Consequence Relations and Natural Deduction

Joshua D. Guttman
Worcester Polytechnic Institute

September 9, 2010

Contents

1 Consequence Relations

A useful notion that cuts across both semantic (model-oriented) and syntactic (derivation-oriented) issues is the notion of a consequence relation. We will use capital Greek letters like Γ, Δ (Gamma and Delta) to refer to finite sets of formulas, and lower case Greek letters like ϕ, ψ (phi and psi) to refer to individual formulas. We will save ink by writing Γ, Δ for the set $\Gamma \cup \Delta$, and Γ, ϕ for the set $\Gamma \cup \{\phi\}$, etc.

By $\phi[t_1/x_1, \ldots t_n/x_n]$, we mean the result of plugging in the terms $t_1, \ldots, t_n$ in place of the variables $x_1, \ldots, x_n$. We assume that all the x_i are different variables, and that all of the plugging in happens at once. So, if there are

x_2s inside the term t_1, they are not substituted with t_2s. $\Gamma[t_1/x_1, \ldots t_n/x_n]$ means the result of doing the substitutions to all the formulas in Γ.

Definition 1 *Suppose that $\preceq$ is a relation between finite sets of formulas and individual formulas, as in $\Gamma \preceq \phi$. Then $\preceq$ is a* consequence relation *iff it satisfies these properties:*

Reflexivity: $\Gamma, \phi \preceq \phi$*;*

Transitivity: $\Gamma \preceq \phi$ *and* $\Gamma, \phi \preceq \psi$ *imply* $\Gamma \preceq \psi$*;*

Weakening: $\Gamma \preceq \phi$ *implies* $\Gamma, \Delta \preceq \phi$*; and*

Substitution: $\Gamma \preceq \phi$ *implies* $\Gamma[t_1/x_1, \ldots t_n/x_n] \preceq \phi[t_1/x_1, \ldots t_n/x_n]$.

For now, we will focus on formulas with no variables, so **Substitution** will be irrelevant. We will ignore it until later. The **Reflexivity** and **Transitivity** rules ensure that a consequence relation is a partial order, when restricted to sets containing just one assumption. The **Weakening** rule "lifts" this partial order to sets with more members.

We refer to an instance of a relation $\Gamma \preceq \phi$ or any $\Gamma \; R \; \phi$ as a *judgment*.

Both semantic notions such as *entailment* and syntactic notions such as *derivability* give us examples of consequence relations. Suppose we have a notion of *model* such as $\mathbb{M} \models \phi$ as defined in the Dougherty lecture notes, Def. 2.2.2.[1] Then we have a corresponding notion of *(semantic) entailment* defined:

Definition 2 Γ entails ϕ*, written* $\Gamma \Vdash \phi$*, holds iff, for all models* $\mathbb{M}$*:*

If *for each* $\psi \in \Gamma$, $\mathbb{M} \models \psi$,

then $\mathbb{M} \models \phi$.

That is, $\Gamma \Vdash \phi$ means that every model that makes all of the formulas in Γ true makes ϕ true too.

Lemma 3 *Entailment is a consequence relation, i.e.* $\Vdash$ *satisfies reflexivity, transitivity, and weakening in Def. 1:*

1. $\Gamma, \phi \Vdash \phi$*;*
2. $\Gamma \Vdash \phi$ *and* $\Gamma, \phi \Vdash \psi$ *imply* $\Gamma \Vdash \psi$*; and*

[1] Available at URL `http://web.cs.wpi.edu/~guttman/cs521_website/Dougherty_lecture_notes.pdf`.

$$\frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}$$

Figure 1: ND Introduction and Elimination Rules for $\wedge$

$$\frac{\Gamma\phi \vdash \psi}{\Gamma \vdash \phi \to \psi} \qquad \frac{\Gamma \vdash \phi \to \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

Figure 2: ND Introduction and Elimination Rules for $\to$

3. $\Gamma \Vdash \phi$ *implies* $\Gamma, \Delta \Vdash \phi$.

We turn next to showing that a particular set of rules for constructing proofs is also a consequence relation.

