The substitution principles on natural deductions can be expressed on proof terms. This is because the translations from natural deductions to proof terms and *vice versa* are *compositional*: uses of a hypothesis labelled w in natural deduction corresponds to an occurrence of a variable w in the proof term.

Lemma 4.2 (Substitution on Proof Terms)

1. *If* $\Gamma; (\Delta, w{:}A) \vdash N{:}C$ *and* $\Gamma; \Delta' \vdash M : A$, *then* $\Gamma; (\Delta \times \Delta') \vdash [M/w]N : C$.
2. *If* $(\Gamma, u{:}A); \Delta \vdash N{:}C$ *and* $\Gamma; \cdot \vdash M : A$, *then* $\Gamma; \Delta \vdash [M/u]N : C$.

Proof: By induction on the structure of the first given derivation, using the property of exchange. □

We also have the property of weakening for unrestricted hypotheses. The substitution properties are the critical ingredient for the important *subject reduction* properties, which guarantee that the result of β-reducing a well-typed term will again be well-typed. The expansion rules also preserve types when invoked properly.

Theorem 4.3 (Subject Reduction and Expansion)

1. *If* $\Gamma; \Delta \vdash M : A$ *and* $M \longrightarrow_\beta M'$ *then* $\Gamma; \Delta \vdash M' : A$.
2. *If* $\Gamma; \Delta \vdash M : A$ *and* $M : A \longrightarrow_\eta M'$ *then* $\Gamma; \Delta \vdash M' : A$.

Proof: For subject reduction we examine each possible reduction rule, applying inversion to obtain the shape of the typing derivation. From this we either directly construct the typing derivation of M' or we appeal to the substitution lemma.

For subject expansion we directly construct the typing derivation for M' from the typing derivation of M. □

Note that the opposite of subject reduction does not hold: there are well-typed terms M' such that $M \longrightarrow_\beta M'$ and M is not well-typed (see Exercise 4.4).

4.2 Example: A Small Imperative Language

[*to be filled in*]

4.3 Term Assignment for the Sequent Calculus

Writing an efficient theorem prover is an arduous and error-prone task, since one must carefully optimize at low and high levels of abstraction. This means that it is difficult to trust the correctness of a theorem prover. In order to alleviate this problem, we can follow two strategies. The first goes back to the design

ML [GMW79] where we use the strong typing and the data abstraction mechanisms of the implementation language to reduce the correctness of a larger implementation to the correctness of a small core. Unfortunately, having to always go through primitive rules of inference during search is a severe practical restriction and prohibits many efficient implementation techniques (see, for example, a discussion in [**?**]). Another is to require the prover to be able to generate *proof terms*. If the proof is written in a simple and concise language, we can write an external (and hopefully much simpler) checker, which we may trust much more readily than a complicated prover manipulating constraints, unification, using indexing schemes for fast retrieval of logical assumptions, etc. Unless we can find a way to include admissible and derived rules of inference, we still have to go through primitive inference rules, but we now have an external manifestation of the deduction.

For linear logic, the proof term calculus developed in Section 4.1 is an ideal candidate. Its definition is relatively simple, directly translating the rules of natural deduction. Compare this to the complexity of unification with parameters and manipulating occurrence constraints. In the next section we will also see that type-checking proof terms is certainly decidable and actually not a difficult task.

What remains is to bridge the gap between the rules of the sequent calculus and proof terms for natural deduction. Actually, we have already established the connection between sequent calculus and natural deduction in both directions. Of interest here is soundness, since the constructive soundness proof gives an explicit method for translating a sequent derivation into a natural deduction (see Theorem 2.9). We now need to make this construction explicit. This can be done by defining a judgment which relates a sequent derivation to a proof term, or perhaps to a natural deduction which includes a proof term. Such higher-level judgments which relate derivations quickly become unmanageable, so we write out one judgment which may be thought of as a sequent derivation annotated by a proof term, written as $\Gamma; \Delta \Longrightarrow M : A$. We are writing it in such a way that, if $\Gamma; \Delta \Longrightarrow A$ then there is an annotation $\Gamma; \Delta \Longrightarrow M : A$ and $\Gamma; \Delta \vdash M : A$.

