Abstract Machines and Classical Realizability

Paul Downen

June 24–27, 2022

Contents

Chapter 1

Abstract Machines

1.1 A Tale of Two Semantics

Direct semantics of source language (operational semantics, denotational semantics, etc.):

- Assign a *meaning* to a piece of *source code* which indicates its *value* (what does it return when evaluated?) or *behavior* (what other effects happen when it is run?)
- Can reason directly about expressions of the source language itself $\ddot\smile$
- Makes it easy to reason about high-level properties of code at compile time (“are these two functions equal?”, “is this program type-safe?”) $\ddot\smile$
- Makes it harder to understand the low-level cost of programs at run time (“how many instructions does this loop call take to run?”, “how many allocations does this function call make?”) $\ddot\frown$

Abstract machine semantics:

- Describe a form of *theoretical* low-level machine that abstracts away details from, while remaining close enough to, real-world machines
- To run a source program on the machine, it may or may not have to be compiled to a different machine language first
- Describes a more practical implementation by being more similar to the real machine, providing both a formal specification and a hint of how to implement $\ddot\smile$
- Makes it easy to reason about low-level cost of programs at run time $\ddot\smile$
- Makes it harder to reason about high-level properties of code at compile time $\ddot\frown$

> LISP programmers know the value of everything and the cost of nothing.
>
> C programmers know the cost of everything and the value of nothing.

How can we have our cake and eat it, too?

1. Enrich the direct semantics with a notion of *cost* (see Jan Hoffman, OPLSS 2016; Foundations of Programming Languages, OPLSS 2018)
2. Discover the good high-level properties of a well-designed abstract machine

1.2 Source Language

A small simply-typed λ-calculus with booleans as the only base type.

1.2.1 Syntax & Semantics

Syntax:

$$\begin{aligned} M, N ::= {} & x \mid M\ N \mid \lambda x.M \\ & \mid \mathrm{True} \mid \mathrm{False} \mid \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2 \end{aligned}$$

(Call-by-Name) Operational Semantics:

$$\begin{aligned} (\lambda x.M)\ N &\mapsto M[N/x] && (\beta_{\to}) \\ \mathbf{if}\,\mathrm{True}\,\mathbf{then}\,M_1\,\mathbf{else}\,M_2 &\mapsto M_1 && (\beta_{\mathrm{Bool}\,1}) \\ \mathbf{if}\,\mathrm{False}\,\mathbf{then}\,M_1\,\mathbf{else}\,M_2 &\mapsto M_2 && (\beta_{\mathrm{Bool}\,2}) \end{aligned}$$

1.2.2 Safety

Type System:

$$\begin{gathered} A, B ::= \mathrm{Bool} \mid A \to B \\ \Gamma ::= \bullet \mid \Gamma, x : A \end{gathered}$$

$$\frac{}{\Gamma, x : A \vdash x : A}\ Var$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B} \to E \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \to I$$

$$\frac{}{\Gamma \vdash \mathrm{True} : \mathrm{Bool}}\ \mathrm{Bool}I_1 \qquad \frac{}{\Gamma \vdash \mathrm{False} : \mathrm{Bool}}\ \mathrm{Bool}I_2$$

$$\frac{\Gamma \vdash N : \mathrm{Bool} \quad \Gamma \vdash M_1 : A \quad \Gamma \vdash M_2 : A}{\Gamma \vdash \mathbf{if}\, N\, \mathbf{then}\, M_1\, \mathbf{else}\, M_2 : A}\ \mathrm{Bool}E$$

Lemma 1.2.1 (Progress)**.** *If* $\bullet \vdash M : \mathrm{Bool}$ *then either* M *is a value (*True *or* False*) or there is an* M' *such that* $M \mapsto M'$*.*

Lemma 1.2.2 (Preservation)**.** *If* $\Gamma \vdash M : A$ *and* $M \mapsto M'$ *then* $\Gamma \vdash M' : A$*.*

Corollary 1.2.3 (Type Safety)**.** *If* $\bullet \vdash M : \mathrm{Bool}$ *then every time* $M \mapsto\!\!\!\!\to M' \not\mapsto$*,* M' *has to be a valid final value (*True *or* False*).*

Is this true?
No! Consider $\textbf{if}\,(\lambda x.x)\,\mathrm{False}\,\textbf{then}\,\mathrm{False}\,\textbf{else}\,\mathrm{True} \not\mapsto$.
We forgot reduction inside of *evaluation contexts* (E).

$$E ::= \Box \mid E\ N \mid \textbf{if}\,E\,\textbf{then}\,M_1\,\textbf{else}\,M_2$$

$$\frac{M \mapsto M'}{E[M] \mapsto E[M']}$$

Now we have

$$\begin{array}{ll} \textbf{if}\,\boxed{(\lambda x.x)\ \ \mathrm{False}}\,\textbf{then}\,\mathrm{False}\,\textbf{else}\,\mathrm{True} & \\ \mapsto \boxed{\textbf{if}\,\mathrm{False}\,\textbf{then}\,\mathrm{False}\,\textbf{else}\,\mathrm{True}} & (\beta_{\to}) \\ \mapsto \mathrm{True} & (\beta_{\mathrm{Bool}\,2}) \end{array}$$

Many steps (*i.e.,* the *reflexive*, *transitive* closure) of $\mapsto$ is written as $\mapsto\!\!\!\!\to$:

$$\frac{M \mapsto M'}{M \mapsto\!\!\!\!\to M'}\ \textit{Inclusion} \qquad \frac{}{M \mapsto\!\!\!\!\to M}\ \textit{Reflexivity} \qquad \frac{M \mapsto\!\!\!\!\to M' \quad M' \mapsto\!\!\!\!\to M''}{M \mapsto\!\!\!\!\to M''}\ \textit{Transitivity}$$

1.2.3 Theories of computation

Reduction theory, where $\to$ is "reduction anywhere" and $\twoheadrightarrow$ is zero or more steps of $\to$:

$$\frac{M \mapsto M'}{M \to M'}\ \textit{Inclusion} \qquad \frac{M \to M'}{C[M] \to C[M']}\ \textit{Compatibility}$$

$$\frac{M \to M'}{M \twoheadrightarrow M'}\ \textit{Inclusion} \qquad \frac{}{M \twoheadrightarrow M}\ \textit{Reflexivity} \qquad \frac{M \twoheadrightarrow M' \quad M' \twoheadrightarrow M''}{M \twoheadrightarrow M''}\ \textit{Transitivity}$$

where C can be *any* context.

Exercise 1.2.4. Let

$$\mathit{and} = \lambda x.\lambda y.\,\textbf{if}\,x\,\textbf{then}\,y\,\textbf{else}\,\mathrm{False}$$

Use the reduction theory of the λ-calculus to prove that $\lambda y.(\mathit{and}\,\mathrm{True}\,y) \twoheadrightarrow \lambda y.y$.

Equational theory:

$$\frac{M \mapsto M'}{M = M'}\ \textit{Inclusion} \qquad \frac{M = M'}{C[M] = C[M']}\ \textit{Compatibility}$$

$$\frac{}{M = M}\ \textit{Refl.} \qquad \frac{M = M' \quad M' = M''}{M = M''}\ \textit{Trans.} \qquad \frac{M = M'}{M' = M}\ \textit{Symmetry}$$

More axioms (syntactic rules we assume relate equal terms):

$$\begin{array}{rll}
\lambda x.(M\ x) = M : A \to B & (\text{if } x \notin FV(M)) & (\eta_{\to}) \\
\textbf{if}\, M \,\textbf{then}\, \text{True} \,\textbf{else}\, \text{False} = M : \text{Bool} & & (\eta_{\text{Bool}}) \\
E[\textbf{if}\, M \,\textbf{then}\, N_1 \,\textbf{else}\, N_2] = \textbf{if}\, M \,\textbf{then}\, E[N_1] \,\textbf{else}\, E[N_2] & & (\mu_{\text{Bool}})
\end{array}$$

Exercise 1.2.5. Let

$$not = \lambda x.\, \textbf{if}\, x \,\textbf{then}\, \text{False} \,\textbf{else}\, \text{True}$$

Use the equational theory to prove that $\lambda x.\ not\ (not\ x) = \lambda x.x$.

1.3 Target Machine

1.3.1 Naïve Syntax & Semantics

$$\begin{aligned}
M &::= \text{same as before...} \\
E &::= \alpha \mid M \cdot E \mid \textbf{if then}\, M \,\textbf{else}\, M'; E \\
c &::= \langle M \| E \rangle
\end{aligned}$$

Refocusing rules:

$$\begin{array}{rcll}
\langle M\ N \| E \rangle & \mapsto & \langle M \| N \cdot E \rangle & (\mu_{\to}) \\
\langle \textbf{if}\, M \,\textbf{then}\, N_1 \,\textbf{else}\, N_2 \| E \rangle & \mapsto & \langle M \| \textbf{if then}\, N_1 \,\textbf{else}\, N_1; E \rangle & (\mu_{\text{Bool}})
\end{array}$$

Reduction rules:

$$\begin{array}{rcll}
\langle \lambda x.M \| N \cdot E \rangle & \mapsto & \langle M[N/x] \| E \rangle & (\beta_{\to}) \\
\langle \text{True} \| \textbf{if then}\, N_1 \,\textbf{else}\, N_2; E \rangle & \mapsto & \langle N_1 \| E \rangle & (\beta_{\text{Bool}\,1}) \\
\langle \text{False} \| \textbf{if then}\, N_1 \,\textbf{else}\, N_2; E \rangle & \mapsto & \langle N_2 \| E \rangle & (\beta_{\text{Bool}\,2})
\end{array}$$

But now only *commands* can reduce, not terms. Since there is *only one* command in a program — the one at the top-level — there is no opportunity to simplify sub-expression as before.

1.3.2 Compilation and simplified syntax — commands everywhere!

Idea: Make more things commands by compiling away refocusing rules ahead of time.

$$\begin{aligned}\langle M\ N \| \alpha\rangle &\mapsto \langle M \| N \cdot \alpha\rangle \\ M\ N &:= \mu\alpha.\langle M \| N \cdot \alpha\rangle\end{aligned}$$

$$\begin{aligned}\langle \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2 \| \alpha\rangle &\mapsto \langle M \| \mathbf{if\,then}\langle N_1 \| \alpha\rangle\, \mathbf{else}\langle N_2 \| \alpha\rangle\rangle \\ \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2 &:= \mu\alpha.\langle M \| \mathbf{if\,then}\, N_1\, \mathbf{else}\, N_2; E\rangle \\ &:= \mu\alpha.\langle M \| \mathbf{if\,then}\langle N_1 \| \alpha\rangle\, \mathbf{else}\langle N_2 \| \alpha\rangle\rangle\end{aligned}$$

Note: I have now also made the branches of the **if then else** continuation into commands, to put together the next step with the answer of each branch.

Revised syntax of compiled programs into the machine language:

$$\begin{aligned}v &::= x \mid \lambda x.v \mid \mathrm{True} \mid \mathrm{False} \mid \mu\alpha.c \\ E &::= \alpha \mid v \cdot E \mid \mathbf{if\,then}\, c\, \mathbf{else}\, c' \\ c &::= \langle v \| E\rangle\end{aligned}$$

Have just one μ rule for pushing around the continuation through (one or more) elimination steps:

$$\langle \mu\alpha.c \| E\rangle \mapsto c[E/\alpha] \qquad (\mu)$$

For example, application is compiled and then run as:

$$\begin{aligned}\langle M\ N \| E\rangle &:= \langle \mu\alpha.\langle M \| N \cdot \alpha\rangle \| E\rangle \\ &\mapsto \langle M \| N \cdot E\rangle \qquad (\mu)\end{aligned}$$

The only other reduction rules are:

$$\begin{aligned}\langle \lambda x.M \| N \cdot E\rangle &\mapsto \langle M[N/x] \| E\rangle & (\beta_{\to}) \\ \langle \mathrm{True} \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2\rangle &\mapsto c_1 & (\beta_{\mathrm{Bool}\,1}) \\ \langle \mathrm{False} \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2\rangle &\mapsto c_2 & (\beta_{\mathrm{Bool}\,2})\end{aligned}$$

Exercise 1.3.1. Write a compilation transformation function $[\![M]\!]$,

$$[\![\_]\!] : \lambda\text{-term} \to \text{machine term}$$

which converts terms from the λ-calculus (in section 1.2) to a term v of the machine language defined just above.

Hint: here are a few cases to get you going:

$$\begin{aligned}[\![x]\!] &:= x \\ [\![\lambda x.M]\!] &:= \lambda x.[\![M]\!] \\ [\![M\ N]\!] &:= \mu\alpha.\langle [\![M]\!] \| [\![N]\!] \cdot \alpha\rangle\end{aligned}$$

Fill in the definitions of $[\![\mathrm{True}]\!]$, $[\![\mathrm{False}]\!]$, and $[\![\mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2]\!]$.

Exercise 1.3.2. Translate the λ-calculus *and* and *not* functions to the abstract machine, and show that compilation produces the following machine terms

$$\begin{aligned} \mathit{and} &:= \lambda x.\lambda y.\mu\alpha.\langle x \| \mathbf{if\,then}\langle y\|\alpha\rangle\,\mathbf{else}\langle \mathrm{False}\|\alpha\rangle\rangle \\ \mathit{not} &:= \lambda x.\mu\alpha.\langle x \| \mathbf{if\,then}\langle \mathrm{False}\|\alpha\rangle\,\mathbf{else}\langle \mathrm{True}\|\alpha\rangle\rangle \end{aligned}$$

1.3.3 Theories of computation

Reduction theory:

$$\frac{c \mapsto c'}{c \to c'}\;\mathit{Incl.} \qquad \frac{c \to c'}{C[c] \to C[c']}\;\mathit{Compat.}$$

$$\frac{c \to c'}{c \twoheadrightarrow c'}\;\mathit{Incl.} \qquad \frac{}{c \twoheadrightarrow c}\;\mathit{Refl.} \qquad \frac{c \twoheadrightarrow c' \quad c' \twoheadrightarrow c''}{c \twoheadrightarrow c''}\;\mathit{Trans.}$$

and similar for v and E, where C can be *any* context (the *Compat.* rule assumes that the specific C has a command-shaped hole, but when filled C might build a command, a term, or a continuation).

Exercise 1.3.3. Use the reduction theory of the abstract machine to prove that $[\![\lambda y.(\mathit{and}\,\mathrm{True}\,y)]\!] := \lambda y.\mu\alpha.\langle \mathit{and} \| \mathrm{True} \cdot y \cdot \alpha\rangle \twoheadrightarrow \lambda y.\mu\alpha.\langle y\|\alpha\rangle$.

Equational theory:

$$\frac{c \mapsto c'}{c = c'}\;\mathit{Incl.} \qquad \frac{c = c'}{C[c] = C[c']}\;\mathit{Compat.}$$

$$\frac{}{c = c}\;\mathit{Refl.} \qquad \frac{c = c' \quad c' = c''}{c = c''}\;\mathit{Trans.} \qquad \frac{c = c'}{c' = c}\;\mathit{Symm.}$$

and similar reflexivity, symmetry, and transitive rules for v and E. Plus these additional extensionality rules:

$$\begin{aligned} \mu\alpha.\langle v\|\alpha\rangle &= v && (\text{if } \alpha \notin \mathrm{FV}(v)) && (\eta_\mu) \\ \lambda x.\langle v\| x \cdot \alpha\rangle &= v : A \to B && (\text{if } \alpha, x \notin \mathrm{FV}(v)) && (\eta_\to) \\ \mathbf{if\,then}\langle \mathrm{True}\|E\rangle\,\mathbf{else}\langle \mathrm{False}\|E\rangle &= E : \mathrm{Bool} && && (\eta_{\mathrm{Bool}}) \end{aligned}$$

Exercise 1.3.4. Use the equational theory of the abstract machine to prove that $[\![\lambda x.\ \mathit{not}\ (\mathit{not}\ x)]\!] := \lambda x.\mu\alpha.\langle \mathit{not}\|\mu\beta.\langle \mathit{not}\| x \cdot \beta\rangle \cdot \alpha\rangle = \lambda x.x$.