2 A Derivation System for "Natural Deduction"

We consider the rules suggested by Gerhart Gentzen as a "natural" form of deduction [3]. Gentzen considered these rules natural because they seemed to match directly the meaning of each logical operator.

Each logical operator has one or a couple of rules that allow you to prove formulas containing it as the outermost operator. These are called *introduction* rules. Each operator also has one or a couple of rules that allow you to prove other formulas by extracting the logical content in a formula containing it as outermost operator. They are called *elimination* rules. The introduction rules push formulas up in the partial ordering, while the elimination rules hold them down. Between them, the introduction and elimination rules *fix the meaning* of the logical operators purely in terms of their *deductive power*.

All of this extends to much richer logics, as we will see.

The rules are spread out through Figs. 1–4.

Definition 4 *A* natural deduction derivation *is a tree, conventionally written with the conclusion, the root, at the bottom, such that each judgment is*

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$$

$$\frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \chi \qquad \Gamma, \psi \vdash \chi}{\Gamma \vdash \chi}$$

Figure 3: ND Introduction and Elimination Rules for $\vee$

$$\frac{}{\Gamma, \phi \vdash \phi} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}$$

Figure 4: ND Axioms and Rule for $\bot$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \wedge q \vdash p \wedge q}}{p \wedge q \vdash p}}{p \wedge q \vdash p \vee q}}{\vdash (p \wedge q) \rightarrow (p \vee q)}$$

Figure 5: An Example Derivation

the conclusion of a rule.

A derivation is a natural deduction derivation in intuitionist propositional logic *if each rule is one of those shown in Figs. 1–4.*

An example derivation is shown in Fig. 5. It proves $\vdash (p \wedge q) \rightarrow (p \vee q)$. There are two questions we'd immediately like answers to. First, do the derivable judgments form a consequence relation? That is, if $\Gamma \preceq \phi$ means that there is a derivation of $\Gamma \vdash \phi$ using our rules, then is $\preceq$ a consequence relation?

Second, how do derivable judgments relate to entailment? If $\Gamma \vdash \phi$ is derivable, then is $\Gamma \Vdash \phi$ true? If $\Gamma \Vdash \phi$ then is $\Gamma \vdash \phi$ derivable?

We can answer the first question affirmatively.

Lemma 5 *The set of derivable judgments* $\Gamma \vdash \phi$ *form a consequence relation.*

Proof: 1. **Reflexivity** holds because $\dfrac{}{\Gamma, \phi \vdash \phi}$ is always a derivation.

2. **Transitivity** holds by Fig. 6.

3. **Weakening** holds by *induction on derivations*:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ d_1\\ \vdots\\ \Gamma, \phi \vdash \psi\end{array}}{\Gamma \vdash \phi \rightarrow \psi} \qquad \begin{array}{c}\vdots\\ d_2\\ \vdots\\ \Gamma \vdash \phi\end{array}}{\Gamma \vdash \psi}$$

Figure 6: Composing Derivations for Transitivity

Base Case Suppose that there is a derivation of $\Gamma \vdash \phi$ consisting only of an application of the Axiom rule. That is, $\phi \in \Gamma$. Thus, $\phi \in \Gamma, \Delta$, so $\dfrac{}{\Gamma, \Delta \vdash \phi}$ is an application of the Axiom rule.

Induction Step Suppose that we are given a derivation d where the last step is an application of one of the rules from Figs. 1–4, and the previous steps generate one or more subderivations d_i, each with conclusion $\Gamma_i \vdash \psi_i$.

Induction hypothesis. Assume that for each of the subderivations d_i, there is a weakened subderivation $W(d_i)$ such that $W(d_i)$ has conclusion $\Gamma_i, \Delta \vdash \psi_i$.

Construct the desired derivation of $\Gamma, \Delta \vdash \phi$ by combining the weakened subderivations $W(d_i)$ using the *same* rule of inference.

□

One part of the second question is easy to answer.

Lemma 6 $\vdash \subseteq \Vdash$.
That is, if $\Gamma \vdash \phi$ *is derivable, then* $\Gamma \Vdash \phi$.

Proof: By induction on derivations. □

On the other hand, $\vdash \subsetneq \Vdash$. There are entailment relations that cannot be derived using these rules.