Since variable occurrences in proof terms are critical, we now make the hypothesis labels explicit. However, we will still allow implicit exchange, so that $\Delta, w{:}A$ matches any hypotheses of the form $\Delta_1, w{:}A, \Delta_2$.

Hypotheses. The use of a hypothesis is just translated into the corresponding variable. The dereliction rule requires a substitution.

$$\frac{}{\Gamma; w{:}A \Longrightarrow w : A}\,\mathrm{I} \qquad \frac{(\Gamma, u{:}A); (\Delta, w{:}A) \Longrightarrow M : C}{(\Gamma, u{:}A); \Delta \Longrightarrow [u/w]M : C}\,\mathrm{DL}$$

Why is the substitution in the dereliction rule valid? In the correctness proof of the proof term assignment we need to show that $(\Gamma, u{:}A); \Delta \vdash [u/w]M : C$, given that $(\Gamma, u{:}A); (\Delta, w{:}A) \vdash M : C$. But this follows from the substitution property for proof terms (Lemma 4.2), since $(\Gamma, u{:}A); \cdot \vdash u : A$. Many other cases follow a similar pattern.

Multiplicative Connectives. In general, the right rules of the sequent calculus match the introduction rules of natural deduction. Therefore, the proof term assignment for the left rules is quite straightforward. On the other side, the left rules decompose a proposition in bottom-up search, while the elimination rules work top-down. We therefore substitute a small piece of a derivation which applies the corresponding elimination for the hypothesis in the premiss.

$$\frac{\Gamma; \Delta, w{:}A \Longrightarrow M : B}{\Gamma; \Delta \Longrightarrow \hat{\lambda} w{:}A.\ M : A \multimap B} \multimap\mathrm{R}$$

$$\frac{\Gamma; \Delta_1 \Longrightarrow M : A \qquad \Gamma; \Delta_2, w_2{:}B \Longrightarrow N : C}{\Gamma; \Delta_2 \times \Delta_1, w{:}A \multimap B \Longrightarrow [(w\,\hat{}\,M)/w_2]N : C} \multimap\mathrm{L}$$

$$\frac{\Gamma; \Delta_1 \Longrightarrow M_1 : A \qquad \Gamma; \Delta_2 \Longrightarrow M_2 : B}{\Gamma; \Delta_1 \times \Delta_2 \Longrightarrow M_1 \otimes M_2 : A \otimes B} \otimes\mathrm{R}$$

$$\frac{\Gamma; \Delta, w_1{:}A, w_2{:}B \Longrightarrow N : C}{\Gamma; \Delta, w{:}A \otimes B \Longrightarrow \mathbf{let}\ w_1 \otimes w_2 = w\ \mathbf{in}\ N : C} \otimes\mathrm{L}$$

$$\frac{}{\Gamma; \cdot \Longrightarrow \star : \mathbf{1}} \mathbf{1}\mathrm{R} \qquad \frac{\Gamma; \Delta \Longrightarrow N : C}{\Gamma; \Delta, w{:}\mathbf{1} \Longrightarrow \mathbf{let}\ \star = w\ \mathbf{in}\ N : C} \mathbf{1}\mathrm{L}$$

Additive Connectives. The additive connectives do not introduce any complications.

$$\frac{\Gamma; \Delta \Longrightarrow M : A \qquad \Gamma; \Delta \Longrightarrow N : B}{\Gamma; \Delta \Longrightarrow \langle M, N \rangle A \& B} \&\mathrm{R}$$

$$\frac{\Gamma; \Delta, w_1{:}A \Longrightarrow N : C}{\Gamma; \Delta, w{:}A \& B \Longrightarrow [(\mathrm{fst}\, w)/w_1]N : C} \&\mathrm{L}_1 \qquad \frac{\Gamma; \Delta, w_2{:}B \Longrightarrow N : C}{\Gamma; \Delta, w{:}A \& B \Longrightarrow [(\mathrm{snd}\, w)/w_1]N : C} \&\mathrm{L}_2$$

$$\frac{}{\Gamma; \Delta \Longrightarrow \langle\,\rangle \top} \top\mathrm{R} \qquad \textit{No } \top \textit{ left rule}$$

