} ((\lambda d.\, [\![\mathsf{true}]\!])\, \mathsf{id})\, (\lambda d.\, (\lambda d.\, [\![e_1]\!])\, \mathsf{id})\, (\lambda d.\, (\lambda d.\, [\![e_2]\!])\, \mathsf{id}) \\
&\xrightarrow{1} [\![\mathsf{true}]\!]\, (\lambda d.\, (\lambda d.\, [\![e_1]\!])\, \mathsf{id})\, (\lambda d.\, (\lambda d.\, [\![e_2]\!])\, \mathsf{id}) \\
&= (\lambda xy.\, x\, \mathsf{id})\, (\lambda d.\, (\lambda d.\, [\![e_1]\!])\, \mathsf{id})\, (\lambda d.\, (\lambda d.\, [\![e_2]\!])\, \mathsf{id}) \\
&\xrightarrow{2} ((\lambda d.\, (\lambda d.\, [\![e_1]\!])\, \mathsf{id})\, \mathsf{id}) \\
&\xrightarrow{2} [\![e_1]\!],
\end{aligned}$$

and e_2 was never evaluated.

3 Adequacy

Both the CBV and CBN λ-calculus are *deterministic* reduction strategies in the sense that there is at most one reduction that is enabled in any term. When an expression e in a language is evaluated in a deterministic system, one of three things can happen:

1. There exists an infinite sequence of expressions $e_1, e_2, \ldots$ such that $e \xrightarrow{1} e_1 \xrightarrow{1} e_2 \xrightarrow{1} \cdots$. In this case, we write $e \Uparrow$ and say that e *diverges*.
2. The expression e produces a value v in zero or more steps. In this case we say that e *converges* to the value v and write $e \Downarrow v$.
3. The computation converges to a non-value. When this happens, we say the computation is *stuck*.[1]

A semantic translation is *adequate* if these three behaviors in the source system are accurately reflected in the target system, and vice versa. This relationship is illustrated in the following diagram:

$$\begin{array}{ccc}
e & \xrightarrow{\quad * \quad} & v \\
{\scriptstyle [\![\cdot]\!]}\downarrow & & \downarrow{\scriptstyle [\![\cdot]\!]} \\
[\![e]\!] & \xrightarrow{\quad * \quad} & t \approx [\![v]\!]
\end{array}$$

[1] This cannot happen with our CBN-to-CBV translation, but we will see some examples soon enough.

If e converges to a value v in the source language, then $[\![e]\!]$ must converge to some value t that is equivalent (e.g. β-equivalent) to $[\![v]\!]$ in the target language, and vice-versa. This is formally stated as two properties, *soundness* and *completeness*. For our CBN-to-CBV translation, these properties take the following form:

Soundness:

(i) $[\![e]\!]\Downarrow_{\text{cbv}} t \;\Rightarrow\; \exists v\; t \approx [\![v]\!] \wedge e\Downarrow_{\text{cbn}} v$

(ii) $[\![e]\!]\Uparrow_{\text{cbv}} \;\Rightarrow\; e\Uparrow_{\text{cbn}}$

Completeness:

(i) $e\Downarrow_{\text{cbn}} v \;\Rightarrow\; \exists t\; t \approx [\![v]\!] \wedge [\![e]\!]\Downarrow_{\text{cbv}} t$

(ii) $e\Uparrow_{\text{cbn}} \;\Rightarrow\; [\![e]\!]\Uparrow_{\text{cbv}}$

where $\approx$ is some notion of target term equivalence that is preserved by evaluation. Here we are using t to represent target terms to distinguish them from source terms e.

Soundness says that the computation in the CBV domain starting from the image $[\![e]\!]$ of a CBN program e accurately simulates the computation starting from e in the CBN domain. Thus if the target process terminates in a value, then so does the source process from which it was translated, and the final values must be related in the sense described formally above; and if the target computation diverges, then so would the source process. Completeness says the opposite: every computation in the source domain CBN starting from e is accurately simulated by the computation in the target domain CBV starting from $[\![e]\!]$.