Challenge. Find a Γ, ϕ such that $\Gamma \Vdash \phi$ but $\Gamma \vdash \phi$ is not derivable using our rules. How would one prove it not derivable?

Question. If these rules do not characterize the semantic entailment relation generated from the classical $\models$, what do they characterize?

3 Derivations with Explicit Proof Objects

In this section we annotate our derivations with a representation of the proofs the derivations construct. This will allow us to manipulate the forms of proofs, and also to treat proof and computation in an overlapping way. The remaining presentation draws heavily on [1, 2, 4].

Definition 7 *By a* context Γ*, we mean a set of pairs consisting of a variable and a formula, such that no variable appears more than once. We write members of* Γ *in the form* $v : \phi$*, so a context takes the form:*

$$v_1 : \phi_1, \ldots v_i : \phi_i.$$

$$\frac{\Gamma \vdash s\colon \phi \qquad \Gamma \vdash t\colon \psi}{\Gamma \vdash \langle s, t\rangle\colon \phi \wedge \psi}$$

$$\frac{\Gamma \vdash s\colon \phi \wedge \psi}{\Gamma \vdash \mathsf{fst}(s)\colon \phi} \qquad\qquad \frac{\Gamma \vdash s\colon \phi \wedge \psi}{\Gamma \vdash \mathsf{scd}(s)\colon \psi}$$

Figure 7: Rules for Conjunction, with Explicit Proof Objects

$$\frac{\Gamma \vdash s\colon \phi}{\Gamma \vdash \langle \mathsf{lft}, s\rangle\colon \phi \vee \psi} \qquad\qquad \frac{\Gamma \vdash s\colon \psi}{\Gamma \vdash \langle \mathsf{rgt}, s\rangle\colon \phi \vee \psi}$$

$$\frac{\Gamma \vdash s\colon \phi \vee \psi \qquad \Gamma, x\colon \phi \vdash t\colon \chi \qquad \Gamma, y\colon \psi \vdash r\colon \chi}{\Gamma \vdash \mathsf{cases}(s, \lambda x\,.\,t, \lambda y\,.\,r)\colon \chi}$$

Figure 8: Rules for Disjunction, with Explicit Proof Objects

The empty context is permitted, i.e. when $i = 0$ there are no $v\colon \phi$ pairs.

Thus, a context is a finite partial function from variables to formulas. Henceforth, we will use Γ for contexts rather than simply sets of formulas.

We annotate each of the rules of Figs. 1–4 with explicit proof objects in such a way that the introduction rules and elimination rules cancel out. If a derivation ends with an introduction rule followed by an elimination rule, then we should be able to extract the proof object for the subderivation before the two redundant steps. For instance, the conjunction introduction rule constructs a pair object, from which the two elimination rules can extract the components, i.e. recover the embedded proofs of the individual conjuncts. Likewise, the disjunction introduction rules tag their subderivation $s\colon \phi$ or $s\colon \psi$, so that the *cases* construct can insert it into the appropriate branch proving χ. The rule for falsehood has a less symmetrical role: We call it the "empty promise" rule, on the grounds that it should never be possible to apply it to a closed (variable-free) term s. We give the syntax of explicit proof terms in Fig. 11. Compound terms of certain forms may be *reduced* as indicated in the *reduction rules* shown in Fig. 12. We regard Fig. 12 as giving a kind of operational semantics for an extremely simple programming language. Unfortunately, the most important reduction rule, the "beta rule" β, is not as simple as it looks.

$$\frac{\Gamma, x\colon \phi \vdash s\colon \psi}{\Gamma \vdash \lambda x\,.\,s\colon \phi \to \psi} \qquad\qquad \frac{\Gamma \vdash s\colon \phi \to \psi \qquad \Gamma \vdash t\colon \phi}{\Gamma \vdash (st)\colon \psi}$$

Figure 9: Rules for Implication, with Explicit Proof Objects

$$\frac{}{\Gamma, x\colon \phi \ \vdash \ x\colon \phi} \qquad\qquad \frac{\Gamma \ \vdash \ s\colon \bot}{\Gamma \ \vdash \ \mathsf{emp}(s)\colon \phi}$$