$$\frac{\Gamma; \Delta \Longrightarrow M : A}{\Gamma; \Delta \Longrightarrow \mathrm{inl}^B\, M : A \oplus B} \oplus\mathrm{R}_1 \qquad \frac{\Gamma; \Delta \Longrightarrow M : B}{\Gamma; \Delta \Longrightarrow \mathrm{inr}^A\, M : A \oplus B} \oplus\mathrm{R}_2$$

$$\frac{\Gamma; \Delta, w_1{:}A \Longrightarrow N_1 : C \qquad \Gamma; \Delta, w_2{:}B \Longrightarrow N_1 : C}{\Gamma; \Delta, w{:}A \oplus B \Longrightarrow \mathbf{case}\ w\ \mathbf{of}\ \mathrm{inl}\, w_1 \Rightarrow N_1 \mid \mathrm{inr}\, w_2 \Rightarrow N_2 : C} \oplus\mathrm{L}$$

$$\textit{No } \mathbf{0} \textit{ right rule} \qquad \frac{}{\Gamma; \Delta, w{:}\mathbf{0} \Longrightarrow \mathrm{abort}^C\, w : C} \mathbf{0}\mathrm{L}$$

Exponentials. Surprisingly, we do not need any explicit substitution for unrestricted variables, since the !L rule introduces a **let**-expression.

$$\frac{(\Gamma, u{:}A); \Delta \Longrightarrow M : B}{\Gamma; \Delta \Longrightarrow \lambda u{:}A.\ M : A \supset B}\supset\mathrm{R}$$

$$\frac{\Gamma; \cdot \Longrightarrow M : A \qquad \Gamma; \Delta, w_2{:}B \Longrightarrow N : C}{\Gamma; \Delta, w{:}A \supset B \Longrightarrow [(w\, M)/w_2]N : C}\supset\mathrm{L}$$

$$\frac{\Gamma; \cdot \Longrightarrow M : A}{\Gamma; \cdot \Longrightarrow\ !M : !A}!\mathrm{R}$$

$$\frac{(\Gamma, u{:}A); \Delta \Longrightarrow N : C}{\Gamma; \Delta, w{:}!A \Longrightarrow \mathbf{let}\ !u = w\ \mathbf{in}\ N : C}!\mathrm{L}$$

Cut. It is easy to check that the proof terms assigned with this system have the right type. Moreover, the resulting natural deduction terms are always normal. As can be expected from Theorem 2.14, this term assignment can be extended to derivations with cut, except that the result may no longer be normal. The assignment for the judgment $\Gamma; \Delta \overset{+}{\Longrightarrow} M : A$ is as above, with the following two additional rules.

$$\frac{\Gamma; \Delta \overset{+}{\Longrightarrow} M : A \qquad \Gamma; (\Delta', w{:}A) \overset{+}{\Longrightarrow} N : C}{\Gamma; \Delta' \times \Delta \overset{+}{\Longrightarrow} [M/w]N : C}\mathit{Cut}$$

$$\frac{\Gamma; \cdot \overset{+}{\Longrightarrow} M : A \qquad (\Gamma, u{:}A); \Delta' \overset{+}{\Longrightarrow} N : C}{\Gamma; \Delta' \overset{+}{\Longrightarrow} [M/u]N : C}\mathit{Cut}!$$

The following theorem summarizes the main properties of the term assignment system.

Theorem 4.4 (Term Assignment for Sequent Calculus)

1. *If* $\Gamma; \Delta \Longrightarrow A$ *then* $\Gamma; \Delta \Longrightarrow M : A$ *for a unique* M.
2. *If* $\Gamma; \Delta \Longrightarrow M : A$ *then* $\Gamma; \Delta \vdash M : A \uparrow$.
3. *If* $\Gamma; \Delta \overset{+}{\Longrightarrow} A$ *then* $\Gamma; \Delta \overset{+}{\Longrightarrow} M : A$ *for a unique* M.
4. *If* $\Gamma; \Delta \overset{+}{\Longrightarrow} M : A$ *then* $\Gamma; \Delta \vdash M : A$.

Proof: All by straightforward inductions over the structure of the given derivations, appealing to the substitution lemma 4.2 when necessary. □

Our proof term assignment was purposely designed to generate proof terms for the natural deduction system. This means the proof terms do not faithfully record the structure of the derivation and we cannot uniquely reconstruct a sequent derivation from a proof term. It is also possible to write out a proof term assignment which is faithful and then relate them to natural deduction proof terms (see Exercise 4.5).