Adequacy is the combination of soundness and completeness.

4 Proving Adequacy

We would like to show that evaluation commutes with our translation $[\![\cdot]\!]$ from CBN to CBV. To do this, we first need a notion of target term equivalence ($\approx$) that is preserved by evaluation. This is challenging, because in the evaluation sequence in the target language, intermediate terms may be generated that are not $[\![e]\!]$ for any source term e. For some translations (but not this one), the reverse may also happen. The equivalence must allow for these extra β-redexes that appear during translation.

For our CBN-to-CBV translation, we can define an appropriate equivalence in terms of a reduction relation

$$(\lambda d.\, t)\,\mathsf{id} \xrightarrow[\text{opt}]{1} t, \text{ where } d \notin FV(t) \tag{2}$$

that can be applied in any context, including inside the body of a λ-abstraction. We might call this an *optimization step*. Intuitively, an optimization step unwraps a thunk if it is already applied to the dummy argument id. We write $t_1 \xrightarrow[\text{opt}]{*} t_2$ if t_2 can be obtained from t_1 by applying zero or more optimization steps (2). We define $t_1 \approx t_2$ if t_1 and t_2 reduce to a common normal form via only optimization steps. Ordinarily, to establish that $\approx$ is an equivalence relation, we would need to prove that $\xrightarrow[\text{opt}]{*}$ is confluent. It is indeed confluent, but we do not need to prove it here, because we will only need to use it in one direction. Also note carefully that CBN and CBV themselves do not do optimization steps; this is just a device to define our notion of equivalence.

Adequacy will follow from a series of lemmas, all of which are proved by induction in some form. Most of the work is contained in Lemma 10.5. Let us write $t_1 \xrightarrow[\text{cbv}]{k} t_2$ if t_1 reduces to t_2 in k steps in the CBV reduction order, $t_1 \xrightarrow[\text{cbv}]{*} t_2$ if $t_1 \xrightarrow[\text{cbv}]{k} t_2$ for some $k \geq 0$, and $t_1 \xrightarrow[\text{cbv}]{+} t_2$ if $t_1 \xrightarrow[\text{cbv}]{k} t_2$ for some $k \geq 1$.

To show adequacy, we show that each CBN evaluation step starting from e is mirrored by a sequence of CBV evaluation steps starting from $[\![e]\!]$. To keep track of corresponding stages in the two evaluation sequences, we define a *simulation relation* $\lesssim$ between source and target terms that is more general than the translation $[\![\cdot]\!]$ and is preserved during evaluation of both source and target. Intuitively, $e \lesssim t$ means that CBN term e is simulated by the CBV term t.

Formally, $\lesssim$ is defined by the following rules:

$$x \lesssim x\,\mathsf{id} \qquad \frac{e \lesssim t}{\lambda x.\, e \lesssim \lambda x.\, t} \qquad \frac{e_0 \lesssim t_0 \qquad e_1 \lesssim t_1}{e_0\, e_1 \lesssim t_0\, (\lambda d.\, t_1)} \qquad \frac{e \lesssim t}{e \lesssim (\lambda d.\, t)\,\mathsf{id}} \tag{3}$$

where in the last two rules, the variable d does not occur freely in the body of the abstraction.

The first three rules of (3) ensure that a source term corresponds to its translation. The last rule is different; it takes care of the extra β-reductions that may arise during evaluation. Because the right-hand side of the $\lesssim$ relation becomes structurally smaller in this rule's premise, the definition of the relation is still well-founded. The first three rules are well-founded based on the structure of e; the last is well-founded based on the structure of t. If we were proving a more complex translation correct, we would need more rules like the last rule for other meaning-preserving target-language reductions.

First, let us warm up by showing that a term corresponds to its translation.

Lemma 10.1. $e \lesssim [\![e]\!]$.

Proof. By structural induction on e.