1.3.4 Type safety

Environments:

- Environment $\Gamma = x_1 : A_1, \dots, x_n : A_n$ assigning types to *variables* which stand for unknown terms/values
- Environment $\Delta = \alpha_1 : A_1, \dots, \alpha_n : A_n$ assigning types to *covariables* (*i.e., continuation variables* or *logical duals of variables*)

Three judgements:

- $\Gamma \vdash v : A \mid \Delta$
- $\Gamma \mid E : A \vdash \Delta$
- $c : (\Gamma \vdash \Delta)$

Type system:

$$\frac{\Gamma, x : A \vdash v : B \mid \Delta}{\Gamma \vdash \lambda x.v : A \to B \mid \Delta} \to R \qquad \frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid E : B \vdash \Delta}{\Gamma \mid v \cdot E : A \to B \vdash \Delta} \to L$$

$$\frac{}{\Gamma \vdash \mathrm{True} : \mathrm{Bool} \mid \Delta} \mathrm{Bool}R_1 \qquad \frac{}{\Gamma \vdash \mathrm{False} : \mathrm{Bool} \mid \Delta} \mathrm{Bool}R_2$$

$$\frac{c_1 : (\Gamma \vdash \Delta) \quad c_2 : (\Gamma \vdash \Delta)}{\Gamma \mid \mathbf{if\,then}\, c_1 \,\mathbf{else}\, c_2 : \mathrm{Bool} \vdash \Delta} \mathrm{Bool}L$$

$$\frac{}{\Gamma, x : A \vdash x : A \mid \Delta} Ax \qquad \frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Delta} CoAx$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta} ActR \qquad \frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid E : A \vdash \Delta}{\langle v \| E \rangle : (\Gamma \vdash \Delta)} Cut$$

Lemma 1.3.5 (Progress). *If $c : (\bullet \vdash \alpha : \mathrm{Bool})$ then either c is a final command (of the form $\langle \mathrm{True} \| \alpha \rangle$ or $\langle \mathrm{False} \| \alpha \rangle$) or there is an c' such that $c \mapsto c'$.*

Lemma 1.3.6 (Preservation). *If $c : (\Gamma \vdash \Delta)$ and $c \mapsto c'$ then $c : (\Gamma \vdash \Delta)$.*

Corollary 1.3.7 (Type Safety). *If $c : (\bullet \vdash \alpha : \mathrm{Bool})$ then every time $c \mapsto\!\!\!\!\twoheadrightarrow c' \not\mapsto$, c' has to be a valid final command (of the form $\langle \mathrm{True} \| \alpha \rangle$ or $\langle \mathrm{False} \| \alpha \rangle$).*

1.4 Glossary of arrows

Type of relationship	One Step	Many Step
(Deterministic) Evaluation (only in eval. contexts)	$\mapsto$	$\mapsto\!\!\!\!\twoheadrightarrow$
Reduction Anywhere (in any context)	$\to$	$\twoheadrightarrow$
Forward & Backward (symmetry)		$=$

The hierarchy of these relations is

$$c \mapsto c' \text{ implies } c \mapsto\!\!\!\!\twoheadrightarrow c' \text{ implies } c \twoheadrightarrow c' \text{ implies } c = c'$$
$$c \mapsto c' \text{ implies } c \to c' \text{ implies } c \twoheadrightarrow c' \text{ implies } c = c'$$

Remember that each step down the hierarchy might add new rules. The arrow $c \to c'$ might include extra reduction rules that weren't needed for the operational arrow $c \mapsto c'$ (we didn't have any, but it can happen in practice when you want to add new optimizations besides just simplification). Likewise, the equality relation $c = c'$ might include extra axioms that state two things are equal that goes above and beyond the operational rules for $c \mapsto c'$ and reduction rules for $c \to c'$ (in our case, we had η axioms which formalize certain notions of extensionality equality).

Sometimes in the literature, the many step (*i.e.,* reflexive, transitive) closure of a generic relation R is written uniformly with a star R^* rather than the specialized double arrow head. In that notation, the many-step $\mapsto\!\!\!\!\twoheadrightarrow$ is the $*$-closure $\mapsto^*$ and the many-step $\twoheadrightarrow$ is $\to^*$.

Chapter 2

Classical Realizability

Let's erase the expressions (v, E, and c) from the type system of the abstract machine:

- $\Gamma \vdash v : A \mid \Delta$ becomes $\Gamma \vdash A, \Delta$
- $\Gamma \mid E : A \vdash \Delta$ becomes $\Gamma, A \vdash \Delta$
- $c : (\Gamma \vdash \Delta)$ becomes $\Gamma \vdash \Delta$

Typing rules become logical rules:

$$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta} \to R \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta} \to L$$

$$\frac{}{\Gamma \vdash \mathrm{Bool}, \Delta} \mathrm{Bool}R_1 \quad \frac{}{\Gamma \vdash \mathrm{Bool}, \Delta} \mathrm{Bool}R_2$$

$$\frac{\Gamma \vdash \Delta \quad \Gamma \vdash \Delta}{\Gamma, \mathrm{Bool} \vdash \Delta} \mathrm{Bool}L$$

$$\frac{}{\Gamma, A \vdash A, \Delta} Ax \qquad \frac{}{\Gamma, A \vdash A, \Delta} CoAx$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A, \Delta} ActR \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta} Cut$$

These (ignoring Bool and $ActR$) are the rules of classical logic! Specifically, a system based on Gentzen's sequent calculus LK.[1]

2.1 Realizers

Definition 2.1.1 (Realizer)**.** A *realizer* to a proposition is an *algorithm* (a program, expression, etc. in a computational language) whose type corresponds

[1] I'm taking some liberties with the treatment of environments. If you're a linear logician you may care a lot, and see me after class. Otherwise, it just simplifies away issues we won't be talking about here.

to that proposition.

For example, the implication $A \Rightarrow B$ corresponds to the function type $A \to B$.

2.1.1 Intuitionistic realizers

Intuition: the effect-free expressions (no recursion, state, exceptions, *etc.*) in your favorite (pure) functional language are *realizers* for *intuitionistic logic.*

Examples:

$$A \land B \Rightarrow B \land A$$

Interpret conjunction $A \land B$ as a tuple/pair type $A * B$,

$$\begin{aligned}&\mathbf{data}\, A * B\, \mathbf{where}\\ &\quad (\_,\_) : A \to B \to A * B\end{aligned}$$

$$\begin{aligned}&\mathit{swap} && : A * B \to B * A\\ &\mathit{swap}\ (x, y) &&= (y, x)\end{aligned}$$

$$A \lor B \Rightarrow B \lor A$$

Interpret disjunction $A \lor B$ as a sum type $A + B$,

$$\begin{aligned}&\mathbf{data}\, A + B\, \mathbf{where}\\ &\quad \mathrm{Left} : A \to A + B\\ &\quad \mathrm{Right} : B \to A + B\end{aligned}$$

$$\begin{aligned}&\mathit{flip} && : A + B \to B + A\\ &\mathit{flip}\ (\mathrm{Left}\ x) &&= \mathrm{Right}\, x\\ &\mathit{flip}\ (\mathrm{Right}\ y) &&= \mathrm{Left}\, y\end{aligned}$$

Exercise 2.1.2. Define realizers for these (intuitionistic) tautologies (where $A \iff B$ means to give realizers for both $A \Rightarrow B$ and $B \Rightarrow A$):

1. $A \land (B \land C) \iff (A \land B) \land C$
2. $A \lor (B \lor C) \iff (A \lor B) \lor C$
3. $\top \land A \iff A$
4. $\bot \lor A \iff A$
5. $\top \lor A \iff \top$
6. $\bot \land A \iff \bot$
7. $A \land (B \lor C) \iff (A \land B) \lor (A \land C)$

8. $A \lor (B \land C) \iff (A \lor B) \land (A \lor C)$

Hint: in some cases, there may be multiple valid answers. You should interpret the logical constant $\top$ of truth as the data type 1 with one constructor:

$$\begin{array}{l}\textbf{data}\,1\,\textbf{where}\\ \quad () : 1\end{array}$$

2.1.2 Classical realizers

$$\text{Contrapositive} = (A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A)$$

Logicians like to say that $\neg A$ is the same thing as $A \Rightarrow \bot$ (where propositional constant $\bot$ stands for "false"). Interpret $\bot$ as the empty data type 0,

$$\begin{array}{l}\textbf{data}\,0\,\textbf{where}\\ \qquad \textit{— no constructors}\end{array}$$

so that $\neg A$ is interpreted as $A \to 0$.

$$\begin{array}{ll}\mathit{contra} & : (A \to B) \to ((B \to 0) \to (A \to 0))\\ \mathit{contra}\ f & = \lambda g{:}(B \to 0).\ \lambda x{:}A. g\ (f\ x)\end{array}$$

$$\text{Double Negation Introduction} = A \Rightarrow \neg\neg A$$

$$\begin{array}{ll}\mathit{dni} & : A \to ((A \to 0) \to 0)\\ \mathit{dni}\ x & = \lambda k{:}(A \to 0).\ k\ x\end{array}$$

$$\text{Triple Negation Introduction/Elimination} = \neg\neg\neg A \iff \neg A$$

$$\begin{array}{ll}\mathit{tni} & : (A \to 0) \to (((A \to 0) \to 0) \to 0)\\ \mathit{tni} & = \mathit{dni} \qquad \text{— instantiated to argument type } A \to 0\end{array}$$

$$\begin{array}{ll}\mathit{tne} & : (((A \to 0) \to 0) \to 0) \to (A \to 0)\\ \mathit{tne} & = \mathit{contra}\ \mathit{dni}\end{array}$$

$$\text{Double Negation Elimination} = \neg\neg A \Rightarrow A$$

$$\begin{aligned} \mathit{dne} \quad &: ((A \to 0) \to 0) \to A \\ \mathit{dne}\ h &= \ldots ??? \end{aligned}$$

Try again, in the machine language:

$$\begin{aligned} \mathit{dne} \quad &: ((A \to 0) \to 0) \to A \\ \mathit{dne}\ h &= \mu\alpha{:}A.\langle h \| (\lambda x{:}A.\mu\beta{:}0.\langle x \| \alpha\rangle) \cdot \mathbf{case\,of}\{\}\rangle \end{aligned}$$

where the continuation $\mathbf{case\,of}\{\}$ does a case-analysis on an expected input of type 0 (which is impossible), and since there are no cases to cover, it doesn't say what to do because it represents dead code. The typing rule is:

$$\frac{}{\Gamma \mid \mathbf{case\,of}\{\} : 0 \vdash \Delta}\ 0L$$

and it corresponds to the empty case expression $\mathbf{case}\,M\,\mathbf{of}\{\}$ when $M : 0$ in a pure functional language (like Haskell) pushed onto the call stack like so:

$$\frac{\Gamma \vdash M : 0}{\Gamma \vdash \mathbf{case}\,M\,\mathbf{of}\{\} : A}\ 0E \qquad\qquad \langle \mathbf{case}\,M\,\mathbf{of}\{\} \| E\rangle \mapsto \langle M \| \mathbf{case\,of}\{\}\rangle$$

$$\text{Law of the Excluded Middle} = \neg A \vee A$$

$$\begin{aligned} \mathit{lem} \ &: (A \to 0) + A \\ \mathit{lem} \ &= \mu\alpha{:}(A + (A \to 0)).\langle \mathrm{Left}(\lambda x{:}A.\mu\beta{:}0.\langle \mathrm{Right}\,x \| \alpha\rangle) \| \alpha\rangle \end{aligned}$$

Or written as an equation of machine commands:[2]

$$\langle \mathit{lem} \| \alpha\rangle = \langle \mathrm{Right}(\lambda x{:}A.\langle \mathrm{Left}\,x \| \alpha\rangle) \| \alpha\rangle$$

[2] Notice how the definition of *lem* is definitely *not* linear in the continuation variables. Most importantly, α is used *twice* — first with the Left constructor then second with Right — acting as a bait-and-switch by being given two different (and incompatible) options at different times. Programs that used continuations in a non-linear way are effectively not purely functional, which is what gives them their non-intuitionistic character.

What about *dne*? Is that linear? It seems like the continuation β of the false type is never used. But what if we re-defined falsehood as a *codata type* with one destructor [] that returns nothing, with these (linear) typing rules

$$\frac{}{\bullet \mid [] : \bot \vdash \bullet}\ \bot L \qquad\qquad \frac{c : (\Gamma \vdash \Delta)}{\Gamma \vdash \mu[].c : \bot \mid \Delta}\ \bot R$$

so that we could write the double negation elimination realizer as

$$\begin{aligned} \mathit{dne} \quad &: ((A \to \bot) \to \bot) \to A \\ \mathit{dne}\ h &= \mu\alpha{:}A.\langle h \| (\lambda x{:}A.\mu[].\langle x \| \alpha\rangle) \cdot []\rangle \end{aligned}$$

Is this definition of *dne* in terms of the falsehood codata type $\bot$ linear or non-linear?

Exercise 2.1.3. de Morgan's laws of duality are:

$$\begin{aligned} \neg(A \vee B) &\iff (\neg A) \wedge (\neg B) \\ \neg(A \wedge B) &\iff (\neg A) \vee (\neg B) \end{aligned}$$

Try to write intuitionistic realizers (*i.e.,* in a pure functional language) for both directions of these two laws. Is there any direction that you can't write?

Try again to write classical realizers (*i.e.,* in the language of the abstract machine) for both directions of these two laws. The pattern-matching **case** expression on $A * B$ and $A + B$ types correspond to these continuations:

$$\langle \mathbf{case}\, M\, \mathbf{of}\, (x, y) \to N \| E \rangle \mapsto \langle M \| \mathbf{case\,of}\, (x, y) \to \langle N \| E \rangle \rangle$$

$$\left\langle \begin{array}{l} \mathbf{case}\, M\, \mathbf{of} \\ \quad \mathrm{Left}\, x \to N_1 \\ \quad \mathrm{Right}\, y \to N_2 \end{array} \middle\| E \right\rangle \mapsto \left\langle M \middle\| \begin{array}{l} \mathbf{case\,of} \\ \quad \mathrm{Left}\, x \to \langle N_1 \| E \rangle \\ \quad \mathrm{Right}\, y \to \langle N_2 \| E \rangle \end{array} \right\rangle$$

Alternatively, you could write equations on machine commands that define the functions. For example, here is an equivalent definition of *swap* and $flip$ (from above) as equations on commands:

$$\begin{aligned} \langle \mathit{swap} \| (x, y) \cdot \alpha \rangle &= \langle (y, x) \| \alpha \rangle \\ \langle \mathit{flip} \| (\mathrm{Left}\, x) \cdot \alpha \rangle &= \langle \mathrm{Right}\, x \| \alpha \rangle \\ \langle \mathit{flip} \| (\mathrm{Right}\, y) \cdot \alpha \rangle &= \langle \mathrm{Left}\, y \| \alpha \rangle \end{aligned}$$

Can you write programs for all four?

2.2 Constructive evidence

We humans are all finite beings, with a limited view and knowledge of the universe. Almost assuredly, I know something you don't know, and you know something I don't know. A *proof* is a way to transmit knowledge from one finite being to another, and to *convince* even a careful skeptic that it must be correct.