4.4 Linear Type Checking

The typing rules for the linear λ-calculus are *syntax-directed* in that the principal term constructor determines the typing rule which must be used. Nonetheless, the typing rules are not immediately suitable for an efficient type-checking algorithm since we would have to guess how the linear hypotheses are to be split between the hypothesis in a number of rules.

The occurrence constraints introduced in Section 3.3 would be sufficient to avoid this choice, but they are rather complex, jeopardizing our goal of designing a simple procedure which is easy to trust. Fortunately, we have significantly more information here, since the proof term is given to us. This determines the amount of work we have to do in each branch of a derivation, and we can resolve the don't-care non-determinism directly.

Instead of guessing a split of the linear hypotheses between two premisses of a rule, we pass all linear variables to the first premiss. Checking the corresponding subterm will consume some of these variables, and we pass the remaining ones one to check the second subterms. This idea requires a judgment

$$\Gamma; \Delta_I \setminus \Delta_O \vdash M : A$$

where Δ_I represents the available linear hypotheses and $\Delta_O \subseteq \Delta_I$ the linear hypotheses not used in M. For example, the rules for the simultaneous conjunction and unit would be

$$\frac{\Gamma; \Delta_I \setminus \Delta' \vdash M : A \qquad \Gamma; \Delta' \setminus \Delta_O \vdash N : B}{\Gamma; \Delta_I \setminus \Delta_O \vdash M \otimes N : A \otimes B} \otimes\mathrm{I}$$

$$\frac{}{\Gamma; \Delta_I \setminus \Delta_I \vdash \star : A} \mathbf{1}\mathrm{I}.$$

Unfortunately, this idea breaks down when we encounter the additive unit (and only then!). Since we do not know which of the linear hypotheses might be used in a different branch of the derivation, it would have to read

$$\frac{\Delta_I \supseteq \Delta_O}{\Gamma; \Delta_I \setminus \Delta_O \vdash \langle\,\rangle : \top} \top\mathrm{I}$$

which introduces undesirable non-determinism if we were to guess which subset of Δ_I to return. In order to cirumvent this problem we return all of Δ_I, but flag it to indicate that it may not be exact, but that some of these linear hypotheses may be absorbed if necessary. In other words, in the judgment

$$\Gamma; \Delta_I \setminus \Delta_O \vdash_1 M : A$$

any of the remaining hypotheses in Δ_O need not be consumed in the other branches of the typing derivation. On the other hand, the judgment

$$\Gamma; \Delta_I \setminus \Delta_O \vdash_0 M : A$$

indicates the M uses exactly the variables in $\Delta_I - \Delta_O$.

When we think of the judgment $\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : A$ as describing an algorithm, we think of Γ, Δ_I and M as given, and Δ_O and the slack indicator i as part of the result of the computation. The type A may or may not be given—in one case it is synthesized, in the other case checked. This refines our view as computation being described as the bottom-up construction of a derivation to include parts of the judgment in different roles (as input, output, or bidirectional components). In logic programming, which is based on the notion of computation-as-proof-search, these roles of the syntactic constituents of a judgment are called *modes*. When writing a deductive system to describe an algorithm, we have to be careful to respect the modes. We discuss this further when we come to the individual rules.

Hypotheses. The two variable rules leave no slack, since besides the hypothesis itself, no assumptions are consumed.

$$\frac{}{\Gamma; (\Delta_I, w{:}A) \setminus \Delta_I \vdash_0 w : A}\,w \qquad \frac{}{(\Gamma, u{:}A); \Delta_I \setminus \Delta_I \vdash_0 u : A}\,u$$

Multiplicative Connectives. For linear implication, we must make sure that the hypothesis introduced by $\multimap$I actually was used and is not part of the residual hypothesis Δ_O. If there is slack, we can simply erase it.

$$\frac{\Gamma; (\Delta_I, w{:}A) \setminus \Delta_O \vdash_i M : B \quad \text{where } i = 1 \text{ or } w \text{ not in } \Delta_O}{\Gamma; \Delta_I \setminus (\Delta_O - w{:}A) \vdash_i \hat{\lambda} w{:}A.\, M : A \multimap B}\,\multimap\mathrm{I}^w$$

$$\frac{\Gamma; \Delta_I \setminus \Delta' \vdash_i M : A \multimap B \qquad \Gamma; \Delta' \setminus \Delta_O \vdash_k N : A}{\Gamma; (\Delta_I \setminus \Delta_O) \vdash_{i \vee k} M\,\hat{}\,N : B}\,\multimap\mathrm{E}$$