Figure 10: Rules for Axioms and Falsehood, with Explicit Proof Objects

$$\begin{array}{rcllllll}
s,t,r & ::= & v & | & & & & \\
 & & \langle s,t\rangle & | & \mathsf{fst}(s) & | & \mathsf{scd}(s) & | \\
 & & (\lambda v\ .\ s) & | & s\ t & | & & \\
 & & \langle \mathsf{lft}, s\rangle & | & \langle \mathsf{rgt}, s\rangle & | & \mathsf{cases}(s,t,r) & \\
v & ::= & x & | & y & | & v' & \ldots
\end{array}$$

Figure 11: Syntax of Proof Terms

$$\begin{array}{lcll}
\mathsf{fst}(\langle s, s'\rangle) & \longrightarrow_r & s & \\
\mathsf{scd}(\langle s, s'\rangle) & \longrightarrow_r & s' & \\
\mathsf{cases}(\langle \mathsf{lft}, s\rangle, t, r) & \longrightarrow_r & t\ s & \\
\mathsf{cases}(\langle \mathsf{rgt}, s\rangle, t, r) & \longrightarrow_r & r\ s & \\
(\lambda v\ .\ s)\ t & \longrightarrow_r & s[t/v] & (\beta)
\end{array}$$

Figure 12: Reduction Rules for Proof Terms

4 How to Beta-Reduce

The rule for β-reduction is complicated by the need to rename bound variables when evaluating $s[t/v]$, i.e. plugging a term t into a term s in place of v. The problems arise when t contains a free occurrence of a variable x, but within s, v has an occurrence in the body of some λ-binder $\lambda x\ .\ r$. In this situation, x's free occurrences in t would be "captured" by this λ-binder.

As an example, consider

$$(\lambda v\ .\ (\lambda x\ .\ (x\ v)))\ (x).$$

If we reduce it by plugging in x in place of v, we should *not* obtain $(\lambda x\ .\ (x\ x))$. Nothing here asks us to accept a function x and apply it to itself.

Instead, we would like to change the name of the bound variable x in $\lambda x\ .\ (x\ v)$ before plugging in the meaningful, externally chosen value for x in place of v. Thus, we first *α-convert* $\lambda x\ .\ (x\ v)$ by renaming its x to some new variable z. Now there is no danger and we plug in the free x in place of v without risk of capturing x, obtaining $\lambda z\ .\ (z\ x)$. After all, the renaming can do no harm: the bound variable x or z will really only get its value later when we apply this term to some argument, which will furnish its value.

For this reason, we first explain how to α-convert.

4.1 How to α-Convert

Definition 8 (Free and bound variables) *We define the* free variables *and the* bound variables *of a term recursively:*

$$\begin{aligned} \mathsf{fv}(v) &= \{v\} & \qquad \mathsf{bv}(v) &= \emptyset \\ \mathsf{fv}(s\ t) &= \mathsf{fv}(s) \cup \mathsf{fv}(t) & \qquad \mathsf{bv}(s\ t) &= \mathsf{bv}(s) \cup \mathsf{bv}(t) \\ \mathsf{fv}(\lambda v\ .\ s) &= \mathsf{fv}(s) \setminus \{v\} & \qquad \mathsf{bv}(\lambda v\ .\ s) &= \mathsf{bv}(s) \cup \{v\} \end{aligned}$$

The variables *of s are those of both kinds:* $\mathsf{vars}(s) = \mathsf{fv}(s) \cup \mathsf{bv}(s)$.

Be careful: there are terms in which a variable v occurs both free and bound. For instance, letting s be $(\lambda v\ .\ x\ v)(v)$, $v \in \mathsf{fv}(s) \cup \mathsf{bv}(s)$.

We call an object $\sigma = v \mapsto v'$ a (one-variable) *replacement*. Suppose that $\lambda v\ .\ s$ is a lambda expression whose topmost bound variable v is the same as the argument of σ. Then the *result* of σ on $\lambda v\ .\ s$ is $\lambda v'\ .\ s'$, where s' arises by replacing every free occurrence of v by v' throughout s.

For instance, the result of $\sigma = v \mapsto v'$ on the term

$$\lambda v\ .\ ((\lambda v\ .\ v\ x)\ v)$$

is

$$\lambda v' \,.\, ((\lambda v \,.\, v\ x)\ v').$$

The body of the first term has only one *free* occurrence of v, which has been replaced by v'.