Constructivist motto:

> It's not enough to say *that* your judgement is correct (something works/holds/is true). You must also explain *why* (it works/holds/is true).

Thus, a proof should *construct* an artifact with enough evidence that can be externally checked by any reasonable skeptic which irrefutably justifies the claim.

Who is the skeptic? Should be

- honest (doubts are reasonably grounded and consistent with the shared knowledge) and
- careful (follows agreed-upon rules to a fault), but
- not necessarily creative (can't assume that big leaps of logic are "obvious," must explain everything in small, clear steps)

2.2.1 A rational proof?

Theorem 2.2.1 (Classical (ir)rationality)**.** *There exist two irrational numbers, x, y, such that x^y is a rational number.*

Proof (non-constructive). Let's try $x = y = \sqrt{2}$. $\sqrt{2}^{\sqrt{2}}$ is **either rational or irrational**.

- If $\sqrt{2}^{\sqrt{2}} = x^y$ is rational, then we are done, because $x = y = \sqrt{2}$ are both irrational numbers with a rational exponent x^y.
- Otherwise, $\sqrt{2}^{\sqrt{2}}$ is irrational, so instead try $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$ — both of these are irrational numbers. Then notice that the

  $$x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\times\sqrt{2}} = \sqrt{2}^{\sqrt{2}^2} = \sqrt{2}^2 = 2$$

  is a rational number, derived as the exponent of two irrational numbers. □

2.2.2 Construction of intuitionistic evidence

Intuitionistic evidence supporting truth:

- Evidence *for* $\top$ is trivial, since $\top$ is always trivially true by definition.
- There is no evidence *for* $\bot$, since $\bot$ is always false by definition.
- Evidence *for* $A \wedge B$ consists of *both* evidence *for* A *and* evidence *for* B (both are needed at the same time)
- Evidence *for* $A \vee B$ consists of *either* evidence *for* A *or* evidence *for* B (just one is enough, but you must choose, and your choice is communicated along with the supporting evidence)
- Evidence *for* $\exists x{:}A.P(x)$ consists of a *witness* x from the domain A *together with* evidence *for* P instantiated at x.
- Evidence *for* $\forall x{:}A.P(x)$ consists of an algorithm which takes any x from the domain A and produces the associated evidence *for* P at x.
- Evidence *for* $A \Rightarrow B$ consists of an algorithm which transforms arbitrary evidence *for* A into some evidence *for* B.
- Evidence *for* $\neg A$ consists of an algorithm which transforms arbitrary evidence *for* A into a contradiction (such as evidence showing $\bot$ is true).

If you want to provide evidence that A is true, there are many different specific ways, which depends on the proposition A in question. This is *directly* specific to A.

If you want to provide evidence that A is false, you can only give evidence that $\neg A$ is true, which *always* takes the shape of an algorithm of deriving a contradiction for any way in which A might be true. This is *indirect*, and has nothing to do with A.

For example, how do we show that $\exists n{:}\mathbb{N}.n+n = n$ is true? Provide a concrete witness (0), which we can show that $0 + 0 = 0$.

How do we show that we show that $\forall n{:}\mathbb{N}.n + n = n$ is false? Suppose that it is true, and the derive a contradiction. But we are not obliged to provide a concrete *counterexample*, which would be a number such that the equation doesn't hold.

2.2.3 Construction of classical evidence

Intuitionistic evidence is *too vague* about falsehood. It has a rich language of *truth*, but its idea of *false* is all irreparably smashed through the bottleneck of negation ($\neg A$).

Classical evidence lets us talk about *both* truth and falsehood at the same level.

Split some connectives between *positive* and *negative*:

Standard	Positive	Negative
$A \wedge B$	$A \otimes B$	$A \mathbin{\&} B$
$A \vee B$	$A \oplus B$	$A \parr B$
$\top$	1	$\top$
$\bot$	0	$\bot$
$\neg A$	$\ominus A$	$\neg A$

$\forall x{:}A.P(x)$ and $A \Rightarrow B$ are just *negative*, and $\exists x{:}A.P(x)$ is just *positive*.

Classical evidence supporting truth of *positive* connectives:

- Evidence *for* 1, 0, $A \otimes B$, $A \oplus B$, and $\exists x{:}A.P(x)$ is defined the same as intuitionistic evidence *for* $\top$, $\bot$, $A \wedge B$, $A \vee B$, and $\exists x{:}A.P(x)$, respectively.

- Evidence *for* $\ominus A$ is the same as evidence *against* A.

Classical evidence supporting falsehood of *negative* connectives:

- Evidence *against* $\bot$ is trivial, since $\bot$ is trivially false by definition.

- There is no evidence *against* $\top$, since $\top$ is always true by definition.

- Evidence *against* $A \mathbin{\&} B$ consists of *either* evidence *against* A *or* evidence *against* B (just one is enough, but you must choose, and your choice is communicated along with the supporting evidence).

- Evidence *against* $A \parr B$ consists of *both* evidence *against* A *and* evidence *against* B (both are needed at the same time).

- Evidence *against* $\forall x{:}A.P(x)$ consists of a *counterexample* x from the domain A *together with* evidence *against* P instantiated at x.

- Evidence *against* $A \Rightarrow B$ consists of *both* evidence *for* A *and* evidence *against* B (together at the same time).

- Evidence *against* $\neg A$ is the same as evidence *for* A.

In all other cases:

- Evidence *for* a negative proposition A is an algorithm that derives a contradiction from any possible evidence *against* A.
- Evidence *against* a positive proposition A is an algorithm that derives a contradiction from any possible evidence *for* A.

Why does an intuitionist reject $\neg\neg A \Rightarrow A$? Because having concrete evidence *for* A is *stronger* than saying *it is impossible for evidence for* A *to not exist*. For example, evidence *for* $\exists n{:}\mathbb{N}.n + n = n$ includes a witness (0), but evidence *for* $\neg\neg\exists n{:}\mathbb{N}.n + n = n$ is an *algorithm* which rules out the impossibility of a witness; it *does not* have to communicate the witness to you. These two do not have the same informational content.

Similarly, constructive classical evidence *against* $\forall n{:}\mathbb{N}.n + n = n$ has *more informational content* than intuitionistic evidence *for* $\neg\forall n{:}\mathbb{N}.n + n = n$. To constructively argue *against* $\forall n{:}\mathbb{N}.n + n = n$, I must provide a counterexample (like 3) such that $3+3 \neq 3$. Instead, the intuitionistic evidence *for* $\neg\forall n{:}\mathbb{N}.n+n = n$ need only be an algorithm which shows that assuming $n + n = n$ everywhere leads to a contradiction.

What about $\ominus\neg A$ (or dually $\neg \ominus A$)? Evidence *for* $\ominus\neg A$ is *definitionally the same as* evidence *for* A. Dually, evidence *against* $\neg \ominus A$ is defined to be evidence *against* A.

2.2.4 Realizing mechanical evidence

$$\mathrm{True} \in [\![\mathrm{Bool}]\!]^+ \text{ always}$$
$$\mathrm{False} \in [\![\mathrm{Bool}]\!]^+ \text{ always}$$

$$\mathrm{Left}\, v \in [\![A \oplus B]\!]^+ \text{ if } v \in [\![A]\!]^+$$
$$\mathrm{Right}\, v \in [\![A \oplus B]\!]^+ \text{ if } v \in [\![B]\!]^+$$

$$(n, v) \in [\![\exists x{:}\mathbb{N}.P(x)]\!]^+ \text{ if } n \in \mathbb{N} \text{ and } v \in [\![P(n)]\!]^+$$

$$v \cdot E \in [\![A \to B]\!]^- \text{ if } v \in [\![A]\!]^+ \text{ and } E \in [\![B]\!]^-$$

$$\mathrm{First}\, E \in [\![A \,\&\, B]\!]^- \text{ if } E \in [\![A]\!]^-$$
$$\mathrm{Second}\, E \in [\![A \,\&\, B]\!]^- \text{ if } E \in [\![B]\!]^-$$

$$(n, E) \in [\![\forall x{:}\mathbb{N}.P(x)]\!]^- \text{ if } n \in \mathbb{N} \text{ and } E \in [\![P(n)]\!]^-$$

Exercise 2.2.2. Write down the basic cases for constructing positive evidence for

- $[\![A \oplus B]\!]^+$,
- $[\![1]\!]^+$,
- $[\![0]\!]^+$, and
- $[\![\ominus A]\!]^+$,

and the basic cases for constructing negative evidence for

- $[\![A \parr B]\!]^-$,
- $[\![\top]\!]^-$,
- $[\![\bot]\!]^-$, and
- $[\![\neg A]\!]^-$.

Exercise 2.2.3. Now that the disjunctive/conjunctive connectives have been split into positive versus negative interpretations, we have the freedom a choosing to interpret logical principles using either one. For example, previously we had interpreted the law of excluded middle with a positive disjunction as $\neg A \oplus 0$. This type promises to make a concrete decision on which of $\neg A$ or A must be true (and thus being responsible for providing supporting evidence of why the choice was the correct one).

Another way of writing the law of the excluded middle is with a negative disjunction as $\neg A \parr A$. From the negative perspective, this type says that to refute $\neg A \parr A$, you would have to provide evidence *against* $\neg A$ (which is the same as evidence *for* A) while simultaneously providing evidence *against* A. In any consistent setting, you cannot argue *for* and *against* the same A at the same time, so there is no way to refute $\neg A \parr A$.

Using your interpretation of the basic negative evidence for $[\![\neg A \parr A]\!]^-$, write a realizer capturing the argument *for* $\neg A \parr A$ by showing that every argument *against* it leads to a contradiction. How is the realizer for the negative law of excluded middle $\neg A \parr A$ different from the one for the positive law of the excluded middle $\neg A \oplus A$? *Bonus:* can you describe the differences in linearity or non-linearity between the two realizers?

Chapter 3

Computational Orthogonality

3.1 Indirect realization of evidence

What does it mean to refute a positive type or verify a negative type?

$$E \overset{?}{\in} [\![\mathrm{Bool}]\!]^- \qquad\qquad v \overset{?}{\in} [\![A \to B]\!]^+$$

- Evidence *for* a negative proposition A is an algorithm that derives a contradiction from any possible evidence *against* A.
- Evidence *against* a positive proposition A is an algorithm that derives a contradiction from any possible evidence *for* A.

For specific positive types, refutations look like

$$E \in [\![\mathrm{Bool}]\!]^- \text{ iff } \langle \mathrm{True} \| E \rangle \text{ runs and } \langle \mathrm{False} \| E \rangle \text{ runs}$$

$$\begin{aligned} E \in [\![A \oplus B]\!]^- \quad \text{if} \quad & (\langle \mathrm{Left}\, v \| E \rangle \text{ runs for all } v \in [\![A]\!]^+) \\ \text{and } & (\langle \mathrm{Right}\, v \| E \rangle \text{ runs for all } v \in [\![B]\!]^+) \end{aligned}$$

$$E \in [\![\exists x{:}\mathbb{N}.P(x)]\!]^- \text{ if } \langle (n, v) \| E \rangle \text{ runs for all } n \in \mathbb{N} \text{ and } v \in [\![P(n)]\!]^+$$

But how do we show that **if then** c_1 **else** $c_2 \in [\![\mathrm{Bool}]\!]^-$?

For specific negative types, verifications look like

$$v \in [\![A \to B]\!]^+ \text{ if } \langle v \| v' \cdot E \rangle \text{ runs for all } v' \in [\![A]\!]^+ \text{ and } E \in [\![B]\!]^-$$

$$\begin{aligned} v \in [\![A \,\&\, B]\!]^+ \quad \text{if} \quad & (\langle v \| \mathrm{First}\, E \rangle \text{ runs for all } E \in [\![A]\!]^-) \\ \text{and } & (\langle v \| \mathrm{Second}\, E \rangle \text{ runs for all } E \in [\![B]\!]^-) \end{aligned}$$

$$v \in [\![\forall x{:}\mathbb{N}.P(x)]\!]^+ \text{ if } \langle v \| (n, E)\rangle \text{ runs for all } n \in \mathbb{N} \text{ and } E \in [\![P(n)]\!]^-$$

But how do we show that $\lambda x.v \in [\![A \to B]\!]^+$?

3.2 Machine orthogonality

Definition 3.2.1 (Pole)**.** *running set* of commands $⫫$, also known as a *pole*, can be *any chosen* set of commands satisfying some condition. . . (see later)

Definition 3.2.2 (Orthogonality)**.** $⫫$-*orthogonality* between an individual term v and continuation E of the machine language, written as $v ⫫ E$, is

$$v ⫫ E \text{ iff } \langle v \| E\rangle \in ⫫$$

$⫫$-*orthogonality* between a (potentially empty) subset of terms ($\mathbb{A}^+ = \{v_1, v_2, \dots\}$) and continuations ($\mathbb{B}^- = \{E_1, E_2, \dots\}$) of the machine language, written as $\mathbb{A}^+ ⫫ \mathbb{B}^-$, is

$$\mathbb{A}^+ ⫫ \mathbb{B}^- \text{ iff } \text{ for all } v \in \mathbb{A}^+, E \in \mathbb{B}^-,\ v ⫫ E$$

Given any subset $\mathbb{A}^+$ of terms ($\{v, \dots\}$) from the machine language, *the* $⫫$-*orthogonal of* $\mathbb{A}^+$, written as $\mathbb{A}^{+⫫}$, is the *largest* subset of continuations ($\{E, \dots\}$) such that $\mathbb{A}^+ ⫫ \mathbb{A}^{+⫫}$. In other words, $\mathbb{A}^{+⫫}$ is defined as

$$\mathbb{A}^{+⫫} = \{E \mid \forall v \in \mathbb{A}^+,\ v ⫫ E\}$$

Given any subset $\mathbb{A}^-$ of continuations ($\{E, \dots\}$) from the machine language, the $⫫$-*orthogonal of* $\mathbb{A}^-$, written as $\mathbb{A}^{-⫫}$, is the *largest* subset of terms ($\{v, \dots\}$) such that $\mathbb{A}^{-⫫} ⫫ \mathbb{A}^-$. In other words, $\mathbb{A}^{-⫫}$ is defined as

$$\mathbb{A}^{-⫫} = \{v \mid \forall E \in \mathbb{A}^-,\ v ⫫ E\}$$

Example 3.2.3. The empty set is orthogonal to any set of terms ($\mathbb{A}^+ ⫫ \{\}$) or continuations ($\{\} ⫫ \mathbb{B}^-$). This holds no matter how $⫫$ is defined.

Example 3.2.4. Suppose that the running set $⫫$ contains only *terminating, type-safe commands*, that is

Definition 3.2.5 (Safety Pole).

$$⫫ = \{c \mid \exists \text{ final } c' \text{ s.t. } c \mapsto\!\!\!\!\to c'\}$$

for *any* chosen collection/set/judgement of acceptable "final" commands. For example, we might say that $\langle \mathrm{True} \| \alpha\rangle$ and $\langle \mathrm{False} \| \alpha\rangle$ (where α denotes a "top-level"/initial continuation) or even $\langle x \| v \cdot E\rangle$ (for head reduction hitting a free variable x) and $\langle x \| \alpha\rangle$ (for a totally generic final state) are all acceptable final commands. But we can still rule out *fatal* type errors such as $\langle \mathrm{True} \| x \cdot \alpha\rangle$ (boolean constants cannot be applied to arguments) or $\langle \lambda x.v \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2\rangle$ (if-then-else decisions don't make sense on functions), which are excluded from $⫫$.