Here $i \vee k = 1$ if $i = 1$ or $k = 1$, and $i \vee k = 0$ otherwise. This means we have slack in the result, if either of the two premisses permits slack.

$$\frac{\Gamma; \Delta_I \setminus \Delta' \vdash_i M : A \qquad \Gamma; \Delta' \setminus \Delta_O \vdash_k N : B}{\Gamma; \Delta_I \setminus \Delta_O \vdash_{i \vee k} M \otimes N : A \otimes B} \otimes\mathrm{I}$$

$$\frac{\begin{array}{l}\Gamma; \Delta_I \setminus \Delta' \vdash_i M : A \otimes B \\ \Gamma; (\Delta', w_1{:}A, w_2{:}B) \setminus \Delta_O \vdash_k N : C \\ \text{where } k = 1 \text{ or } w_1 \text{ and } w_2 \text{ not in } \Delta_O\end{array}}{\Gamma; \Delta_I \setminus (\Delta_O - w_1{:}A - w_2{:}B) \vdash_{i \vee k} \mathbf{let}\ w_1 \otimes w_2 = M\ \mathbf{in}\ N : C} \otimes\mathrm{E}^{w_1, w_2}$$

In the $\otimes$E rule we stack the premisses on top of each other since they are too long to fit on one line. The unit type permits no slack.

$$\frac{}{\Gamma; \Delta_I \setminus \Delta_I \vdash_0 \star : \mathbf{1}} \mathbf{1}\mathrm{I}$$

$$\frac{\Gamma; \Delta_I \setminus \Delta' \vdash_i M : \mathbf{1} \qquad \Gamma; \Delta' \setminus \Delta_O \vdash_k N : C}{\Gamma; \Delta_I \setminus \Delta_O \vdash_{i \vee k} \mathbf{let}\ \star = M\ \mathbf{in}\ N : C} \mathbf{1}\mathrm{E}$$

Additive Connectives. The mechanism of passing and consuming resources was designed to eliminate unwanted non-determinism in the multiplicative connectives. This introduces complications in the additives, since we have to force premisses to consume exactly the same resources. We write out four version of the &I rule.

$$\frac{\Gamma; \Delta_I \setminus \Delta'_O \vdash_0 M : A \quad \Gamma; \Delta_I \setminus \Delta''_O \vdash_0 N : B \quad \Delta'_O = \Delta''_O}{\Gamma; \Delta_I \setminus (\Delta'_O \cap \Delta''_O) \vdash_0 \langle M, N \rangle : A \& B} \&\mathrm{I}_{00}$$

$$\frac{\Gamma; \Delta_I \setminus \Delta'_O \vdash_0 M : A \quad \Gamma; \Delta_I \setminus \Delta''_O \vdash_1 N : B \quad \Delta'_O \subseteq \Delta''_O}{\Gamma; \Delta_I \setminus (\Delta'_O \cap \Delta''_O) \vdash_0 \langle M, N \rangle : A \& B} \&\mathrm{I}_{10}$$

$$\frac{\Gamma; \Delta_I \setminus \Delta'_O \vdash_1 M : A \quad \Gamma; \Delta_I \setminus \Delta''_O \vdash_0 N : B \quad \Delta'_O \supseteq \Delta''_O}{\Gamma; \Delta_I \setminus (\Delta'_O \cap \Delta''_O) \vdash_0 \langle M, N \rangle : A \& B} \&\mathrm{I}_{01}$$

$$\frac{\Gamma; \Delta_I \setminus \Delta'_O \vdash_1 M : A \quad \Gamma; \Delta_I \setminus \Delta''_O \vdash_1 N : B}{\Gamma; \Delta_I \setminus (\Delta'_O \cap \Delta''_O) \vdash_1 \langle M, N \rangle : A \& B} \&\mathrm{I}_{11}$$

