ConstLog: Constructive Logic

Lecture Notes on Certifying Theorem Provers

Frank Pfenning André Platzer

Carnegie Mellon University || Karlsruhe Institute of Technology

Lecture 13

1 Introduction

The previous lectures identified a decision procedure for intuitionistic propositional logic using contraction-free inversion. How do we trust a theorem prover or decision procedure for a logic? Ideally, we would prove it correct (constructively, of course!) and extract the prover implementation from the proof. For example, for the contraction-free sequent calculus for intuitionistic propositional logic (G4ip) we could try to prove: Every sequent $\Gamma \longrightarrow_{G4ip} A$, either has a deduction $\mathcal{D}$ or not. This is not an easy enterprise: We have to generalize it and then carry out a well-founded induction over the weighted multiset ordering for its progress measure. Then we realize that the extracted decision procedure is not very useful because it does not account for invertible rules. So we write another system, which we (a) have to prove equivalent to the first one, and (b) we have to then prove decidable. Both of these are not easy, but in the end it will give us a warm and fuzzy feeling to have a definitively correct implementation. After a couple of months of hard work. Then we realize another optimization can make our decision procedure more efficient and we have to go back to the drawing board with our proof.

In this lecture we develop an alternative approach. In this approach we first write a small, trusted *proof checker* for the logic at hand. Ideally we use the simplest and most fundamental formulation of the logic (that usually means natural deduction) to keep this checker as clean, short, and simple as possible. Constructive proofs are fundamentally related to programs, and propositions to types, so this is the same as writing a type checker for a small core programming language.

As a second step we instrument our decision procedure or theorem prover to not just answer "yes" or "no", but produce a proof term whenever the answer is "yes". This proof term can then be independently checked by our small, trusted checker. Only if the checker also says "yes" do we accept the answer of the decision procedure. This does *not* guarantee that the prover is correct. It might, for example, incorrectly say "yes" on other propositions and supply an incorrect proof term. It could also say "no" even though the proposition is provable and we may never discover it. But, generally speaking, we are more interested in theorems and proofs, than in nonprovability, so having our prover certify theorems is a big step forward.

In this lecture we develop this in two steps: we first develop the proof checker and then we instrument the sequent calculus to produce proof terms.

2 Designing a Proof Checker

Natural deduction is the simplest and purest form of logic specification for intuitionistic logic, using only one judgment (A *true*) and the notion of hypothetical judgment. However, it is not ideal if we try to develop a checker for proof terms. For example, consider the rule

$$\frac{M : A}{\mathbf{inl}\, M : A \vee B} \vee I_1$$

The term $\mathbf{inl}\, M$ in this rule has some inherent ambiguity because it serves as a proof term for $A \vee B$ *for any* B! In order to enforce uniqueness, we would have to annotate the constructor with the actual proposition B, for example:

$$\frac{\mathbf{inl}_B\, M : A \vee B}{M : A} \vee I_1$$

We would soon find that our proof terms are littered with propositions, which is annoying and can also be inefficient in terms of space needed to represent proofs and time for checking them.

Now we should all remember the notion of *verification*. Recall that verifications proceed with introduction rules from below and elimination rules from above. When viewed in these two directions, *all* rules just break down the proposition we are trying to prove into its constituents. Our motivation was foundational: the meaning of a proposition should depend only on its constituents. But we can now reap the benefits in terms of proof checking: a proof term that we extract from a verification should not need any internal propositions, because verifications are only ever decomposed!

Let's start with conjunction, always a good and easy place to start.

$$\frac{A\uparrow \quad B\uparrow}{A \wedge B\uparrow} \wedge I \qquad \frac{M : A\uparrow \quad N : B\uparrow}{\langle M, N\rangle : A \wedge B\uparrow} \wedge I$$

This works perfectly: to check the proof term $\langle M, N\rangle$ against the proposition $A \wedge B$ we can check its subterm M against A and N against B.

The elimination rules, however, do not work like that.

$$\frac{A \wedge B\downarrow}{A\downarrow} \wedge E_1 \qquad \frac{A \wedge B\downarrow}{B\downarrow} \wedge E_2$$

$$\frac{R : A \wedge B\downarrow}{\mathbf{fst}\, R : A\downarrow} \wedge E_1 \qquad \frac{R : A \wedge B\downarrow}{\mathbf{snd}\, R : B\downarrow} \wedge E_2$$

We *can not* check **fst** R against A by checking R against $A \wedge B$ because we do not know B. But we are reading the rule the wrong way! In a verification the introduction rules are read bottom-up and the elimination rules are read top-down.