Suppose that $c_1, c_2 \in ⫫$, *i.e.,* $c_i \mapsto\!\!\!\twoheadrightarrow c_i'$ final. It follows that $\textbf{if then}\, c_1\, \textbf{else}\, c_2 \in \{\mathrm{True}, \mathrm{False}\}^{⫫}$ because

$$\begin{aligned}\langle \mathrm{True} \| \textbf{if then}\, c_1\, \textbf{else}\, c_2\rangle &\mapsto c_1 \mapsto\!\!\!\twoheadrightarrow c_1' \text{ final}\\ \langle \mathrm{False} \| \textbf{if then}\, c_1\, \textbf{else}\, c_2\rangle &\mapsto c_2 \mapsto\!\!\!\twoheadrightarrow c_2' \text{ final}\end{aligned}$$

Definition 3.2.6 (Pole)**.** A *pole* $⫫$ can be *any chosen* set of commands that is *closed under expansion:* $c \in ⫫$ whenever $c \mapsto c' \in ⫫$.

3.3 Logic of orthogonality

Intuition: *orthogonality* in an *abstract machine* follows similar laws as *negation* in *intuitionistic logic.* In other words, we can interpret logical implication ($\Rightarrow$) as set inclusion ($\subseteq$) and logical negation ($\neg$) as orthogonality ($\_^{⫫}$).

Notation: $\mathbb{A}^{\pm}$ stands for *either* a set of machine terms or a set of machine continuations (your choice). Given several such sets, $\mathbb{A}_1^{\pm}, \dots, \mathbb{A}_n^{\pm}$, assume that the same choice is made for each of them (they are all sets of terms, or sets of continuations). The opposite choice is written $\mathbb{A}^{\mp}$.

Property 3.3.1 (Contrapositive)**.** *If* $\mathbb{A}^{\pm} \subseteq \mathbb{B}^{\pm}$*, then* $\mathbb{B}^{\pm⫫} \subseteq \mathbb{A}^{\pm⫫}$.

Proof. Without loss of generality, assume that $\mathbb{A}^{\pm} = \mathbb{A}^{+}$ and $\mathbb{B}^{\pm} = \mathbb{B}^{+}$ are sets of machine terms (the other case follows dually), and suppose $\mathbb{A}^{+} \subseteq \mathbb{B}^{+}$.

Given an arbitrary machine continuation $E \in \mathbb{B}^{+⫫}$, we must show that $E \in \mathbb{A}^{+⫫}$. In other words, given that $v' \mathrel{⫫} E$ for all $v' \in \mathbb{B}^{+}$ (the definition of $\mathbb{B}^{+⫫}$) we must prove that $v' \mathrel{⫫} E$ for all $v \in \mathbb{A}^{+}$ (the definition of $\mathbb{B}^{+⫫}$). Since $\mathbb{A}^{+} \subseteq \mathbb{B}^{+}$, any $v \in \mathbb{A}^{+}$ is also a $v \in \mathbb{B}^{+}$, which forces $v \mathrel{⫫} E$ because of the definition of $E \in \mathbb{B}^{+⫫}$. Therefore, $E \in \mathbb{A}^{+⫫}$ by definition of orthogonality. □

Property 3.3.2 (Double Orthogonal Introduction)**.** $\mathbb{A}^{\pm} \subseteq \mathbb{A}^{\pm⫫⫫}$.

Proof. Without loss of generality, assume that $\mathbb{A}^{\pm} = \mathbb{A}^{+}$ is a set of machine terms (the other case follows dually).

Suppose $v \in \mathbb{A}^{+}$, and we must now show that $v \in \mathbb{A}^{+⫫⫫}$, *i.e.,* that $v \mathrel{⫫} E$ for all $E \in \mathbb{A}^{+⫫}$. By definition of $\mathbb{A}^{+⫫}$, it must be that $v \mathrel{⫫} E$ for any $E \in \mathbb{A}^{+⫫}$. Therefore, $v \in \mathbb{A}^{+⫫⫫}$ by definition of orthogonality. □

Property 3.3.3 (Triple Orthogonal Elimination)**.** $\mathbb{A}^{\pm⫫⫫⫫} = \mathbb{A}^{\pm⫫}$.

Proof. Exercise left to reader. *Hint:* Follows directly from double orthogonal introduction and contrapositive, *i.e.,* you do not need to refer to the definition of $\_^{⫫}$. To show the two sides are equal, you can prove $\mathbb{A}^{\pm⫫⫫⫫} \subseteq \mathbb{A}^{\pm⫫}$ and $\mathbb{A}^{\pm⫫⫫⫫} \supseteq \mathbb{A}^{\pm⫫}$ separately. □

Other logical connectives can be understood as set operations. Conjunction ($\wedge$) corresponds to set union ($\cup$) and disjunction ($\vee$) corresponds to set intersection ($\cap$).

Property 3.3.4 (De Morgan Laws)**.**

1. $(\mathbb{A}^{\pm} \cup \mathbb{B}^{\pm})^{\perp\!\!\!\perp} = \mathbb{A}^{\pm\perp\!\!\!\perp} \cap \mathbb{B}^{\pm\perp\!\!\!\perp}$

2. $(\mathbb{A}^{\pm} \cap \mathbb{B}^{\pm})^{\perp\!\!\!\perp} \supseteq \mathbb{A}^{\pm\perp\!\!\!\perp} \cup \mathbb{B}^{\pm\perp\!\!\!\perp}$

3. There (may) exist instances where $(\mathbb{A}^{\pm} \cap \mathbb{B}^{\pm})^{\perp\!\!\!\perp} \not\subseteq \mathbb{A}^{\pm\perp\!\!\!\perp} \cup \mathbb{B}^{\pm\perp\!\!\!\perp}$

Proof. (1) $(\mathbb{A}^{\pm} \cup \mathbb{B}^{\pm})^{\perp\!\!\!\perp} = \mathbb{A}^{\pm\perp\!\!\!\perp} \cap \mathbb{B}^{\pm\perp\!\!\!\perp}$ and (2) $(\mathbb{A}^{\pm} \cap \mathbb{B}^{\pm})^{\perp\!\!\!\perp} \supseteq \mathbb{A}^{\pm\perp\!\!\!\perp} \cup \mathbb{B}^{\pm\perp\!\!\!\perp}$ are left as an exercise to the reader.

To show the failure of (3), we only need to exhibit a concrete example of $\perp\!\!\!\perp$, $\mathbb{A}^{\pm}$, and $\mathbb{B}^{\pm}$ where the subset inclusion fails.

Suppose that $\perp\!\!\!\perp$ is the safety pole (definition 3.2.5)

$$\perp\!\!\!\perp = \{c \mid \exists \text{ final } c' \text{ s.t. } c \mapsto\!\!\!\!\rightarrow c'\}$$

and that there is *at least one* command $\mho$ not in $\perp\!\!\!\perp$. For example, we might have the fatal type error $\mho = \langle \mathrm{True} \| \mathrm{False} \cdot \alpha \rangle \notin \perp\!\!\!\perp$.

Now, consider this **if then else** continuation that is always unsafe for either boolean value:

$$E_{\mho} = \textbf{if then}\, \mho \,\textbf{else}\, \mho$$

and notice that

$$\begin{array}{lll} E_{\mho} \notin \{\mathrm{True}\}^{\perp\!\!\!\perp} & \text{because} & \langle \mathrm{True} \| E_1 \rangle \mapsto_{\beta_{\mathrm{Bool}\,1}} \mho \text{ not final} \\ E_{\mho} \notin \{\mathrm{False}\}^{\perp\!\!\!\perp} & \text{because} & \langle \mathrm{False} \| E_2 \rangle \mapsto_{\beta_{\mathrm{Bool}\,2}} \mho \text{ not final} \end{array}$$

so this continuation is not found in the union of the two orthogonals,

$$E_{\mho} \notin \{\mathrm{True}\}^{\perp\!\!\!\perp} \cup \{\mathrm{False}\}^{\perp\!\!\!\perp}$$

However, this continuation *is* found in the orthogonal of intersection

$$E_{\mho} \in (\{\mathrm{True}\} \cap \{\mathrm{False}\})^{\perp\!\!\!\perp}$$

because the intersection $\{\mathrm{True}\} \cap \{\mathrm{False}\} = \{\}$ is empty! So by definition, $(\{\mathrm{True}\} \cap \{\mathrm{False}\})^{\perp\!\!\!\perp} = \{\}^{\perp\!\!\!\perp}$ is the largest set such that $\{\} \perp\!\!\!\perp \{\}^{\perp\!\!\!\perp}$. Since there are no terms to consider — and therefore no *reason* to rule out any continuation as unsafe — $\{\}^{\perp\!\!\!\perp}$ contains *every* continuation of the machine language. Specifically, $E_{\mho} \in \{\}^{\perp\!\!\!\perp}$. □

3.4 Semantic types

In the type system of the abstract machine, a *syntactic type* categorizes *both* a *term* and a *continuation.*

Likewise, the semantics of a type should specify *both all the terms* included in that type *as well as all the continuations* included in that type.

But how do we know if such a definition of a semantic type is good? It needs to be both

- *sound* — meaning that interactions allowed by the type are all safe, and
- *complete* — meaning that anything that *could* be safely included *is.* In other words, terms and continuations are only excluded when there is a *reason* they would be unsafe.

In particular, soundness justifies the safety of the *Cut* rule. Completeness ensures there is enough stuff in the type, and in particular, lets us reason by *inversion* on some canonical constructions.

Definition 3.4.1 (Orthogonal Candidate)**.** A *pre-candidate* for the semantic interpretation of a type is a pair $\mathbb{A} = (\mathbb{A}^+, \mathbb{A}^-)$ where

- $\mathbb{A}^+$ is a set of machine terms, and
- $\mathbb{A}^-$ is a set of machine continuations.

A $\perp\!\!\!\perp$-*orthogonal candidate* for the semantic interpretation of a type is a pre-candidate $(\mathbb{A}^+, \mathbb{A}^-)$ such that $\mathbb{A}^+ = \mathbb{A}^{-\perp\!\!\!\perp}$ and $\mathbb{A}^- = \mathbb{A}^{+\perp\!\!\!\perp}$. In other words, every $\perp\!\!\!\perp$-orthogonal candidate is

- *sound,* meaning that $\mathbb{A}^+ \perp\!\!\!\perp \mathbb{A}^-$, *i.e.,*

  $$\forall v \in \mathbb{A}^+, E \in \mathbb{A}^-,\ v \perp\!\!\!\perp E$$

  equivalent to the fact that $\mathbb{A}^+ \subseteq \mathbb{A}^{-\perp\!\!\!\perp}$ and $\mathbb{A}^- \subseteq \mathbb{A}^{+\perp\!\!\!\perp}$, and
- *complete,* meaning that
  - $v \in \mathbb{A}^+$ whenever $v \perp\!\!\!\perp E$ for all $E \in \mathbb{A}^-$, equivalent to the fact that $\mathbb{A}^{-\perp\!\!\!\perp} \subseteq \mathbb{A}^+$, and
  - $E \in \mathbb{A}^-$ whenever $v \perp\!\!\!\perp E$ for all $v \in \mathbb{A}^+$, equivalent to the fact that $\mathbb{A}^{+\perp\!\!\!\perp} \subseteq \mathbb{A}^-$

But how do we make one of these things? Two ways:

- *Positive:* Start with your collection of "canonical" constructions (terms). Pick *all* the continuations that are safe with those canonical terms. Then pick any (additional) terms that are compatible with *those* continuations.

  In symbols, if you start with the set $\mathbb{C}^+$ of canonical term constructions, then the *positive* $\perp\!\!\!\perp$*-orthogonal candidate* containing $\mathbb{C}^+$ is:

  $$\mathrm{Pos}(\mathbb{C}^+) = (\mathbb{C}^{+\perp\!\!\!\perp\perp\!\!\!\perp}, \mathbb{C}^{+\perp\!\!\!\perp})$$

- *Negative:* Start with your collection of "canonical" observations (continuations). Pick *all* the terms that are safe with those canonical observations. Then pick any (additional) continuations that are compatible with *those* terms.

  In symbols, if you start with the set $\mathbb{C}^-$ of canonical observation constructions, then the *negative* $\perp\!\!\!\perp$*-orthogonal candidate* containing $\mathbb{C}^-$ is:

  $$\mathrm{Neg}(\mathbb{C}^-) = (\mathbb{C}^{-\perp\!\!\!\perp}, \mathbb{C}^{-\perp\!\!\!\perp\perp\!\!\!\perp})$$

Why do these work?

Property 3.4.2. *For any set of terms* $\mathbb{C}^+$ *and continuations* $\mathbb{C}^-$*, both* $\mathrm{Pos}(\mathbb{C}^+)$ *and* $\mathrm{Neg}(\mathbb{C}^-)$ *are* $\perp\!\!\!\perp$*-orthogonal candidates.*

Proof. Since the positive and negative cases are perfectly symmetric, consider just the case of $\mathrm{Pos}(\mathbb{C}^+)$ below.

By definition, $\mathrm{Pos}(\mathbb{C}^+) = (\mathbb{C}^{+\perp\!\!\!\perp\perp\!\!\!\perp}, \mathbb{C}^{+\perp\!\!\!\perp})$. Note that the term side $\mathbb{C}^{+\perp\!\!\!\perp\perp\!\!\!\perp}$ is equal to the orthogonal to the continuation side, $(\mathbb{C}^{+\perp\!\!\!\perp})^{\perp\!\!\!\perp}$, by definition. Furthermore, the continuation side $\mathbb{C}^{+\perp\!\!\!\perp}$ is equal to the orthogonal of the term side, $(\mathbb{C}^{+\perp\!\!\!\perp\perp\!\!\!\perp})^{\perp\!\!\!\perp}$, by triple orthogonal elimination. □

Example positive semantics for booleans:

$$\begin{aligned}
\llbracket \mathrm{Bool} \rrbracket &= \mathrm{Pos}(\{\mathrm{True}, \mathrm{False}\}) \\
\llbracket \mathrm{Bool} \rrbracket^- &= \{\mathrm{True}, \mathrm{False}\}^{\perp\!\!\!\perp} = \{E \mid \mathrm{True} \perp\!\!\!\perp E \text{ and } \mathrm{False} \perp\!\!\!\perp E\} \\
\llbracket \mathrm{Bool} \rrbracket^+ &= \{\mathrm{True}, \mathrm{False}\}^{\perp\!\!\!\perp\perp\!\!\!\perp} = \{v \mid \forall E,\ \mathrm{True} \perp\!\!\!\perp E \text{ and } \mathrm{False} \perp\!\!\!\perp E \text{ implies } v \perp\!\!\!\perp E\}
\end{aligned}$$

$\mathrm{True}, \mathrm{False} \in \llbracket \mathrm{Bool} \rrbracket = \{\mathrm{True}, \mathrm{False}\}^{\perp\!\!\!\perp\perp\!\!\!\perp}$ by double orthogonal introduction.