Note that in $\&\mathrm{I}_{00}$, $\Delta'_O \cap \Delta''_O = \Delta'_O = \Delta''_O$ by the condition in the premiss. Similarly for the other rules. We chose to present the rules in a uniform way despite this redundancy to highlight the similarities. Only if both premisses permit slack do we have slack overall.

$$\frac{\Gamma; \Delta \setminus \Delta_O \vdash_i M : A \& B}{\Gamma; \Delta_I \setminus \Delta_O \vdash_i \mathrm{fst}\, M : A}\,\&\mathrm{E_L} \qquad \frac{\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : A \& B}{\Gamma; \Delta_I \setminus \Delta_O \vdash_i \mathrm{snd}\, M : B}\,\&\mathrm{E_R}$$

Finally, we come to the reason for the slack indicator.

$$\frac{}{\Gamma; \Delta_I \setminus \Delta_I \vdash_1 \langle\rangle : \top}\,\top\mathrm{I} \qquad \textit{No } \top \textit{ elimination}$$

The introduction rules for disjunction are direct.

$$\frac{\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : A}{\Gamma; \Delta_I \setminus \Delta_O \vdash_i \mathrm{inl}^B : A \oplus B}\,\oplus\mathrm{I_L} \qquad \frac{\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : B}{\Gamma; \Delta_I \setminus \Delta_O \vdash_i \mathrm{inr}^A : A \oplus B}\,\oplus\mathrm{I_R}$$

The elimination rule for disjunction combines resource propagation (as for multiplicatives) introduction of hypothesis, and resource coordination (as for additives) and is therefore somewhat tedious. It is left to Exercise 4.6. The **0**E rule permits slack, no matter whether the derivation of the premiss permits slack.

$$\textit{No } \mathbf{0} \textit{ introduction} \qquad \frac{\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : \mathbf{0}}{\Gamma; \Delta_I \setminus \Delta_O \vdash_1 \mathrm{abort}^C\, M : C}\,\mathbf{0}\mathrm{E}$$

Exponentials. Here we can enforce the emptiness of the linear context directly.

$$\frac{(\Gamma, u{:}A); \Delta_I \setminus \Delta_O \vdash_i M : B}{\Gamma; \Delta_I \setminus \Delta_O \vdash_i \lambda u{:}A.\, M : A \supset B}\,\supset\mathrm{I}^u$$

$$\frac{\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : A \supset B \qquad \Gamma; \cdot \setminus \Delta^* \vdash_k N : A}{\Gamma; \Delta_I \setminus \Delta_O \vdash_i M\, N : B}\,\supset\mathrm{E}$$

Here Δ^* will always have to be $\cdot$ (since it must be a subset of $\cdot$) and k is irrelevant. The same is true in the next rule.

$$\frac{\Gamma; \cdot \setminus \Delta^* \vdash_i M : A}{\Gamma; \Delta_I \setminus \Delta_I \vdash_0\, !M : !A}\,!\mathrm{I}$$

$$\frac{\Gamma; \Delta_I \setminus \Delta' \vdash_i M : !A \qquad (\Gamma, u{:}A); \Delta' \setminus \Delta_O \vdash_k N : C}{\Gamma; \Delta_I \setminus \Delta_O \vdash_{i \vee j} \mathbf{let}\ !u = M\ \mathbf{in}\ N : C}\,!\mathrm{E}^u$$

The desired soundness and completeness theorem for the algorithmic typing judgment must first be generalized before it can be proved by induction. For this generalization, the mode (input and output) of the constituents of the judgment is a useful guide. For example, in the completness direction (3), we can expect to distinguish cases based on the slack indicator which might be returned when we ask the question if there are Δ_O and i such that $\Gamma; \Delta \setminus \Delta_O \vdash_i M : A$ for the given Γ, Δ, M and A.

Lemma 4.5 (Properties of Algorithmic Type Checking)

1. *If* $\Gamma; \Delta_I \setminus \Delta_O \vdash_0 M : A$ *then* $\Delta_I \supseteq \Delta_O$ *and* $\Gamma; \Delta_I - \Delta_O \vdash M : A$.

2. *If* $\Gamma; \Delta_I \setminus \Delta_O \vdash_1 M : A$ *then* $\Delta_I \supseteq \Delta_O$ *and for any* Δ *such that* $\Delta_I \supseteq \Delta \supseteq \Delta_I - \Delta_O$ *we have* $\Gamma; \Delta \vdash M : A$.