If $v' \notin \mathsf{fv}(\lambda v \,.\, s)$, then the result of $\sigma = v \mapsto v'$ on $\lambda v \,.\, s$ will represent the same function as $\lambda v \,.\, s$. We cannot prove this currently, since we haven't formalized "the function that s represents." However, the claim is a reasonable constraint on any formalization we would give: The name of the bound variable v doesn't matter, since it is gone as soon as we plug in an argument for v. Moreover, we will plug in the argument in the same locations in the result $\lambda v' \,.\, s'$ as in $\lambda v \,.\, s$.

Two terms are α-equivalent if one can be obtained from the other by applying this operation to their parts.

Definition 9 *α-equivalence, written $\equiv_\alpha$, is the smallest relation such that:*

1. *$s \equiv_\alpha s$;*
2. *if $s \equiv_\alpha s'$ and $t \equiv_\alpha t'$, then $(s\ t) \equiv_\alpha (s'\ t')$;*
3. *if $s \equiv_\alpha s'$, then $(\lambda v \,.\, s) \equiv_\alpha (\lambda v \,.\, s')$;*
4. *if $v' \notin \mathsf{fv}(\lambda v \,.\, s)$, and $\lambda v' \,.\, s'$ is the result of $v \mapsto v'$ on $\lambda v \,.\, s$, then*

$$(\lambda v \,.\, s) \equiv_\alpha (\lambda v' \,.\, s').$$

Lemma 10 *Let s, t be terms.*

1. *$\mathsf{bv}(s)$ and $\mathsf{fv}(s)$ are finite.*
2. *If t is a subterm of s, then $\mathsf{fv}(t) \subseteq \mathsf{vars}(s)$.*
3. *Let $F \subseteq \mathcal{V}$ be a finite set of variables. There is an s' such that $s' \equiv_\alpha s$ and $\mathsf{bv}(s') \cap F = \emptyset$.*
4. *There is a s' such that $s' \equiv_\alpha s$ and $\mathsf{bv}(s') \cap \mathsf{fv}(t) = \emptyset$.*
5. *There is an s' such that $s' \equiv_\alpha s$ and: (i) any two λ-expressions within s' bind different variables and (ii) $\mathsf{bv}(s') \cap \mathsf{fv}(s') = \emptyset$.*

A term s' satisfying the conditions (i,ii) in Clause 5 is said to be in "standard form." People frequently assume, in the middle of a proof, that any term they're working with is in standard form. Clause 5 justifies this, because even if the term *isn't* in standard form, some α-equivalent term is.

Proof: 1. By induction on the structure of terms. *Base case.* If s is a variable v, then $\mathsf{bv}(v) = \emptyset$ and $\mathsf{fv}(v) = \{v\}$, both of which are finite.

Induction step. Suppose (i.h.) that $\mathsf{bv}(s)$ and $\mathsf{fv}(s)$ are finite, and so are $\mathsf{bv}(t)$ and $\mathsf{fv}(t)$.

Then so are $\mathsf{fv}(s\ t)$ and $\mathsf{bv}(s\ t)$, since the union of two finite sets is finite. $\mathsf{fv}(\lambda v\ .\ s)$ is a subset of $\mathsf{fv}(s)$, so finite. $\mathsf{bv}(\lambda v\ .\ s)$ is the union of the finite set $\{v\}$ with a finite set, so also finite.

2. For every path p to an occurrence of a free variable v in t, consider $q^\frown p$, where q is a path in s to an occurrence of t. Then $q^\frown p$ is a path to v. It is a path to a bound occurrence if q traverses some λv, and otherwise it is a path to a free occurrence.

3. Let $\langle v_i \rangle_{i \in \mathbb{N}}$ be an infinite sequence that enumerates all the variables $v \in \mathcal{V}$. For any term t, let $\mathsf{fresh}(t) = \max\{i\colon v_i \in F \cup \mathsf{vars}(t)\}$: so $\mathsf{fresh}(t)$ is well-defined because F and $\mathsf{vars}(t)$ are finite. Now argue by induction on the structure of terms.