For any $c_1, c_2 \in \perp\!\!\!\perp$, we have $\mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2 \in \llbracket \mathrm{Bool} \rrbracket = \{\mathrm{True}, \mathrm{False}\}^{\perp\!\!\!\perp}$ because $\perp\!\!\!\perp$ is *closed under expansion*.

$$\begin{aligned}
\langle \mathrm{True} \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2 \rangle \mapsto c_1 \in \perp\!\!\!\perp \text{ so } \langle \mathrm{True} \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2 \rangle \in \perp\!\!\!\perp \\
\langle \mathrm{False} \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2 \rangle \mapsto c_2 \in \perp\!\!\!\perp \text{ so } \langle \mathrm{False} \| \mathbf{if\,then}\, c_1\, \mathbf{else}\, c_2 \rangle \in \perp\!\!\!\perp
\end{aligned}$$

If $c[E/\alpha]$ for any $E \in \llbracket \mathrm{Bool} \rrbracket^-$, then $\mu\alpha.c \in \llbracket \mathrm{Bool} \rrbracket^+ = \llbracket \mathrm{Bool} \rrbracket^{-\perp\!\!\!\perp}$ because $\perp\!\!\!\perp$ is *closed under expansion*. Given any $E \in \mathrm{Bool}^-$,

$$\langle \mu\alpha.c \| E \rangle \mapsto c[E/\alpha] \in \perp\!\!\!\perp \text{ so } \langle \mu\alpha.c \| E \rangle$$

Example negative semantics for functions:

$$\begin{aligned}
\llbracket A \to B \rrbracket &= \mathrm{Neg}(\{v \cdot E \mid v \in \llbracket A \rrbracket^+, E \in \llbracket B \rrbracket^-\}) \\
\llbracket A \to B \rrbracket^+ &= \{v \mid \forall v' \in \llbracket A \rrbracket^+, E \in \llbracket B \rrbracket^-,\ v \perp\!\!\!\perp v' \cdot E\} \\
\llbracket A \to B \rrbracket^- &= \{E \mid \forall v,\ (\forall v' \in \llbracket A \rrbracket^+, E \in \llbracket B \rrbracket^-,\ v \perp\!\!\!\perp v' \cdot E) \text{ implies } v \perp\!\!\!\perp E\}
\end{aligned}$$

If $v \in \llbracket A \rrbracket^+$ and $E \in \llbracket B \rrbracket^-$, then $v \cdot E \in \llbracket A \to B \rrbracket$ by double orthogonal introduction.

If $v[v'/x] \in \llbracket B \rrbracket^+$ for all $v' \in \llbracket A \rrbracket^+$ and $\llbracket B \rrbracket$ is a $\perp\!\!\!\perp$-orthogonal candidate, then $\lambda x.v \in \llbracket A \to B \rrbracket^+$ because $\perp\!\!\!\perp$ is closed under expansion. Given any $v' \in \llbracket B \rrbracket^+$, $E \in \llbracket B \rrbracket^-$,

$$\begin{aligned}
\langle \lambda x.v \| v' \cdot E \rangle &\mapsto \langle v[v'/x] \| E \rangle && (\beta_\to) \\
&\in \perp\!\!\!\perp && (\textit{soundness: } \llbracket B \rrbracket^+ \perp\!\!\!\perp \llbracket B \rrbracket^-) \\
\langle \lambda x.v \| v' \cdot E \rangle &\in \perp\!\!\!\perp && (\textit{expansion} \text{ of } \perp\!\!\!\perp)
\end{aligned}$$

3.5 Interpretation of types

Goal: Interpret typing judgements, $c : (\Gamma \vdash \Delta)$, etc., as statements.

Environments Γ and Δ are interpreted as specifications on (simultaneous) substitutions, $\sigma = v_1/x_1, \dots, v_n/x_n, E_1/\alpha_1, \dots, E_n/\alpha_n$.

$$\begin{aligned} [\![\Gamma]\!] &= \{\sigma \mid \forall x{:}A \in \Gamma,\ x[\sigma] \in [\![A]\!]^+\} \\ [\![\Delta]\!] &= \{\sigma \mid \forall \alpha{:}A \in \Delta,\ \alpha[\sigma] \in [\![A]\!]^-\} \end{aligned}$$

Semantic judgements:

$$\begin{aligned} c : (\Gamma \vDash \Delta) &= \forall \sigma,\ \sigma \in [\![\Gamma]\!] \text{ and } \sigma \in [\![\Delta]\!] \text{ implies } c[\sigma] \in \perp\!\!\!\perp \\ \Gamma \vDash v : A \mid \Delta &= \forall \sigma,\ \sigma \in [\![\Gamma]\!] \text{ and } \sigma \in [\![\Delta]\!] \text{ implies } v[\sigma] \in [\![A]\!]^+ \\ \Gamma \mid e : A \vDash \Delta &= \forall \sigma,\ \sigma \in [\![\Gamma]\!] \text{ and } \sigma \in [\![\Delta]\!] \text{ implies } e[\sigma] \in [\![A]\!]^- \end{aligned}$$

Lemma 3.5.1 (Adequacy)**.** *For* any *pole* $\perp\!\!\!\perp$*,*

1. *If* $c : (\Gamma \vdash \Delta)$ *is derivable, then* $c : (\Gamma \vDash \Delta)$ *holds.*

2. *If* $\Gamma \vdash v : A \mid \Delta$ *is derivable, then* $\Gamma \vDash v : A \mid \Delta$ *holds.*

3. *If* $\Gamma \mid E : A \vdash \Delta$*, then* $\Gamma \mid E : A \vDash \Delta$ *holds.*

Proof. By induction on the structure of the given typing derivation. □

Theorem 3.5.2 (Boolean Command)**.** *If* $c : (\bullet \vdash \alpha : \mathrm{Bool})$ *then* $c \mapsto\!\!\!\!\to \langle \mathrm{True} \| \alpha \rangle$ *or* $c \mapsto\!\!\!\!\to \langle \mathrm{False} \| \alpha \rangle$*.*

Proof. First, set $\perp\!\!\!\perp$ to the safety pole

$$\perp\!\!\!\perp = \{c \mid \exists \text{ final } c' \text{ s.t. } c \mapsto\!\!\!\!\to c'\}$$

where the *only* final commands are $\langle \mathrm{True} \| \alpha \rangle$ and $\langle \mathrm{False} \| \alpha \rangle$. Note that this forces $\alpha \in [\![\mathrm{Bool}]\!]^-$. As such, the identity substitution α/α is a valid instance of the output environment $[\![\alpha : \mathrm{Bool}]\!]$.

From adequacy, the derivation of $c : (\bullet \vdash \alpha : \mathrm{Bool})$ ensures $c : (\bullet \vDash \alpha : \mathrm{Bool})$. Therefore, $c[\alpha/\alpha] = c \in \perp\!\!\!\perp$, in other words, $c \mapsto\!\!\!\!\to \langle \mathrm{True} \| \alpha \rangle$ or $c \mapsto\!\!\!\!\to \langle \mathrm{False} \| \alpha \rangle$. □

Corollary 3.5.3 (Boolean Decision)**.** *If* $\bullet \vdash v : \mathrm{Bool} \mid \bullet$ *then* $\langle v \| \alpha \rangle \mapsto\!\!\!\!\to \langle \mathrm{True} \| \alpha \rangle$ *or* $\langle v \| \alpha \rangle \mapsto\!\!\!\!\to \langle \mathrm{False} \| \alpha \rangle$*.*

Exercise 3.5.4. The boolean semantic type can be generalized to the sum semantic type defined as this positive candidate:

$$[\![A \oplus B]\!] = \mathrm{Pos}\left(\{\mathrm{Left}\, v \mid v \in [\![A]\!]^+\} \cup \{\mathrm{Right}\, v \mid v \in [\![B]\!]^+\}\right)$$

Use adequacy to prove that

Theorem 3.5.5 (Constructive Decision). *If* $\bullet \vdash v : A \oplus B \mid \bullet$ *then* $\langle v \| \alpha \rangle \mapsto\!\!\!\!\to \langle \mathrm{Left}\, v' \| \alpha \rangle$ *for some* $v' \in [\![A]\!]^+$*, or* $\langle v \| \alpha \rangle \mapsto\!\!\!\!\to \langle \mathrm{Right}\, v' \| \alpha \rangle$ *for some* $v' \in [\![B]\!]^-$*.*

Exercise 3.5.6. A semantic version of the existential quantification over numbers, $\exists x{:}\mathbb{N}.P(x)$, can be interpreted as a positive candidate:

$$[\![\exists x{:}\mathbb{N}.P(x)]\!] = \mathrm{Pos}(\{(n, v) \mid n \in \mathbb{N}, v \in [\![P(n)]\!]^+\})$$

Use adequacy to prove that

Theorem 3.5.7 (Constructive Decision). *If* $\bullet \vdash v : \exists x{:}\mathbb{N}.P(x) \mid \bullet$ *then* $\langle v \| \alpha \rangle \mapsto\!\!\!\!\twoheadrightarrow \langle (n, v') \| \alpha \rangle$ *for some* $n \in \mathbb{N}$ *and* $v' \in [\![P(n)]\!]^+$.

3.6 Equational reasoning — generalizing to binary relations

The model above (built on $c : (\Gamma \vDash \Delta)$, $\Gamma \vDash v : A \mid \Delta$, and $\Gamma \mid E : A \vDash \Delta$) is good for describing *unary* predicates that ask a question about just *one* thing: is *this* expression type safe, *does* this expression terminate, *etc.* This matches the unary nature of typing judgements deciding that just *one* expression (command, term, or continuation) is well-typed at a time. These questions can be asked (and answered) by strategically capturing the main predicate on commands in the choice for $\perp\!\!\!\perp$, and the rest of the model for types compatible with that $\perp\!\!\!\perp$ follows automatically.

But what about *binary* relationships that ask a question about *two* things: are *these two* terms equivalent in any well-typed closing context, so that they would always produce the same answer? For example, you might want to ask if you can prove two commands equal using axioms as $c =_{\mu\beta\eta} c'$, then are they contextually/observationally equivalent (and similar for $v =_{\mu\beta\eta} v' : A$ and $E =_{\mu\beta\eta} E' : A$)? Such questions aren't easy to answer by just instantiating the choice of the set $\perp\!\!\!\perp$.

Instead, if you want to ask a binary question, you should have a binary model. Thankfully, the extension of the above model to binary relationships is pretty straightforward — fundamentally it does all the same things, just doubling up everything.

The important first step is to generalize the pole $\perp\!\!\!\perp$ to be a binary relation on commands, *i.e.,* a set containing pairs of commands which we decide are related. The crucial *closure under expansion* property then says that commands c_1, c_2 are always related when they step to related commands c'_1, c'_2 in the future:[1]

$$\text{if } c_i \mapsto\!\!\!\!\twoheadrightarrow c'_i \text{ (for } i \in \{1, 2\}\text{), and } (c_1,' c'_2) \in \perp\!\!\!\perp \text{ then } (c_1, c_2) \in \perp\!\!\!\perp$$

[1] The reason that I use the many-step $\mapsto\!\!\!\!\twoheadrightarrow$ here is to allow for the fact that the two initial commands c_1, c_2 may need to take a *different* number of steps before they are related in the future. For example, in order to justify the inclusion of a single step in an equational theory

$$\frac{c \mapsto c'}{c = c'}\ \textit{Inclusion}$$

we would know that $(c', c') \in \perp\!\!\!\perp$ when $\perp\!\!\!\perp$ is reflexive, and also $c \mapsto\!\!\!\!\twoheadrightarrow c'$ in one step whereas $c' \mapsto\!\!\!\!\twoheadrightarrow c'$ in zero steps.

From there, the notion of orthgonality is also generalized to judge the compatibility pairs of related terms with pairs of related continuations:

$$(v, v') \perp\!\!\!\perp (E, E') \text{ iff } (\langle v \| E \rangle, \langle v' \| E' \rangle) \in \perp\!\!\!\perp$$

And pre-candidates now store a binary relation on terms (*i.e.,* a set of pairs of terms that are related) and a binary relation on continuations (*i.e.,* a set of pairs of continuations that are related). The definition of the orthogonal to a binary relation $\mathbb{A}^{\pm\perp\!\!\!\perp}$ finds the biggest relation (*i.e.,* biggest set of related pairs) that is $\perp\!\!\!\perp$-compatible with $\mathbb{A}$, similar to before. After that point, the same logical properties of orthogonality still hold (contrapositive, double orthogonal introduction, triple orthgonoal elimination, and the de Morgan laws), so that you can follow the same path.

Chapter 4

Fixing Recursion

4.1 Two orders of candidates

Definition 4.1.1 (Partial Order)**.** A *partial order* of a collection D is any binary relation, $x \leq y$ for $x, y \in D$, with the following properties for all $x, y, z \in D$:

- *Reflexivity:* $x \leq x$,
- *Transitivity:* if $x \leq y$ and $y \leq z$ then $x \leq z$, and
- *Antisymmetry:* if $x \leq y$ and $y \leq x$ then $x = y$.

Note that partial orders *do not* need to be *total*; there is no requirement that for arbitrary $x, y \in D$, they must be ordered in one of the two possible ways — either $x \leq y$ or $y \leq x$.

Example 4.1.2. The subset relation — $X \subseteq Y$ — is a partial order over sets.

Definition 4.1.3. Given semantic type pre-candidates $\mathbb{A} = (\mathbb{A}^+, \mathbb{A}^-)$ and $\mathbb{B} = (\mathbb{B}^+, \mathbb{B}^-)$, the *refinement* (intuitively, *containment*) partial order between candidates, written $\mathbb{A} \sqsubseteq \mathbb{B}$, and the *subtype* partial order between candidates, written $\mathbb{A} \leq \mathbb{B}$ are defined as

$$\mathbb{A} \sqsubseteq \mathbb{B} = \mathbb{A}^+ \subseteq \mathbb{B}^+ \text{ and } \mathbb{A}^- \subseteq \mathbb{B}^-$$
$$\mathbb{A} \leq \mathbb{B} = \mathbb{A}^+ \subseteq \mathbb{B}^+ \text{ and } \mathbb{A}^- \supseteq \mathbb{B}^-$$

Definition 4.1.4 (Orthogonal)**.** The orthogonal of a pre-candidate is

$$(\mathbb{A}^+, \mathbb{A}^-)^{\perp\!\!\!\perp} = (\mathbb{A}^{-\perp\!\!\!\perp}, \mathbb{A}^{+\perp\!\!\!\perp})$$

Notice that $\perp\!\!\!\perp$-orthogonal candidates are exactly the pre-candidates which are fixed points of orthogonality, $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp}$.

Property 4.1.5 (Fixed-point Candidates)**.** *$\mathbb{A}$ is a $\perp\!\!\!\perp$-orthogonal candidate (according to the previous definition) exactly when it is a fixed point of orthogonality, $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp}$. Notably, the two properties of candidates are equivalent to the two directions of this equality:*

- Soundness *$(\mathbb{A}^+ \perp\!\!\!\perp \mathbb{A}^-)$ holds iff* $\mathbb{A} \sqsubseteq \mathbb{A}^{\perp\!\!\!\perp}$*, and*
- Completeness *holds iff* $\mathbb{A}^{\perp\!\!\!\perp} \sqsubseteq \mathbb{A}$.

Property 4.1.6. *Given any two $\perp\!\!\!\perp$-orthogonal candidates $\mathbb{A} = (\mathbb{A}^+, \mathbb{A}^-)$ and $\mathbb{B} = (\mathbb{B}^+, \mathbb{B}^-)$,*

$$\mathbb{A} \leq \mathbb{B} \textit{ iff } \mathbb{A}^+ \perp\!\!\!\perp \mathbb{B}^-$$

Property 4.1.7 (Refinement Orthogonal)**.** *Given any pre-candidate $\mathbb{A}$,*

1. Double orthogonal introduction: $\mathbb{A} \sqsubseteq \mathbb{A}^{\perp\!\!\!\perp\perp\!\!\!\perp}$.
2. Triple orthogonal elimination: $\mathbb{A}^{\perp\!\!\!\perp\perp\!\!\!\perp\perp\!\!\!\perp} = \mathbb{A}^{\perp\!\!\!\perp}$.