3. *If* $\Gamma; \Delta \vdash M : A$ *then either*

   (a) $\Gamma; (\Delta' \times \Delta) \setminus \Delta' \vdash_0 M : A$ *for any* Δ', *or*

   (b) $\Gamma; (\Delta' \times \Delta) \setminus (\Delta' \times \Delta_O) \vdash_1 M : A$ *for all* Δ' *and some* $\Delta_O \subseteq \Delta$.

Proof: By inductions on the structure of the given derivations.[1] Items (1) and (2) must be proven simultaneously. □

From this lemma, the soundness and completeness of algorithmic type checking follow directly.

Theorem 4.6 (Algorithmic Type Checking)
$\Gamma; \Delta \vdash M : A$ *if and only if either*

1. $\Gamma; \Delta \setminus \cdot \vdash_0 M : A$, *or*

2. $\Gamma; \Delta \setminus \Delta' \vdash_1 M : A$ *for some* Δ'.

Proof: Directly from Lemma 4.5 □

4.5 Exercises

Exercise 4.1 Prove that if $\Gamma; \Delta \vdash M : A$ and $\Gamma; \Delta \vdash M : A'$ then $A = A'$.

Exercise 4.2 A function in a functional programming language is called *strict* if it is guaranteed to use its argument. Strictness is an important concept in the implementation of lazy functional languages, since a strict function can evaluate its argument eagerly, avoiding the overhead of postponing its evaluation and later memoizing its result.

In this exercise we design a λ-calculus suitable as the core of a functional language which makes strictness explicit at the level of types. Your calculus should contain an unrestricted function type $A \to B$, a strict function type $A \twoheadrightarrow B$, a vacuous function type $A \dashrightarrow B$, a full complement of operators refining product and disjoint sum types as for the linear λ-calculus, and a modal operator to internalize the notion of closed term as in the linear λ-calculus. Your calculus should not contain quantifiers.

1. Show the introduction and elimination rules for all types, including their proof terms.

[1] [*check*]

2. Given the reduction and expansions on the proof terms.

3. State (without proof) the valid substitution principles.

4. If possible, give a translation from types and terms in the strict λ-calculus to types and terms in the linear λ-calculus such that a strict term is well-typed if and only if its linear translation is well-typed (in an appropriately translated context).

5. Either sketch the correctness proof for your translation in each direction by giving the generalization (if necessary) and a few representative cases, or give an informal argument why such a translation is not possible.

Exercise 4.3 Give an example which shows that the substitution $[M/w]N$ must be capture-avoiding in order to be meaningful. *Variable capture* is a situation where a bound variable w' in N occurs free in M, and w occurs in the scope of w'. A similar definition applies to unrestricted variables.

Exercise 4.4 Give a counterexample to the conjecture that if $M \longrightarrow_\beta M'$ and $\Gamma; \Delta \vdash M' : A$ then $\Gamma; \Delta \vdash M : A$. Also, either prove or find a counterexample to the claim that if $M \longrightarrow_\eta M'$ and $\Gamma; \Delta \vdash M' : A$ then $\Gamma; \Delta \vdash M : A$.

Exercise 4.5 The proof term assignment for sequent calculus identifies many distinct derivations, mapping them to the same natural deduction proof terms. Design an alternative system of proof terms from which the sequent derivation can be reconstructed uniquely (up to weakening of unrestricted hypotheses and absorption of linear hypotheses in the $\top$R rule).

1. Write out the term assignment rules for all propositional connectives.

2. Give a calculus of reductions which corresponds to the initial and principal reductions in the proof of admissibility of cut.

3. Show the reduction rule for the dereliction cut.

4. Show the reduction rules for the left and right commutative cuts.

5. Sketch the proof of the subject reduction properties for your reduction rules, giving a few critical cases.

6. Write a translation judgment $S \Longrightarrow M$ from faithful sequent calculus terms to natural deduction terms.

7. Sketch the proof of type preservation for your translation, showing a few critical cases.

Exercise 4.6 Supply the missing rules for $\oplus$E in the definition of the judgment $\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : A$ and show the corresponding cases in the proof of Lemma 4.5.