Base case. If s is a variable v, then $\mathsf{bv}(v) = \emptyset$, so s' can be v.

Induction step. Suppose given t_0 and t_1. Assume (i.h.) that there are t_0' and t_1' such that $t_i' \equiv_\alpha t_i$ and $\mathsf{bv}(t_i') \cap F = \emptyset$.

If $s = (t_0\ t_1)$, then the desired s' is $s' = (t_0'\ t_1')$.

If $s = \lambda v\ .\ t_0$, and $v \notin F$, then the desired s' is $s' = \lambda v\ .\ t_0'$.

Otherwise, let $v' = \mathsf{fresh}(t_0')$, and let t' result from s' under $v \mapsto v'$.

4. Apply Clause 3 to $F = \mathsf{fv}(t)$.

5. Use $v' = \mathsf{fresh}(t_0')$ as in Clause 3, starting with $F = \mathsf{vars}(s)$. □

4.2 How to β-Reduce

Terminology varies slightly among the different notions introduced in the next definition.

Definition 11 *Let $s = \lambda v\ .\ s_0$, and let t be a term. Let $s' = \lambda v'\ .\ s_0'$ be chosen in some canonical way such that $s' \equiv_\alpha s$ and $\mathsf{bv}(s') \cap \mathsf{fv}(t) = \emptyset$. Then $s_0'[t/v']$ is defined to be the result of replacing every free occurrence of v' with the term t.*

*We say $(s\ t)$ β-*reduces to *r iff $r \equiv_\alpha s_0'[t/v']$, and we write $(s\ t) \to_b r$.*

We write $s \to_{\beta 1} t$ for the least relation such that:

1. *$s \to_b t$ implies $s \to_{\beta 1} t$;*
2. *$s_0 \to_{\beta 1} s_1$ implies:*
   (a) *$(s_0\ t) \to_{\beta 1} (s_1\ t)$;*

(b) $(t\ s_0) \to_{\beta 1} (t\ s_1)$*; and*

(c) $\lambda v\ .\ s_0 \to_{\beta 1} \lambda v\ .\ s_1$*.*

$\to_\beta$ *is the least relation which is transitive and closed under* $\equiv_\alpha$ *and includes* $\to_{\beta 1}$*. Thus,* $s \to_\beta t$ *holds whenever* $s \equiv_\alpha t$*, or* $s \to_{\beta 1} t$*, or there is an* r *such that* $s \to_\beta r$ *and* $r \to_\beta t$*.*

The relation $\to_b$ gives the correct meaning for the last line of Fig. 12.

5 Reduction and Typing for Proof Terms

Throughout Section 4, we ignored the other proof terms, generated using pairing $\langle \cdot, \cdot \rangle$ and the functions $\mathsf{fst}, \mathsf{scd}, \mathsf{cases}$. It can be easily extended to the larger language. In particular, the definition of $\equiv_\alpha$ extends mechanically through the remaining proof terms, with the same properties. The reduction relation takes the form:

Definition 12 *The one-step reduction relation* $\longrightarrow_1$ *is the smallest relation such that:*

1. *If* $s \longrightarrow_r t$ *as defined in Fig. 12, then* $s \longrightarrow_1 t$*;*

2. $s_0 \longrightarrow_1 s_1$ *implies:*

(i)	$(s_0\ t) \longrightarrow_1 (s_1\ t)$*;*	*(ii)*	$(t\ s_0) \longrightarrow_1 (t\ s_1)$*;*
(iii)	$\lambda v\ .\ s_0 \longrightarrow_1 \lambda v\ .\ s_1$*;*	*(iv)*	$\mathsf{fst}(s_0) \longrightarrow_1 \mathsf{fst}(s_1)$*;*
(v)	$\mathsf{scd}(s_0) \longrightarrow_1 \mathsf{scd}(s_1)$*;*	*(vi)*	$\langle s_0, t\rangle \longrightarrow_1 \langle s_1, t\rangle$*;*
(vii)	$\langle t, s_0\rangle \longrightarrow_1 \langle t, s_1\rangle$	*(viii)*	$\mathsf{cases}(s_0, t, r) \longrightarrow_1 \mathsf{cases}(s_1, t, r)$*;*
(ix)	$\mathsf{cases}(t, s_0, r) \longrightarrow_1 \mathsf{cases}(t, s_1, r)$*;*	*(x)*	$\mathsf{cases}(t, r, s_0) \longrightarrow_1 \mathsf{cases}(t, r, s_1)$*.*