Property 4.1.8 (Monotonicity & Antitonicity)**.** *Given any pre-candidates $\mathbb{A}$ and $\mathbb{B}$,*

1. Antitonicity (*a.k.a* contrapositive): *if* $\mathbb{A} \sqsubseteq \mathbb{B}$ *then* $\mathbb{A}^{\perp\!\!\!\perp} \sqsupseteq \mathbb{B}^{\perp\!\!\!\perp}$.
2. Monotonicity: *if* $\mathbb{A} \leq \mathbb{B}$ *then* $\mathbb{A}^{\perp\!\!\!\perp} \leq \mathbb{B}^{\perp\!\!\!\perp}$.

Exercise 4.1.9. Prove the above double orthogonal introduction, triple orthogonal elimination, antitonicity, and monotonicity properties of refinement and subtyping, in terms of the more basic properties of orthogonals on a single set (chapter 3).

Can also (slightly) generalize the positive/negative construction of candidates to take an initial sound pre-candidate ($(\mathbb{C}^+, \mathbb{C}^-) = \mathbb{C} \sqsubseteq \mathbb{C}^{\perp\!\!\!\perp}$) instead of just a set, so that

$$\mathrm{Pos}(\mathbb{C}) = (\mathbb{C}^{+\perp\!\!\!\perp\perp\!\!\!\perp}, \mathbb{C}^{+\perp\!\!\!\perp}) \qquad \mathrm{Neg}(\mathbb{C}) = (\mathbb{C}^{-\perp\!\!\!\perp}, \mathbb{C}^{-\perp\!\!\!\perp\perp\!\!\!\perp})$$

By doing so, we can position these candidates as the extremal cases of *completions* of a sound but (potentially) incomplete $\mathbb{C}$, meaning that they extend $\mathbb{C}$

$$\mathbb{C} \sqsubseteq \mathrm{Pos}(\mathbb{C}) \qquad \mathbb{C} \sqsubseteq \mathrm{Neg}(\mathbb{C})$$

and Pos($\mathbb{C}$) is the *least* (*w.r.t* subtyping) one to do so, whereas Neg($\mathbb{C}$) is the *greatest* one (*w.r.t* subtyping). In other words, if you find any *other* $\perp\!\!\!\perp$-orthogonal candidate $\mathbb{A}$ extending $\mathbb{C}$ then it always happens to lie in between those two:

$$\text{if } \mathbb{C} \sqsubseteq \mathbb{C}^{\perp\!\!\!\perp} \text{ and } \mathbb{C} \sqsubseteq \mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp} \text{ then } \mathrm{Pos}(\mathbb{C}) \leq \mathbb{A} \leq \mathrm{Neg}(\mathbb{C})$$

4.2 Lattices of types — intersection and union

Definition 4.2.1 (Lattice)**.** A collection D is a *(binary, lower) semi-lattice* with respect to a partial order ($\leq$) if:

- there is a *least element* $\bot$ such that $\bot \leq x$ for all $x \in D$, and

- for any two $x, y \in D$, there is a *meet* (*a.k.a intersection*) $x \wedge y$ which is the *greatest lower bound* of x and y, *i.e.,*

$$x \wedge y \leq x \quad x \wedge y \leq y \quad \forall z \in D,\ z \leq x \text{ and } z \leq y \text{ implies } z \leq x \wedge y$$

Additionally, D is a *(binary) lattice* with respect to a partial order $(\leq)$ if:

- there is a *greatest element* $\top$ such that $x \leq \top$ for all $x \in D$, and
- for any two $x, y \in D$ there is a *join* (*a.k.a union*) $x \vee y$ which is the *least upper bound* of x and y, *i.e.,*

$$x \leq x \vee y \quad y \leq x \vee y \quad \forall z \in D,\ x \leq z \text{ and } y \leq z \text{ implies } x \vee y \leq z$$

A *complete lattice* generalizes the binary operators over any (finite or infinite) set of such elements.

Example 4.2.2. Sets ordered by subset inclusion ($A \subseteq B$) form a lower semi-lattice with empty set $\{\}$ as the least element and the usual set intersection operation $A \cap B$.

Given any set U, subsets of U form a lattice where, in addition to the least $\{\}$ and intersection $A \cap B$, there is a greatest set U and union $A \cup B \subseteq U$ for all $A \subseteq U$ and $B \subseteq U$.

Definition 4.2.3 (Refinement & Subtype Lattices)**.** There are two complete lattices over type pre-candidates: one with respect to refinement ($\mathbb{A} \sqsubseteq \mathbb{B}$) and one with respect to subtyping ($\mathbb{A} \leq \mathbb{B}$). The binary refinement union/intersection ($\sqcup$, $\sqcap$) and subtyping union/intersection ($\vee$,$\wedge$) are:

$$\begin{aligned}
(\mathbb{A}^+, \mathbb{A}^-) \sqcup (\mathbb{B}^+, \mathbb{B}^-) &= (\mathbb{A}^+ \cup \mathbb{B}^+, \mathbb{A}^- \cup \mathbb{B}^-)\\
(\mathbb{A}^+, \mathbb{A}^-) \vee (\mathbb{B}^+, \mathbb{B}^-) &= (\mathbb{A}^+ \cup \mathbb{B}^+, \mathbb{A}^- \cap \mathbb{B}^-)\\
(\mathbb{A}^+, \mathbb{A}^-) \sqcap (\mathbb{B}^+, \mathbb{B}^-) &= (\mathbb{A}^+ \cap \mathbb{B}^+, \mathbb{A}^- \cap \mathbb{B}^-)\\
(\mathbb{A}^+, \mathbb{A}^-) \wedge (\mathbb{B}^+, \mathbb{B}^-) &= (\mathbb{A}^+ \cap \mathbb{B}^+, \mathbb{A}^- \cup \mathbb{B}^-)
\end{aligned}$$

The pre-candidates $(\{\}, \{\})$ and $(\mathit{Term}, \mathit{Cont})$ are the least and greatest pre-candidates, respectively, with respect to refinement.

The pre-candidates $(\{\}, \mathit{Cont})$ and $(\mathit{Term}, \{\})$ are the least and greatest pre-candidates, respectively, with respect to subtyping.

Property 4.2.4 (de Morgan)**.** *For all pre-candidates* $\mathbb{A}$ *and* $\mathbb{B}$*:*

1. $(\mathbb{A} \sqcup \mathbb{B})^{\perp\!\!\perp} = (\mathbb{A}^{\perp\!\!\perp}) \sqcap (\mathbb{B}^{\perp\!\!\perp})$
2. $(\mathbb{A} \sqcap \mathbb{B})^{\perp\!\!\perp} \sqsupseteq (\mathbb{A}^{\perp\!\!\perp}) \sqcup (\mathbb{B}^{\perp\!\!\perp})$

Property 4.2.5 (Sound & Complete Refinement Semi-Lattices)**.** *The lower refinement semi-lattice over pre-candidates preserves* soundness *and the upper refinement semi-lattice preserves* completeness. *Specifically, for any pre-candidates* $\mathbb{A}$ *and* $\mathbb{B}$*:*

1. $(\{\}, \{\})$ *is (trivially) sound,*

2. *if* $\mathbb{A}$ *and* $\mathbb{B}$ *are sound, then so is* $\mathbb{A} \sqcap \mathbb{B}$*,*

3. $(\mathit{Term}, \mathit{Cont})$ *is (trivially) complete, and*

4. *if* $\mathbb{A}$ *and* $\mathbb{B}$ *are complete, then so is* $\mathbb{A} \sqcup \mathbb{B}$*.*

Property 4.2.6 (Sound Subtyping Lattice)**.** *The complete lattice over pre-candidates preserves* soundness *(in both directions), but does not necessarily preserve* completeness *(in both directions).*

$\mathbb{A} \vee \mathbb{B}$ might be missing some terms which could be soundly included, and $\mathbb{A} \wedge \mathbb{B}$ might be missing some continuations.

Definition 4.2.7. There is a complete lattice over (sound and complete) $⫫$-orthogonal candidates with respect to subtype order, with the binary union ($\curlyvee$) and intersection ($\curlywedge$) between any $⫫$-orthogonal candidates $\mathbb{A} = (\mathbb{A}^+, \mathbb{A}^-)$ and $\mathbb{B} = (\mathbb{B}^+, \mathbb{B}^-)$

$$\mathbb{A} \curlyvee \mathbb{B} = (\mathbb{A} \vee \mathbb{B})^{⫫⫫} = \mathrm{Pos}(\mathbb{A}^+ \cup \mathbb{B}^+)$$
$$\mathbb{A} \curlywedge \mathbb{B} = (\mathbb{A} \wedge \mathbb{B})^{⫫⫫} = \mathrm{Neg}(\mathbb{A}^- \cup \mathbb{B}^-)$$

and the least ($\emptyset$) and greatest ($\mathbb{T}$) elements

$$\emptyset = \mathrm{Pos}\{\} = (\mathit{Cont}^{⫫}, \mathit{Cont}) \qquad \mathbb{T} = \mathrm{Neg}\{\} = (\mathit{Term}, \mathit{Term}^{⫫})$$

Proof. Using de Morgan laws of orthogonality, along with the fact that $⫫$-orthogonal candidates are fixed points of $\_^{⫫}$. Notice that for any $\mathbb{A} = \mathbb{A}^{⫫}$ and $\mathbb{B} = \mathbb{B}^{⫫}$, we have

$$\mathbb{A} \curlywedge \mathbb{B} \leq \mathbb{A} \wedge \mathbb{B} \leq \mathbb{A}, \mathbb{B} \leq \mathbb{A} \vee \mathbb{B} \leq \mathbb{A} \curlyvee \mathbb{B}$$

where

$$\mathbb{A} \curlywedge \mathbb{B} = (\mathbb{A} \curlywedge \mathbb{B})^{⫫} \qquad \mathbb{A} \curlyvee \mathbb{B} = (\mathbb{A} \curlyvee \mathbb{B})^{⫫}$$

but it may be the case that

$$\mathbb{A} \wedge \mathbb{B} \neq (\mathbb{A} \wedge \mathbb{B})^{⫫} \qquad \mathbb{A} \vee \mathbb{B} \neq (\mathbb{A} \vee \mathbb{B})^{⫫}$$ □

This lattice gives us a semantic interpretation of intersection and union types, where

$$[\![A \wedge B]\!] = [\![A]\!] \curlywedge [\![B]\!] \qquad [\![A \vee B]\!] = [\![A]\!] \curlyvee [\![B]\!]$$

with all the correct subtyping properties.

4.3 Recursive types — induction and coinduction

4.3.1 Variations on induction

How to define the set of numbers?

Incremental Kleene fixed point

Kleene fixed point construction:

1. start with the empty set $\{\}$,
2. at each step, build on the previous step (add 0 and the successor of every number known before), then
3. the inductively-defined set is the limit (the union) of all the finite approximations generated from steps 1 and 2.

$$\begin{aligned}
\mathbb{N}_0 &= \{\} \\
\mathbb{N}_1 &= \{0\} \\
\mathbb{N}_2 &= \{0, 1\} \\
\mathbb{N}_3 &= \{0, 1, 2\} \\
&\vdots \\
\mathbb{N}_{i+1} &= \{0\} \cup \{n + 1 \mid n \in \mathbb{N}_i\} \\
&\vdots \\
\mathbb{N}_\infty &= \bigcup_{i=0}^{\infty} \mathbb{N}_i
\end{aligned}$$

$$\begin{aligned}
\mathbb{N}_{+1}(X) &= \{0\} \cup \{n + 1 \mid n \in X\} \\
\mathbb{N}_{i+1} &= \mathbb{N}_{+1}(\mathbb{N}_i) \\
\mathbb{N}_i &= \mathbb{N}_{+1}^{i}\{\}
\end{aligned}$$

This works because $\mathbb{N}_{+1}$ is *monotonic*, for all $X \subseteq Y$, $\mathbb{N}_{+1}(X) \subseteq \mathbb{N}_{+1}(Y)$. Therefore, $\mathbb{N}_0 \subseteq \mathbb{N}_1 \subseteq \mathbb{N}_2 \subseteq \cdots \subseteq \mathbb{N}_i \subseteq \mathbb{N}_{i+1}$.

Kleene-construction of an inductive type of numbers:

$$\begin{aligned}
\mathbb{N}_0 &= \mathrm{Pos}\{\} = \emptyset \\
\mathbb{N}_{i+1} &= \mathrm{Pos}\left(\{\mathrm{Zero}\} \cup \{\mathrm{Succ}\, v \mid v \in \mathbb{N}_i\}\right) \\
\mathbb{N}_\infty &= \bigcurlyvee_{i=0}^{\infty} \mathbb{N}_i
\end{aligned}$$

works because

$$\begin{aligned}\mathbb{N}_{+1}(\mathbb{A}) &= \mathrm{Pos}\left(\{\mathrm{Zero}\} \cup \{\mathrm{Succ}\, v \mid v \in \mathbb{A}\}\right)\\ \mathbb{N}_{i+1} &= \mathbb{N}_{+1}(\mathbb{N}_i)\\ \mathbb{N}_i &= \mathbb{N}^{i}_{+1}(\emptyset)\end{aligned}$$

and $\mathbb{N}_{+1}$ is *monotonic* with respect to subtyping (not refinement!) — for all $\mathbb{A} \leq \mathbb{B}$, $\mathbb{N}_{+1}(\mathbb{A}) \leq \mathbb{N}_{+1}(\mathbb{B})$ — so that

$$\mathbb{N}_0 \leq \mathbb{N}_1 \leq \mathbb{N}_2 \leq \cdots \leq \mathbb{N}_{i+1} \leq \cdots \leq \mathbb{N}_\infty$$

Inductive sized types

Directly naming types for each of these approximations gives a direct interpretation of *sized types*, where the i-indexed family $\mathrm{Nat}\, i$ corresponds to natural numbers *strictly less than* size i:

$$\begin{aligned}&\textbf{data}\,\mathrm{Nat} : \mathrm{Size} \to \mathrm{Type}\,\textbf{where}\\ &\quad \mathrm{Zero} : \mathrm{Nat}\ \ (i+1)\\ &\quad \mathrm{Succ} : \forall i{:}\,\mathrm{Size}\,.\ \mathrm{Nat}\ \ i \to \mathrm{Nat}\ \ (i+1)\end{aligned}$$

$$\llbracket \mathrm{Nat}\, i \rrbracket = \mathbb{N}_i \qquad\qquad \text{(if } i \text{ is a size index)}$$

$$\llbracket \mathrm{AnyNat} \rrbracket = \llbracket \exists i{:}\,\mathrm{Size}\,.\ \mathrm{Nat}\, i \rrbracket = \curlyvee_{i=0}^{\infty} \mathbb{N}_i = \mathbb{N}_\infty$$

Exercise 4.3.1. There are a few variations on this kind of definition. You could tighten this to an type describing the number of the *exact* size measured

$$\begin{aligned}&\textbf{data}\,\mathrm{Nat}^{=} : \mathrm{Size} \to \mathrm{Type}\,\textbf{where}\\ &\quad \mathrm{Zero} : \mathrm{Nat}^{=}\, 0\\ &\quad \mathrm{Succ} : \forall i{:}\,\mathrm{Size}\,.\ \mathrm{Nat}^{=}\, i \to \mathrm{Nat}^{=}\ \ (i+1)\end{aligned}$$

or you could further generalize to *strong induction* over the size (no longer an indexed family)

$$\begin{aligned}&\textbf{data}\,\mathrm{Nat}^{<}(i : \mathrm{Size}) : \mathrm{Type}\,\textbf{where}\\ &\quad \mathrm{Zero} : \mathrm{Nat}^{<}\, i\\ &\quad \mathrm{Succ} : \forall j < i.\ \mathrm{Nat}^{<}\, j \to \mathrm{Nat}^{<}\, i\end{aligned}$$

Write down the definition of the $\perp\!\!\!\perp$-orthogonal candidates for the instances of $\mathrm{Nat}^{=}$ and $\mathrm{Nat}^{<}$ at each i.