- Case x: $x \lesssim x\,\mathsf{id}$ by definition.
- Case $\lambda x.\, e'$: We have $[\![e]\!] = \lambda x.\, [\![e']\!]$. By the induction hypothesis, $e' \lesssim [\![e']\!]$, so $\lambda x.\, e' \lesssim \lambda x.\, [\![e']\!]$ by the second rule of (3).
- Case $e_0\, e_1$: We have $[\![e]\!] = [\![e_0]\!]\, (\lambda d.\, [\![e_1]\!])$. By the induction hypothesis, $e_0 \lesssim [\![e_0]\!]$ and $e_1 \lesssim [\![e_1]\!]$, so $e_0\, e_1 \lesssim [\![e_0]\!]\, (\lambda d.\, [\![e_1]\!])$ by the third rule of (3). □

Next, let us show that if e is simulated by t, its translation is $\approx$-equivalent to t in a very strong sense.

Lemma 10.2. *If $e \lesssim t$, then $t \xrightarrow[\text{opt}]{*} [\![e]\!]$. (In particular, $t \approx [\![e]\!]$.)*

Proof. Induction on the derivation of $e \lesssim t$.

- Case $x \lesssim x\,\mathsf{id}$:
  We have $x\,\mathsf{id} \xrightarrow[\text{opt}]{0} x\,\mathsf{id} = [\![x]\!]$.
- Case $\lambda x.\, e' \lesssim \lambda x.\, t'$ where $e' \lesssim t'$:
  By the induction hypothesis, $t' \xrightarrow[\text{opt}]{*} [\![e']\!]$, therefore $\lambda x.\, t' \xrightarrow[\text{opt}]{*} \lambda x.\, [\![e']\!] = [\![\lambda x.\, e']\!]$, as $\xrightarrow[\text{opt}]{1}$ reductions in the body of λ-abstractions are permitted.
- Case $e_0\, e_1 \lesssim t_0\, (\lambda d.\, t_1)$ where $e_0 \lesssim t_0$ and $e_1 \lesssim t_1$:
  By the induction hypothesis, $t_0 \xrightarrow[\text{opt}]{*} [\![e_0]\!]$ and $t_1 \xrightarrow[\text{opt}]{*} [\![e_1]\!]$, thus $t_0\, (\lambda d.\, t_1) \xrightarrow[\text{opt}]{*} [\![e_0]\!](\lambda d.\, [\![e_1]\!]) = [\![e_0\, e_1]\!]$.
- Case $e \lesssim (\lambda d.\, t)\,\mathsf{id}$ where $e \lesssim t$:
  By the induction hypothesis, $t \xrightarrow[\text{opt}]{*} [\![e]\!]$. But then $(\lambda d.\, t)\,\mathsf{id} \xrightarrow[\text{opt}]{1} t \xrightarrow[\text{opt}]{*} [\![e]\!]$. □

The next lemma says that if a value $\lambda x.\, e$ corresponds to a term t, then we can always reduce t to a value $\lambda x.\, t'$ while preserving the correspondence with $\lambda x.\, e$.

Lemma 10.3. *If $\lambda x.\, e \lesssim t$, then there exists t' such that $t \xrightarrow[\text{cbv}]{*} \lambda x.\, t'$ and $e \lesssim t'$ (thus also $\lambda x.\, e \lesssim \lambda x.\, t'$).*

Proof. By induction on the derivation of $\lambda x.\, e \lesssim t$.

- Case $y \lesssim y\,\mathsf{id}$: Impossible, as $y \neq \lambda x.\, e$.
- Case $e_0\, e_1 \lesssim t_0\, (\lambda d.\, t_1)$: Impossible, as $e_0\, e_1 \neq \lambda x.\, e$.
- Case $\lambda x.\, e \lesssim \lambda x.\, t'$ where $e \lesssim t'$:
  Immediate, as the right-hand side is already reduced.
- Case $e_0 \lesssim (\lambda d.\, t_0)\,\mathsf{id}$, where $e_0 \lesssim t_0$:
  In this case $e_0 = \lambda x.\, e$ and $t = (\lambda d.\, t_0)\,\mathsf{id}$. By the induction hypothesis, there exists t' such that $e \lesssim t'$ and $t = (\lambda d.\, t_0)\,\mathsf{id} \xrightarrow[\text{cbv}]{1} t_0 \xrightarrow[\text{cbv}]{*} \lambda x.\, t'$. □

The next lemma deals with substitution.