The reduction relation $\longrightarrow$ *is the least relation which is transitive and closed under* $\equiv_\alpha$ *and includes* $\longrightarrow_1$*.*

Thus, $s \longrightarrow t$ *holds whenever* $s \equiv_\alpha t$*, or* $s \longrightarrow_1 t$*, or there is an* r *such that* $s \longrightarrow r$ *and* $r \longrightarrow t$*.*

We have also ignored the typing discipline, and the whole discussion of Section 4 is equally applicable if it is altered or discarded. However, the type structure ensures that interesting additional properties hold true of the reduction relation. We here follow Barendregt [1, Section 3.1].

We will write henceforth $\Gamma \vdash s\colon \phi$ to mean that there is a derivation using the explicit proof rules (Figs. 7–10) where $\Gamma \vdash s\colon \phi$ is the last line of the derivation (the root of the tree).

Lemma 13 (Context Lemma) *Suppose that* Γ, Γ' *are contexts.*

1. *If* $\Gamma \subseteq \Gamma'$*, then* $\Gamma \vdash s\colon \phi$ *implies* $\Gamma' \vdash s\colon \phi$*.*
2. *If* $\Gamma \vdash s\colon \phi$*, then* $\mathsf{fv}(s) \subseteq \mathsf{dom}(\Gamma)$*.*
3. $\Gamma \vdash s\colon \phi$*, and* $\Gamma' = \Gamma \restriction \mathsf{fv}(s)$*, then* $\Gamma' \vdash s\colon \phi$

Proof: By induction on derivations. Clause 1 is the analog to Weakening (Lemma 5) in the explicit proof formalism. □

Clause 2 of this lemma has a significant consequence: it says that the closed proof objects—the ones with no free variables—are very important. They are the only ones that prove a conclusion with no premises, i.e. with $\Gamma = \emptyset$. The other clauses say that premises in Γ with variables not appearing free in the right hand side do no good (Clause 3) and do no harm (Clause 1).

We next summarize what must be true when $\Gamma \vdash s\colon \phi$, as a function of the syntactic form of s as a proof term. Naturally, for a given proof term, we can use this lemma repeatedly on its subexpressions to unfold the derivation from the bottom up.

Lemma 14 (Generation) *Suppose that* $\Gamma \vdash s_0\colon \phi$*. If* s_0 *is of the form:*

v**, then** $x\colon \phi \in \Gamma$*;*

$\langle s, t\rangle$**, then** $\phi = \phi_1 \wedge \phi_2$ *and* $\Gamma \vdash s\colon \phi_1$ *and* $\Gamma \vdash t\colon \phi_2$*;*

$\mathsf{fst}(s)$**, then** *for some* ψ*,* $\Gamma \vdash s\colon \phi \wedge \psi$*;*

$\mathsf{scd}(s)$**, then** *for some* ψ*,* $\Gamma \vdash s\colon \psi \wedge \phi$*;*

$\lambda v\,.\,s$**, then** $\phi = \phi_1 \to \phi_2$ *and* $\Gamma, v\colon \phi_1 \vdash \phi_2$*;*

$(s\ t)$**, then** *for some* ψ*,* $\Gamma \vdash s\colon \psi \to \phi$ *and* $\Gamma \vdash t\colon \psi$*;*

$\langle \mathsf{lft}, s\rangle$**, then** $\phi = \phi_1 \vee \phi_2$ *and* $\Gamma \vdash s\colon \phi_1$*;*

$\langle \mathsf{rgt}, s\rangle$**, then** $\phi = \phi_1 \vee \phi_2$ *and* $\Gamma \vdash s\colon \phi_2$*;*

$\mathsf{cases}(s, t, r)$**, then** *for some disjunction* $\psi_1 \vee \psi_2$*,* $\Gamma \vdash s\colon \psi_1 \vee \psi_2$*, and* $\Gamma \vdash t\colon \psi_1 \to \phi$*, and* $\Gamma \vdash r\colon \psi_2 \to \phi$*.*