Suppose $v \in \curlyvee_{i=0}^{\infty} \mathbb{N}_i$. Is there some specific approximation, n, such that $v \in \mathbb{N}_n$? Maybe not!

$$\curlyvee_{i=0}^{\infty} \mathbb{N}_i = \left(\bigvee_{i=0}^{\infty} \mathbb{N}_i\right)^{\perp\!\!\!\perp\,\perp\!\!\!\perp} = \mathrm{Pos}\left(\bigcup_{i=0}^{\infty} \mathbb{N}_i^{+}\right)$$

In particular it may be possible (depending on $\perp\!\!\!\perp$ and the expressions in the language), that a term which unfolds to infinite successors $\mathrm{Succ}(\mathrm{Succ}(\mathrm{Succ}(\mathrm{Succ}(\dots))))$ is in the limit $\bigcurlyvee_{i=0}^{\infty} \mathbb{N}_i$ due to completeness, even though it is *not* in any of the finite approximations $\mathbb{N}_i$.

Direct Knaster-Tarski fixed point

Goal: be able to do standard induction on only the *finite* canonical values — *e.g.,* only consider the Zero and $\mathrm{Succ}\, v$ case for a previously-known v — even if the limit of the whole type might include other weird, non-canonical values.

Knaster-Tarski fixed point solution: the greatest lower bound of over-approximations

$$\mathbb{N} = \bigcurlywedge \{\mathbb{A} \in \perp\!\!\!\perp\text{-orthgonal candidates} \mid \mathrm{Zero} \in \mathbb{A} \text{ and } v \in \mathbb{A} \text{ implies } \mathrm{Succ}\, v \in \mathbb{A}\}$$

We know that

- $\mathrm{Zero} \in \mathbb{N}$, and
- $\mathrm{Succ}\, v \in \mathbb{N}$ for all $v \in \mathbb{N}$

because they are in all (over)-approximations $\mathbb{A}$.

But even if there are other terms in $\mathbb{N}$, we can still do induction!

Suppose

$$\mathrm{Zero} \perp\!\!\!\perp E \qquad\qquad v \perp\!\!\!\perp E \text{ implies } \mathrm{Succ}\, v \perp\!\!\!\perp E$$

We have the $\perp\!\!\!\perp$-orthogonal candidate $\mathrm{Neg}\{E\}$ — where $E \in \mathrm{Neg}\{E\}$ from double orthogonal introduction — such that

- $\mathrm{Zero} \in \mathrm{Neg}\{E\}$, and
- $\mathrm{Succ}\, v \in \mathrm{Neg}\{E\}$ for all $v \in \mathrm{Neg}\{E\}$.

That means

$$\mathbb{N} \leq \mathrm{Neg}\{E\}$$

and thus

$$E \in \mathbb{N}$$

by definition of $\leq$ on pre-candidates (*Hint:* subtyping flows backwards for continuations, where $\mathbb{A} \leq \mathbb{B}$ means all continuations of $\mathbb{B}^-$ must also be found in $\mathbb{A}^-$).

In other words, E only needs to consider the canonical cases of numbers — Zero and $\mathrm{Succ}\, v$ assuming v already works — even though $\mathbb{N}$ may have many other non-canonical terms.

4.3.2 Variations on coinduction

Because $\perp\!\!\!\perp$-orthogonal candidates are naturally dual, coinduction works by just flipping the roles between the term side and continuation side.

Incremental Kleene fixed point

Streams can be seen as a codata type

$$\begin{aligned}
&\textbf{codata}\,\mathrm{Stream}\ \ (A : \mathrm{Type}) : \mathrm{Type}\,\textbf{where}\\
&\qquad \mathrm{Head} : \mathrm{Stream}\,A \to A\\
&\qquad \mathrm{Tail} : \mathrm{Stream}\,A \to \mathrm{Stream}\,A
\end{aligned}$$

You can incrementally build up the set of all basic stream projections similar to the plain set of natural numbers as:

$$\begin{aligned}
\mathbb{S}_0(\mathbb{A}) &= \{\}\\
\mathbb{S}_1(\mathbb{A}) &= \{\mathrm{Head}\,E \mid E \in \mathbb{A}\}\\
\mathbb{S}_2(\mathbb{A}) &= \{\mathrm{Head}\,E \mid E \in \mathbb{A}\} \cup \{\mathrm{Tail}(\mathrm{Head}\,E) \mid E \in \mathbb{A}\}\\
\mathbb{S}_2(\mathbb{A}) &= \{\mathrm{Head}\,E \mid E \in \mathbb{A}\} \cup \{\mathrm{Tail}(\mathrm{Head}\,E) \mid E \in \mathbb{A}\} \cup \{\mathrm{Tail}(\mathrm{Tail}(\mathrm{Head}\,E)) \mid E \in \mathbb{A}\}\\
&\ \ \vdots\\
\mathbb{S}_{i+1}(\mathbb{A}) &= \mathbb{S}_i(\mathbb{A}) \cup \{\mathrm{Tail}\,E \mid E \in \mathbb{S}_i(\mathbb{A})\}\\
&\ \ \vdots\\
\mathbb{S}(\mathbb{A}) &= \bigcup_{i=0}^{\infty} \mathbb{S}_i = \{\mathrm{Tail}^i(\mathrm{Head}\,E) \mid i \in \mathbb{N}, E \in \mathbb{A}\}
\end{aligned}$$

But this is woefully incomplete to describe the semantics of a type like $\mathrm{Stream}\,A$.

Trick: build a negative $\perp\!\!\!\perp$-orthogonal candidate, perfectly symmetric to the method used for Nat.

The Kleene-construction of a coinductive type of streams:

$$\begin{aligned}
\mathbb{S}_0(\mathbb{A}) &= \mathrm{Neg}\{\} = \mathbb{T}\\
\mathbb{S}_{i+1}(\mathbb{A}) &= \mathrm{Neg}\left(\{\mathrm{Head}\,E \mid E \in \mathbb{A}\} \cup \{\mathrm{Tail}\,E \mid E \in \mathbb{S}_i(\mathbb{A})\}\right)\\
\mathbb{S}_\infty(\mathbb{A}) &= \bigcurlywedge_{i=0}^{\infty} \mathbb{S}_i(\mathbb{A})
\end{aligned}$$

works because

$$\begin{aligned}
\mathbb{S}_{+1}(\mathbb{A})(\mathbb{B}) &= \mathrm{Neg}\left(\{\mathrm{Head}\,E \mid E \in \mathbb{A}\} \cup \{\mathrm{Tail}\,E \mid E \in \mathbb{B}\}\right)\\
\mathbb{S}_{i+1}(\mathbb{A}) &= \mathbb{S}_{+1}(\mathbb{A})(\mathbb{S}_i(\mathbb{A}))\\
\mathbb{S}_i(\mathbb{A}) &= \mathbb{S}_{+1}(\mathbb{A})^i(\mathbb{T})
\end{aligned}$$

and $\mathbb{S}_{+1}(\mathbb{A})$ is *monotonic* with respect to subtyping — for all $\mathbb{B} \leq \mathbb{C}$, $\mathbb{S}_{+1}(\mathbb{A})(\mathbb{B}) \leq \mathbb{S}_{+1}(\mathbb{A})(\mathbb{C})$. So that,

$$\mathbb{S}_0(\mathbb{A}) \geq \mathbb{S}_1(\mathbb{A}) \geq \mathbb{S}_2(\mathbb{A}) \geq \cdots \geq \mathbb{S}_{i+1}(\mathbb{A}) \geq \cdots \geq \mathbb{S}_\infty(\mathbb{A})$$

Sized coinductive types

Approximations $\mathbb{S}_i(\mathbb{A})$ limit the size of the *observations* you can make on a stream (rather than on the stream itself) — after a certain depth of projection, there is no longer any constraint on how the stream might respond. Sized types let you directly naming these finite approximations. The i-indexed family Stream $A\ i$ corresponds to the streams that correctly respond to projections *strictly less than* size i:

$$\begin{aligned}&\textbf{codata}\, \mathrm{Stream}\, A : \mathrm{Size} \to \mathrm{Type}\, \textbf{where}\\ &\quad \mathrm{Head} : \forall i{:}\, \mathrm{Size}\,.\ \mathrm{Stream}\, A\ (i+1) \to A\\ &\quad \mathrm{Tail} : \forall i{:}\, \mathrm{Size}\,.\ \mathrm{Stream}\, A\ (i+1) \to \mathrm{Stream}\, A\ i\end{aligned}$$

$$\begin{aligned}[\![\mathrm{Stream}\, A\ i]\!] &= \mathbb{S}_i(\mathbb{A})\\ [\![\mathrm{InfStream}\, A]\!] &= [\![\forall i{:}\, \mathrm{Size}\,.\ \mathrm{Stream}\, A\ i]\!] = \bigcurlywedge_{i=0}^{\infty} \mathbb{S}_i([\![A]\!]) = \mathbb{S}_\infty([\![A]\!])\end{aligned}$$

Exercise 4.3.2. As with sized induction, there are a few variations on this kind of sized coinductive codata type. You could tighten the type of streams to describe a measurement of the *exact* size of depth you are allowed to look into the stream (no more, no less) as

$$\begin{aligned}&\textbf{codata}\, \mathrm{Stream}^{=}\, A : \mathrm{Size} \to \mathrm{Type}\, \textbf{where}\\ &\quad \mathrm{Head} : \mathrm{Stream}^{=}\, A\ 0 \to A\\ &\quad \mathrm{Tail} : \forall i{:}\, \mathrm{Size}\,.\, \mathrm{Stream}^{=}\, A\ (i+1) \to \mathrm{Stream}^{=}\, A\ i\end{aligned}$$

or you could further generalize to *strong coinduction* over the depth of tail projections (no longer an indexed family)

$$\begin{aligned}&\textbf{codata}\, \mathrm{Stream}^{<}\ (A : \mathrm{Type})\ (i : \mathrm{Size}) : \mathrm{Type}\, \textbf{where}\\ &\quad \mathrm{Head} : \mathrm{Stream}^{<}\, A\ i \to A\\ &\quad \mathrm{Tail} : \forall j < i.\ \mathrm{Stream}^{<}\, A\ i \to \mathrm{Stream}^{<}\, A\ j\end{aligned}$$

Again, for certain design decisions (choices of $\perp\!\!\!\perp$ and the machine language in question), it may be possible to program infinite loops, which will endlessly inspect deeper into a stream without end, corresponding to the projection Tail(Tail(Tail(. . .))) that end up the limit $\bigcurlywedge_{i=1}^{\infty} \mathbb{S}_i(\mathbb{A})$, even though it is *not* in any of the finite approximations $\mathbb{S}_i(\mathbb{A})$ leading up to it.

Direct Knaster-Tarski fixed point

Goal: be able to do coinduction on only the *finite* canonical observations — *e.g.,* only consider the $\mathrm{Head}\,E$ (for an E expecting an element) and $\mathrm{Tail}\,E$ (for a previously-known E) destructors — even if the limit of the whole type might include other weird, non-canonical observations.

The dual Knaster-Tarski fixed point solution: the least upper bound of under approximations:

$$\mathbb{S}(\mathbb{A}) = \curlyvee \{\mathbb{B} \in \perp\!\!\!\perp\text{-orthogonal candidates} \mid \begin{array}{l} \mathrm{Head}\,E \in \mathbb{B} \text{ for all } E \in \mathbb{A} \text{ and} \\ \mathrm{Tail}\,E \in \mathbb{B} \text{ for all } E \in \mathbb{B}\} \end{array}$$

We know that

- $\mathrm{Head}\,E \in \mathbb{S}(\mathbb{A})$ for all $E \in \mathbb{A}$, and
- $\mathrm{Tail}\,E \in \mathbb{S}(\mathbb{A})$ for all $E \in \mathbb{S}(\mathbb{A})$

because they are in all (under)-approximations $\mathbb{B}$.

But we can still do coinduction (*e.g.,* generate streams incrementally by their Head and Tail) even if there are other "weird" observations!

Suppose

$$E \in \mathbb{A} \text{ implies } v \qquad \perp\!\!\!\perp \mathrm{Head}\,E v \perp\!\!\!\perp E \qquad \text{implies } v \perp\!\!\!\perp \mathrm{Tail}\,E$$

We have the $\perp\!\!\!\perp$-orthogonal candidate $\mathrm{Pos}\{v\}$ — where $v \in \mathrm{Pos}\{v\}$ from double orthogonal introduction — such that

- $\mathrm{Head}\,E \in \mathrm{Pos}\{v\}$ for all $E \in \mathbb{A}$, and
- $\mathrm{Tail}\,E \in \mathrm{Pos}\{v\}$ for all $E \in \mathrm{Pos}\{v\}$.

That means

$$\mathrm{Pos}\{v\} \leq \mathbb{S}(\mathbb{A})$$

and thus

$$v \in \mathbb{S}(\mathbb{A})$$

by definition of $\leq$ on pre-candidates.

In other words, v only needs to consider the canonical observations on streams — $\mathrm{Head}\,E$ when E expects an element from $\mathbb{A}$ and $\mathrm{Tail}\,E$ assuming E already works — even though $\mathbb{S}(\mathbb{A})$ may have many other non-canonical ways to observe streams.

4.4 Evaluation order — producers and consumers

Thus far, *all* consumers (continuations) and producers (terms) have been *substitutable*:

- A variable x may be substituted by *any* term v (according to the call-by-name evaluation order).
- A covariable α may be substituted by *any* continuation E, which corresponds *exactly* to the call-by-name evaluation contexts. This is because every E that I can write is *strict* (it immediately uses its input).

Many other systems — with other evaluation strategies — need a more sophisticated definition than $⫫$-orthogonality ($\mathbb{A} = \mathbb{A}^{⫫}$) to ensure that soundness and completeness give the properties you need.

4.4.1 Non-strictness in call-by-name

Suppose we want to add a **let**-binding to our language.

$$\langle \mathbf{let}\, x = N\, \mathbf{in}\, M \| E\rangle \mapsto \langle N \| \mathbf{let}\, x\, \mathbf{in}\, \langle M \| E\rangle\rangle$$

This is the *dual of* μ: a **let** names its input, the same way that μ names its continuation. So we could compile **let** to $\tilde{\mu}$ in honor of this duality — writing **let** x **in** c as $\tilde{\mu}x.c$ — so that

$$\begin{aligned}(\mathbf{let}\, x = N\, \mathbf{in}\, M) &= \mu\alpha.\langle N \| \tilde{\mu}x.\langle M \| \alpha\rangle\rangle \\ \langle v \| \tilde{\mu}x.c\rangle &\mapsto c[v/x] \qquad\qquad (\tilde{\mu})\end{aligned}$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}\, ActR \qquad \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c : A \vdash \Delta}\, ActL$$

and there is a generalized syntax e of *non-strict* continuations,

$$\begin{aligned}
&c ::= \langle v \| e\rangle \\
&e ::= E \mid \tilde{\mu}x.e \qquad\qquad E ::= \alpha \mid v \cdot E \mid \mathbf{if\, then}\, c\, \mathbf{else}\, c' \\
&v ::= x \mid \mu\alpha.c \mid \lambda x.v \mid \mathrm{True} \mid \mathrm{False}
\end{aligned}$$

With the addition of $\tilde{\mu}$ — with its operational reduction and typing rules — you can no longer show that the interpretation of a A has all the well-typed terms. In particular, you may know that

$$\text{for all } v \in \{\mathrm{True}, \mathrm{False}\},\ ⫫ \ni c[v/x] \leftarrow\!\shortmid \langle v \| \tilde{\mu}x.c\rangle \in ⫫$$

so that $\tilde{\mu}x.c \in [\![\mathrm{Bool}]\!]^- = \{\mathrm{True}, \mathrm{False}\}^{⫫}$. But now, knowing only that

$$\text{for all } E \in [\![\mathrm{Bool}]\!]^-,\ ⫫ \ni c'[E/\alpha] \leftarrow\!\shortmid \langle \mu\alpha.c' \| E\rangle \in ⫫$$

is not enough to show that $\mu\alpha.c' \perp\!\!\!\perp \mu x.c$, so completeness isn't strong enough to conclude that $\mu\alpha.c' \in [\![\mathrm{Bool}]\!]^+$, even though it should be because

$$c'[\tilde{\mu}x.c/\alpha] \not\mapsfrom \langle \mu\alpha.c' \| \tilde{\mu}x.c \rangle \mapsto c[\mu\alpha.c'/x]$$

In general, the naïve positive candidate

$$\mathrm{Pos}(\mathbb{C}) = (\mathbb{C}^{\perp\!\!\!\perp\perp\!\!\!\perp}, \mathbb{C}^{\perp\!\!\!\perp})$$

might not let you show that it includes some terms which are safe with all strict (*i.e.,* substitutable) continuations in $\mathbb{C}^{\perp\!\!\!\perp}$, but we don't know anything about the interaction with non-strict continuation which seize control of the command.