Lemma 10.4. *If $e_1 \lesssim t_1$ and $e_2 \lesssim t_2$, then $e_1\{e_2/x\} \lesssim t_1\{\lambda d.\, t_2/x\}$.*

Proof. We proceed by induction on the derivation of $e_1 \lesssim t_1$.

- Case $x \lesssim x\,\mathsf{id}$:
  By the fourth rule of (3) with premise $e_2 \lesssim t_2$,
  $$x\{e_2/x\} = e_2 \lesssim (\lambda d.\, t_2)\,\mathsf{id} = (x\,\mathsf{id})\{\lambda d.\, t_2/x\}.$$
- Cases $y \lesssim y\,\mathsf{id}$ where $y \neq x$ and $\lambda x.\, e \lesssim \lambda x.\, t$:
  These cases are trivial, as the substitutions have no effect.
- Case $\lambda y.\, e \lesssim \lambda y.\, t$ where $e \lesssim t$, $x \neq y$:
  By α-converting if necessary, we can assume without loss of generality that $y \notin FV(e_2) \cup FV(t_2)$. Since $e \lesssim t$, by the induction hypothesis we have $e\{e_2/x\} \lesssim t\{\lambda d.\, t_2/x\}$. Using this as the premise in the second rule of (3), we have
  $$(\lambda y.\, e)\{e_2/x\} = \lambda y.\, (e\{e_2/x\}) \lesssim \lambda y.\, (t\{\lambda d.\, t_2/x\}) = (\lambda y.\, t)\{\lambda d.\, t_2/x\}.$$
- Case $e\, e' \lesssim t\, (\lambda d'.\, t')$, where $e \lesssim t$ and $e' \lesssim t'$:
  By the induction hypothesis, $e\{e_2/x\} \lesssim t\{\lambda d.\, t_2/x\}$ and $e'\{e_2/x\} \lesssim t'\{\lambda d.\, t_2/x\}$. By the third rule of (3),
  $$\begin{aligned}(e\, e')\{e_2/x\} &= (e\{e_2/x\})\,(e'\{e_2/x\}) \\ &\lesssim (t\{\lambda d.\, t_2/x\})\,(\lambda d'.\, (t'\{\lambda d.\, t_2/x\})) \\ &= (t\{\lambda d.\, t_2/x\})\,((\lambda d'.\, t')\{\lambda d.\, t_2/x\}) \\ &= (t\,(\lambda d'.\, t'))\{\lambda d.\, t_2/x\}.\end{aligned}$$
- Case $e_1 \lesssim (\lambda d'.\, t')\,\mathsf{id}$, where $e_1 \lesssim t'$:
  By the induction hypothesis, $e_1\{e_2/x\} \lesssim t'\{\lambda d.\, t_2/x\}$. By the fourth rule of (3),
  $$e_1\{e_2/x\} \lesssim= (\lambda d'.\, (t'\{\lambda d.\, t_2/x\}))\,\mathsf{id} = ((\lambda d'.\, t')\,\mathsf{id})\{\lambda d.\, t_2/x\}.$$

□

The following lemma shows that the relation $\lesssim$ is preserved under evaluation of the source and target.

Lemma 10.5. *If $e \lesssim t$ and $e \xrightarrow[\text{cbn}]{1} e'$, then there exists t' such that $t \xrightarrow[\text{cbv}]{+} t'$ and $e' \lesssim t'$.*

$$\begin{array}{ccc} e & \xrightarrow[\text{cbn}]{1} & e' \\ \lesssim \downarrow & & \downarrow \lesssim \\ t & \xrightarrow[\text{cbv}]{+} & t' \end{array}$$

Proof. We proceed by induction on the derivation of $e \lesssim t$.