Proof: The proof is essentially by pattern matching the form of each rule's conclusion. The only wrinkle is in the last clause, where we have an implication $\psi_1 \to \phi$ because we have lambda-bound the variable in each subsidiary derivation in the or-elimination rule. □

Lemma 15 (Subterms typable) *Suppose that* $\Gamma \vdash s\colon \phi$*, and* t *is a subterm of* s*. Then for some* Γ' *and some* ψ*,* $\Gamma' \vdash t\colon \psi$*.*

Indeed, with our current rules, $\Gamma' \supseteq \Gamma$ *and* ψ *is a subformula of* ϕ*.*

Proof: By induction on the derivations. □

Note, however, that there are some terms that are not typable with any Γ and ϕ. An example is $(x\ x)$. So no term with any part of this form is typable in this system.

The following lemma shows that proof terms are generic. Derivations do not depend on the fact that any formula is atomic. Thus, the same explicit proof term can have many typings (and prove many theorems) if an atomic formula part is replaced by a compound.

Lemma 16 (Generic proof objects) *Suppose for some atomic formula* p*,* $\Gamma, x\colon p \vdash s\colon \phi$*. Then for any* ψ*,* $\Gamma, x\colon \psi \vdash s\colon \phi'$*, where* ϕ' *results from* ϕ *by replacing every occurrence of* p *with the formula* ψ*.*

Lemma 17 *Suppose that* $\Gamma, x\colon \phi \vdash s\colon \psi$ *and* $\Gamma \vdash t\colon \phi$*. Then* $\Gamma \vdash s[t/x]\colon \psi$*.*

This leads directly to a crucial theorem. It relates the reduction relation of Def. 12 to derivations (or type judgments, which is the same thing).

Theorem 18 (Subject Reduction) *Suppose that* $s \longrightarrow t$ *and* $\Gamma \vdash s\colon \phi$*. Then* $\Gamma \vdash t\colon \phi$ *also.*

This is also known as a *type preservation theorem*, since it says that the type ϕ is preserved no matter how we reduce s. The phrase "subject reduction" comes from the idea that in the judgment $s\colon \phi$, s is the "subject" and ϕ is the "predicate." It says that when the subject reduces, the predicate still applies. Theorems of this form are ubiquitous in programming language semantics.

6 Normalization

A term t is *a normal form* with respect to a reduction relation such as our $\longrightarrow$ if it has no parts that can be reduced, i.e. there is no $s \neq t$ such that $t \longrightarrow s$. If $s \longrightarrow t$ and t is a normal form, then we think of t as a value that the program s computes. If in addition t is closed—it has no free variables—this is especially appropriate. Particularly in our context where it means that t proves a theorem ϕ with no premises: $\emptyset \vdash t\colon \phi$. In this section, we will prove that every well-typed term has a normal form.

Theorem 19 (Normal Form) *If* $\Gamma \vdash s\colon \phi$*, then there is a normal form* t *such that* $s \longrightarrow t$*. By Thm. 18,* $\Gamma \vdash t\colon \phi$*.*

If a derivation system satisfies the analogue of Thm. 19, then a great deal of information about what it proves follows by considering the forms of its closed normal forms. This is a crucial technique for proving consistency theorems, and also for proving that one derivation system is a conservative extension of another.

[[To be continued...]]

References

[1] Henk Barendregt. Lambda calculi with types. *Handbook of logic in computer science*, 2:117–309, 1992.

[2] Henk Barendregt and Silvia Ghilezan. Lambda terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10(01):121–134, 2000.

[3] Gerhard Gentzen. Investigations into logical deduction. In Manfred Szabo, editor, *Complete Works of Gerhard Gentzen*. North Holland, 1969. Originally published in *Mathematische Zeitschrift*, 1934–1935.

[4] Ralph Loader. Notes on simply typed lambda calculus. Technical Report ECS-LFCS-98-381, University of Edinburgh, 1998. At URL `http://www.lfcs.inf.ed.ac.uk/reports/98/ECS-LFCS-98-381/`.