Read top-down, the $\wedge E_1$ expresses the following: *"If R has type $A \wedge B$ then **fst** R will have type A."* Fortunately, this is entirely sensible (although at the moment we can't be sure). With this discipline of reading direction in mind, it also becomes clearer why type annotations are never needed for verifications. Introduction rules decompose verifications upwards to subpropositions, while elimination rules decompose uses downwards to subpropositions. Either way, the remaining context makes the propositions/types clear.

Distinguishing the two judgments, we say that verifications $A\uparrow$ are annotated with *checkable terms* N, and propositions whose use is justified with $A\downarrow$ are annotated with *synthesizing terms* R. We are shooting for the following theorem (to be refined later, see Theorem 1):

(i) Given N and A, either $N : A\uparrow$ or not, and

(ii) given R there exists a proposition A such that $R : A\downarrow$ or there exists no such A.

We say that N *checks against* A and R *synthesizes* A.

So far we have

$$\begin{array}{llll} \text{Checkable terms} & M, N & ::= & \langle M, N\rangle \mid \ldots \\ \text{Synthesizing terms} & R & ::= & \mathbf{fst}\, R \mid \mathbf{snd}\, R \mid \ldots \end{array}$$

Continuing with implication:

$$\dfrac{\begin{array}{c}\overline{A\downarrow}^{\;u}\\ \vdots \\ B\uparrow\end{array}}{A \supset B\uparrow} \supset I^u \qquad \dfrac{\begin{array}{c}\overline{u : A\downarrow}^{\;u}\\ \vdots \\ M : B\uparrow\end{array}}{(\mathsf{fn}\ \ u \Rightarrow M) : A \supset B\uparrow} \supset I^u$$

which means that functions are checkable while variables are synthesizing. How about the elimination rule? Just based on the directions of the inferences in verifications, we get the following:

$$\dfrac{A \supset B\downarrow \quad A\uparrow}{B\downarrow} \supset E \qquad \dfrac{R : A \supset B\downarrow \quad M : A\uparrow}{R\, M : B\downarrow} \supset E$$

Recall that developing verifications from below and uses from above meet at the judgmental $\downarrow\uparrow$ rule. We model this by allowing any synthesizing term R as a checkable one. Intuitively, this should be okay because we can first synthesize the type of R and then just compare it to the given type.

$$\dfrac{A\downarrow}{A\uparrow} \downarrow\uparrow \qquad \dfrac{R : A\downarrow}{R : A\uparrow} \downarrow\uparrow$$

Filling in more details in our picture, we now have:

$$\begin{array}{llll} \text{Checkable terms} & M, N & ::= & \langle M, N\rangle \mid (\mathsf{fn}\ u \Rightarrow M) \mid R \mid \ldots \\ \text{Synthesizing terms} & R & ::= & \mathbf{fst}\, R \mid \mathbf{snd}\, R \mid u \mid R\, M \mid \ldots \end{array}$$

The other connectives, summarized in Fig. 1, don't present any more new and interesting ideas. We do note, for example, that **inl** M ends up as being a *checkable term*, which avoids the problem we encountered for natural deduction in general. In particular, we don't need to annotate **inl** with a type, because **inl** M is always *checked against* $A \vee B$. In summary (so far):

$$\frac{M : A\uparrow \quad N : B\uparrow}{\langle M, N\rangle : A \wedge B\uparrow} \wedge I \qquad \frac{R : A \wedge B\downarrow}{\mathbf{fst}\, R : A\downarrow} \wedge E_1 \qquad \frac{R : A \wedge B\downarrow}{\mathbf{snd}\, R : B\downarrow} \wedge E_2$$

$$\frac{\begin{array}{c}\overline{u : A\downarrow}^{\,u} \\ \vdots \\ M : B\uparrow\end{array}}{(\mathsf{fn}\ u \Rightarrow M) : A \supset B\uparrow} \supset I^u \qquad \frac{R : A \supset B\downarrow \quad M : A\uparrow}{R\, M : B\downarrow} \supset E$$

$$\frac{}{(\,) : \top\uparrow} \top I \qquad \text{no } \top E$$

$$\frac{M : A\uparrow}{\mathbf{inl}\, M : A \vee B\uparrow} \vee I_1 \quad \frac{N : B\uparrow}{\mathbf{inr}\, N : A \vee B\uparrow} \vee I_2$$

$$\frac{R : A \vee B\downarrow \quad \begin{array}{c}\overline{u : A\downarrow}^{\,u} \\ \vdots \\ M : C\uparrow\end{array} \quad \begin{array}{c}\overline{v : B\downarrow}^{\,v} \\ \vdots \\ N : C\uparrow\end{array}}{(\mathbf{case}\ R\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow M \mid \mathbf{inr}\, v \Rightarrow N) : C\uparrow} \vee E$$