- Case $x \lesssim x\,\mathsf{id}$: This is vacuously true, as there is no evaluation step possible from x.
- Case $\lambda x.\, e \lesssim \lambda x.\, t$: As $\lambda x.\, e$ is a value, this is also vacuously true.
- Case $e_0\, e_1 \lesssim t_0\, (\lambda d.\, t_1)$, where $e_0 \lesssim t_0$ and $e_1 \lesssim t_1$:
  There are two subcases, depending on the form of the derivation of $e \xrightarrow[\text{cbn}]{1} e'$.
  - Subcase $e_0\, e_1 \xrightarrow[\text{cbn}]{1} e_0'\, e_1$, where $e_0 \xrightarrow[\text{cbn}]{1} e_0'$:
    By the induction hypothesis, there exists t_0' such that $e_0' \lesssim t_0'$ and $t_0 \xrightarrow[\text{cbv}]{+} t_0'$. Then $t_0\, (\lambda d.\, t_1) \xrightarrow[\text{cbv}]{+} t_0'\, (\lambda d.\, t_1)$, and by the third rule of (3), $e_0'\, e_1 \lesssim t_0'\, (\lambda d.\, t_1)$.
  - Subcase $(\lambda x.\, e')\, e_1 \xrightarrow[\text{cbn}]{1} e'\,\{e_1/x\}$:
    In this case, $\lambda x.\, e' \lesssim t_0$ and $e_1 \lesssim t_1$.
    By Lemma 10.3, there exists t' such that $t_0 \xrightarrow[\text{cbv}]{*} \lambda x.\, t'$ and $e' \lesssim t'$. We thus have $t_0\, (\lambda d.\, t_1) \xrightarrow[\text{cbv}]{*} (\lambda x.\, t')\, (\lambda d.\, t_1) \xrightarrow[\text{cbv}]{1} t'\,\{\lambda d.\, t_1/x\}$, and by Lemma 10.4, $e'\,\{e_1/x\} \lesssim t'\,\{\lambda d.\, t_1/x\}$.
- Case $e_0 \lesssim (\lambda d.\, t_0)\,\mathsf{id}$, where $e_0 \lesssim t_0$:
  By the induction hypothesis, there exists t_0' such that $e_0 \lesssim t_0'$ and $t_0 \xrightarrow[\text{cbv}]{*} t_0'$. Then $(\lambda d.\, t_0)\,\mathsf{id} \xrightarrow[\text{cbv}]{1} t_0 \xrightarrow[\text{cbv}]{*} t_0'$. □

We are now ready to prove the adequacy of the translation.

Theorem 10.6. *The CBN-to-CBV translation (1) is sound and complete.*

Proof. First completeness. Given a source term e and its translation $[\![e]\!]$, from Lemma 10.1 we have that $e \lesssim [\![e]\!]$. From Lemma 10.5, we have that each step of the CBN evaluation of e is mirrored by a CBV execution on the target side that preserves $e \lesssim t$. Thus if the evaluation of e diverges, so will the evaluation of $[\![e]\!]$. On the other hand, if the evaluation of e converges to a value v, then by Lemma 10.3, the evaluation of $[\![e]\!]$ will converge to a value t such that $v \lesssim t$. By Lemma 10.2, $[\![v]\!] \approx t$. This establishes completeness.

For soundness, we need to show that every evaluation in the target language corresponds to some evaluation in the source language. Suppose we have a target-language evaluation $[\![e]\!] \to t$ to a value t. There are three possibilities for the evaluation of e. First, the evaluation could get stuck. This cannot happen for this source language, because all terms are either values or have a legal evaluation step. Second, e could evaluate to a value v. But then $v \lesssim t$ by Lemma 10.5, because the target-language evaluation is deterministic. Third, the evaluation of e might diverge. But then Lemma 10.5 says there is a divergent target-language evaluation. The determinism of the target language ensures that this cannot happen. □