4.4.2 Computation in call-by-value

Suppose instead we want to study call-by-value evaluation. In call-by-value, there are non-value terms that we want to evaluate first, *instead* of substituting them into a variable. A call-by-value version of the abstract machine looks like:

$$\begin{aligned}
c &::= \langle v \| E \rangle \\
v &::= V \mid \mu\alpha.c & \qquad\qquad V &::= x \mid \lambda x.v \mid \mathrm{True} \mid \mathrm{False} \\
E &::= \alpha \mid \tilde{\mu}x.c \mid V \cdot E \mid \mathbf{if\,then}\,c\,\mathbf{else}\,c'
\end{aligned}$$

$$\begin{aligned}
\langle \mu\alpha.c \| E \rangle &\mapsto c[E/\alpha] & (\mu) \\
\langle V \| \tilde{\mu}x.c \rangle &\mapsto c[V/x] & (\tilde{\mu}) \\
\langle \lambda x.v \| V \cdot E \rangle &\mapsto \langle v[V/x] \| E \rangle & (\beta_\to) \\
&\textit{same as before} & (\beta_{\mathrm{Bool}})
\end{aligned}$$

The same problem happens, where now you can have non-values in a negative type like $[\![A \to B]\!]$, where

$$\text{for all } E \in [\![A \to B]\!]^-,\ \perp\!\!\!\perp \ni \langle \mu\alpha.c \| E \rangle \mapsfrom c[E/\alpha] \in \perp\!\!\!\perp$$

but now there can be non-canonical continuations like $\tilde{\mu}x.c'$ which should be in $[\![A \to B]\!]$ because

$$\text{for all } V \in [\![A \to B]\!]^+,\ \perp\!\!\!\perp \ni \langle V \| \tilde{\mu}x.c' \rangle \mapsfrom c'[V/x] \in \perp\!\!\!\perp$$

but completeness can't prove that $\tilde{\mu}x.c' \in [\![A \to B]\!]^-$ because

$$c[\tilde{\mu}x.c'/\alpha] \mapsfrom \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \not\mapsto c'[\mu\alpha.c/x]$$

In general, the naïve negative candidate

$$\mathrm{Neg}(\mathbb{C}) = (\mathbb{C}^{\perp\!\!\!\perp}, \mathbb{C}^{\perp\!\!\!\perp\perp\!\!\!\perp})$$

might not let you show that it includes some continuations which are safe with all immediate (*i.e.,* substitutable) values in $\mathbb{C}^{\perp\!\!\!\perp}$, but we don't know anything about the interaction with non-value computations which seize control of the command.

4.4.3 Sharing in call-by-need

Call-by-need reduction delays performing computations until they are "needed," but when they are needed it shares the work performed by the computation by reusing the returned value in the future.

Call-by-need evaluation has *both* of the problems above, because it has *both*

- non-substitutable terms ($\mu\alpha.c$) representing computations which can't be copied because we need to remember to copy the first value it returns to all future observers, and
- non-substitutable, continuations ($\tilde{\mu}x.c$) representing non-strict consumers that don't need their input yet.

As such, both $\mathrm{Pos}(\mathbb{C}^+)$ and $\mathrm{Neg}\,\mathbb{C}^-$, as defined above, provides a notion of completeness that is too weak in practice.

4.4.4 Non-determinism in unrestricted reduction

In contrast, we could try removing the restriction on the μ and $\tilde{\mu}$ reduction rules,

$$\begin{aligned} \langle \mu\alpha.c \| e\rangle &\mapsto c[e/\alpha] && (\mu) \\ \langle v \| \tilde{\mu}x.c\rangle &\mapsto c[v/x] && (\tilde{\mu}) \end{aligned}$$

so that the critical pair

$$c[\tilde{\mu}x.c'/\alpha] \leftarrow\!\shortmid \langle \mu\alpha.c \| \tilde{\mu}x.c'\rangle \mapsto c'[\mu\alpha.c/x]$$

can choose to go in *either* direction.

This causes a different sort of problem. Now, even if you know that

$$\text{for all } e \in [\![A]\!]^-,\ \perp\!\!\!\perp \ni \langle \mu\alpha.c \| e\rangle \leftarrow\!\shortmid c[e/\alpha] \in \perp\!\!\!\perp$$

you might find a counter-example to safety by following the other reduction path, where there is a $\tilde{\mu}x.c' \in [\![A]\!]^-$ such that

$$\perp\!\!\!\perp \ni c[\tilde{\mu}x.c'/\alpha] \leftarrow\!\shortmid \langle \mu\alpha.c \| \tilde{\mu}x.c'\rangle \mapsto c'[\mu\alpha.c/x] \notin \perp\!\!\!\perp$$

4.4.5 Strengthening completeness — the (co)value restriction

Suppose that there are some chosen subset of terms called *values*, and some subset of continuations called *covalues* (read as your choice of either "continuation values" or "the dual of values"),

$$\mathit{Value} \subseteq \mathit{Term} \qquad\qquad \mathit{CoValue} \subseteq \mathit{Continuation}$$

such that any *Value* contains exactly the terms substitutable by $\tilde{\mu}$ reduction and *CoValue* contains exactly the continuations substitutable by the μ rule.

Idea: completeness shouldn't require that you prove something works with *all* (potentially non-substitutable, non-strict, non-(co)value) opposing sides. Instead, you should only have to show that a potential term/continuation is safe with just the substitutable covalues/values it might interact with.

The (co)value restriction on a pre-candidate $\mathbb{A}$ is

$$\mathbb{A}^v = \mathbb{A} \sqcap (\mathit{Value}, \mathit{CoValue})$$

so that $\mathbb{A}^v$ keeps only the (co)values from $\mathbb{A}$.

Property 4.4.1.

1. Idempotency: $\mathbb{A}^{vv} = \mathbb{A}^v$
2. Refinement: $\mathbb{A}^v \sqsubseteq \mathbb{A}$
3. $\perp\!\!\!\perp$-Extension: $\mathbb{A}^{\perp\!\!\!\perp} \sqsubseteq \mathbb{A}^{v\perp\!\!\!\perp}$
4. Restricted double orthogonal introduction: $\mathbb{A}^v \sqsubseteq \mathbb{A}^{v\perp\!\!\!\perp v\perp\!\!\!\perp v}$
5. Restricted triple orthogonal elimination: $\mathbb{A}^{v\perp\!\!\!\perp v\perp\!\!\!\perp v\perp\!\!\!\perp v} = \mathbb{A}^{v\perp\!\!\!\perp v}$

Proof. Properties 1 and 2 follow from the definition of $\mathbb{A}^v$ (as a greatest lower bound) and 3 follows from contrapositive.

The final properties are left as an exercise for the reader. □

Definition 4.4.2 (Orthogonal Candidate)**.** A *(co)value restricted $\perp\!\!\!\perp$-orthogonal candidate* is any pre-candidate $\mathbb{A}$ such that $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp} = \mathbb{A}^{v\perp\!\!\!\perp}$.

In other words, a (co)value restricted $\perp\!\!\!\perp$-orthogonal candidate has the same soundness property as ordinary $\perp\!\!\!\perp$-orthogonal candidates, as well as this strengthened completeness property:

- $v \in \mathbb{A}^+$ whenever $v \perp\!\!\!\perp E$ for all $E \in \mathbb{A}^{v-}$, equivalent to the fact that $\mathbb{A}^{-v\perp\!\!\!\perp} \subseteq \mathbb{A}^+$, and
- $e \in \mathbb{A}^-$ whenever $V \perp\!\!\!\perp e$ for all $V \in \mathbb{A}^{v+}$, equivalent to the fact that $\mathbb{A}^{+v\perp\!\!\!\perp} \subseteq \mathbb{A}^-$

Notice that because $\perp\!\!\!\perp$-extension, $\mathbb{A}^{\perp\!\!\!\perp} \sqsubseteq \mathbb{A}^{v\perp\!\!\!\perp}$, always holds, the interesting informational content of the double-fixed point $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp} = \mathbb{A}^{v\perp\!\!\!\perp}$ is

$$\mathbb{A}^{v\perp\!\!\!\perp} \sqsubseteq \mathbb{A} \sqsubseteq \mathbb{A}^{\perp\!\!\!\perp}$$

In other words, these double-fixed points guarantee the same (strong) soundness property before which ensures safety of *any* combination between the two sides of $\mathbb{A}$, but the stronger completeness property means we only need to show that terms/continuations are safe with respect to (co)values of $\mathbb{A}^v$ in order to prove they are in $\mathbb{A}$.

How do we make these double-fixed points? Can generalize the positive/negative construction of a candidate from an initial set of canonical constructions $\mathbb{C}^+ \subseteq$ *Value* or canonical observations $\mathbb{C}^- \subseteq$ *CoValue* as:

$$\mathrm{Pos}^v(\mathbb{C}^+) = (\mathbb{C}^+, \mathbb{C}^{+\perp\!\!\!\perp})^{v\perp\!\!\!\perp v\perp\!\!\!\perp} \qquad \mathrm{Neg}^v(\mathbb{C}^-) = (\mathbb{C}^{-\perp\!\!\!\perp}, \mathbb{C}^-)^{v\perp\!\!\!\perp v\perp\!\!\!\perp}$$

This definition works for positive definitions in call-by-name and negative definitions in call-by-value. It also works for other evaluation strategies like call-by-need (or its dual).

Moreover, this “generalized” definition is equal to the simple ones in the most auspicious circumstances

Property 4.4.3.

- *Whenever CoValue = Cont (as in call-by-value),* $\mathrm{Pos}^v(\mathbb{C}^+) = \mathrm{Pos}(\mathbb{C}^+)$*.*
- *Whenever Value = Term (as in call-by-name),* $\mathrm{Neg}^v(\mathbb{C}^-) = \mathrm{Neg}(\mathbb{C}^-)$*.*

4.4.6 Strengthening soundness — symmetric fixed points

To handle non-determinism, we can only reason about terms based on what actions they are responsible for — reductions induced by only the continuation cannot be known. Dually, to prove properties about continuations, we should only have to describe what behaviors are caused by that continuation, irrespective of what its input might do if it takes control.

Idea: generalize the *symmetric* orthogonality operation, $(\mathbb{A}^+, \mathbb{A}^-)^{\perp\!\!\!\perp} = (\mathbb{A}^{-\perp\!\!\!\perp}, \mathbb{A}^{+\perp\!\!\!\perp})$, to instead be an *asymmetric* saturation operation, $(\mathbb{A}^+, \mathbb{A}^-)^s = (\mathbb{A}^{-s+}, \mathbb{A}^{+s-})$.

Each side only considers only the actions it actively participates in: the terms in $\mathbb{A}^{-s+}$ cannot be the one responsible for causing a problem, and the continuations in $\mathbb{A}^{+s-}$ cannot be blamed for causing a problem.

But it still might be the case that $\mathbb{A}^+ \not\perp\!\!\!\perp \mathbb{A}^{+s-}$ (or $\mathbb{A}^{-s+} \perp\!\!\!\perp \mathbb{A}^-$) due to non-determinism.

Key lemma: at the fixed point, $\mathbb{A} = \mathbb{A}^s$ implies $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp}$!

Definition 4.4.4 (Symmetric Candidate)**.** A *symmetric candidate* is a pre-candidate $\mathbb{A}$ such that $\mathbb{A} = \mathbb{A}^s$ (and thus $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp}$ as well).

Idea: if we can find fixed points of $\_^s$ that have the required canonical constructions/observations, then we can always construct symmetric candidates of arbitrary types.

Lemma 4.4.5. *If* $\mathbb{C}$ *is self-orthogonal (*i.e., $\mathbb{C} \sqsubseteq \mathbb{C}^{\perp\!\!\!\perp}$*,* i.e., $\mathbb{C}^+ \perp\!\!\!\perp \mathbb{C}^-$*), and contains only deterministic terms and continuations, then there is a symmetric candidate* $\mathbb{A}$ *such that* $\mathbb{C} \sqsubseteq \mathbb{A} = \mathbb{A}^s = \mathbb{A}^{\perp\!\!\!\perp}$*.*

Proof (sketch). The key fact is that saturation $\_^s$ has similar logical properties as orthogonality, importantly,

- *Monotonicity:* if $\mathbb{A} \leq \mathbb{B}$ then $\mathbb{A}^s \leq \mathbb{B}^s$.

- *Contrapositive (*a.k.a *antitonicity)* if $\mathbb{A} \sqsubseteq \mathbb{B}$ then $\mathbb{A}^s \sqsupseteq \mathbb{B}^s$.

Because of monotonicity with respect to subtyping, we know that we can find fixed points via the Knaster-Tarski fixed point theorem.

So lets build the fixed point to this subtyping-monotonic operation:

$$\mathrm{Next}(\mathbb{C})(\mathbb{A}) = \mathbb{C} \sqcup \mathbb{A}^s$$

Both $\mathbb{C} \sqcup \_$ and $\_^s$ are monotonic (w.r.t. subtyping), and so is $\mathrm{Next}(\mathbb{C})$. In other words, Knaster-Tarski's fixed point theorem ensures there is an $\mathbb{A}$ such that

$$\mathbb{A} = \mathbb{C} \sqcup \mathbb{A}^s$$

From there, it can then be shown that $\mathbb{A} = \mathbb{A}^s$ (because $\mathbb{C} \sqsubseteq \mathbb{A}^s$, due to deterministic reduction of the inhabitants of $\mathbb{C}$) so that $\mathbb{A} \sqsubseteq \mathbb{A}^{\perp\!\!\!\perp}$, and thus

$$\mathbb{C} \sqsubseteq \mathbb{A}^s = \mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp} \qquad \square$$

This construction is much more powerful (in the way it handles nondeterminism) but much more vague (by not giving a finitely-defined fixed point like before; in fact, it provides a *complete lattice* of possible symmetric candidates to choose from). However, in the special case where reduction is deterministic, the two notions of candidates coincide.

Property 4.4.6. *Assuming $\mapsto$ is deterministic, any pre-candidate $\mathbb{A}$ is a restricted $\perp\!\!\!\perp$-orthogonal candidate ($\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp} = \mathbb{A}^{v\perp\!\!\!\perp}$) if and only if it is a symmetric candidate ($\mathbb{A} = \mathbb{A}^s$).*