$$\text{no } \bot I \qquad \frac{R : \bot\downarrow}{\mathbf{abort}\, R : C\uparrow} \bot E$$

$$\frac{R : A\downarrow}{R : A\uparrow} \downarrow\uparrow$$

Figure 1: Proof checking rules

$$\begin{array}{lll} \text{Checkable terms} & M, N ::= & \langle M, N\rangle \mid (\mathsf{fn}\ u \Rightarrow M) \mid R \\ & & \mid \mathbf{inl}\, M \mid \mathbf{inr}\, N \mid (\mathbf{case}\ R\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow M \mid v \Rightarrow N) \\ & & \mid \mathbf{abort}\, R \\ \text{Synthesizing terms}\ R & ::= & \mathbf{fst}\, R \mid \mathbf{snd}\, R \mid u \mid R\, M \mid \ldots \end{array}$$

In order to formulate our theorem, we make the hypothetical judgment explicit. Let $\Delta = (u_1{:}A_1{\downarrow}, \ldots, u_n{:}A_n{\downarrow})$ and recall from Lecture 2 the notation $\Delta \vdash M : A{\uparrow}$ for this natural deduction (similarly for other judgments):

$$\begin{array}{c} u_1{:}A_1{\downarrow} \quad \ldots \quad u_n{:}A_n{\downarrow} \\ \vdots \\ M : A \uparrow \end{array}$$

In lecture we worked our way up to the theorem, but hopefully have enough intuition now to state it directly. We refer to this as *bidirectional type checking* because we interleave checking (bottom-up) and synthesis (top-down).

Theorem 1 (Decidability of bidirectional type checking)**.**

(i) Given Δ, M, and A, either $\Delta \vdash M : A \uparrow$ or $\Delta \not\vdash M : A \uparrow$, and

(ii) Given Δ and R, either there exists a unique *A such that $\Delta \vdash R : A \downarrow$ or there exists no A such that $\Delta \vdash R : A \downarrow$.*

Proof: By mutual induction on the structure of M and R, simultaneously for all Δ and, in part (i), all A. For part (i) alone, we may have been able to use the structure of A, but for part (ii) we do not even have an A to begin with. Δ does not give us much structure to work with, which leaves the structure of M and R.

A subtle point in the case for $\downarrow\uparrow$ is that the term R does not become smaller, so we impose an induction order in which (ii) $<$ (i). This means in an appeal to the induction hypothesis (ii) in a case for (i) the proof term can remain the same, but in an appeal to (i) from (ii), the proof term must become strictly smaller (which, fortunately, it does in all the cases).

We show four cases.

Case: $M = (\mathsf{fn}\ u \Rightarrow M_2)$ for some M_2. Then we distinguish cases on A. We refer to *inversion* when a judgment could have been derived by no rule (and therefore does not hold) or by just one rule (and therefore its premises would have to hold, too).

Subcase: $A = A_1 \supset A_2$ for some A_1 and A_2. Then

Either $\Delta, u{:}A_1{\downarrow} \vdash M_2 : A_2 \uparrow$ or not	by i.h.(i) on M_2
$\Delta, u{:}A_1{\downarrow} \vdash M_2 : A_2 \uparrow$	first subsubcase
$\Delta \vdash \mathsf{fn}\ u \Rightarrow M_2 : A_1 \supset A_2 \uparrow$	by rule $\supset I$
$\Delta, u{:}A_1{\downarrow} \not\vdash M_2 : A_2 \uparrow$	second subsubcase
$\Delta \not\vdash (\mathsf{fn}\ u \Rightarrow M_2) : (A_1 \supset A_2) \uparrow$	by inversion

Subcase: $A \neq A_1 \supset A_2$ for all A_1 and A_2. Then

$\Delta \not\vdash (\mathsf{fn}\ u \Rightarrow M_2) : A$	by inversion

Case: $M = R\,N$ for some R and N.

$\Delta \vdash R : B \downarrow$ for a unique B or there is no such B — by i.h.(ii) on R
$\Delta \vdash R : B \downarrow$ for a unique B — first subcase
  $B = B_1 \supset B_2$ for some B_1 and B_2 — first subsubcase
  $\Delta \vdash N : B_1 \uparrow$ or $\Delta \not\vdash N : B_1 \uparrow$ — by i.h.(i) on N
    $\Delta \vdash N : B_1 \uparrow$ — first sub^{3}case
    $\Delta \vdash R\,N : B_2 \downarrow$ — by rule $\supset E$
    B_2 is unique — by inversion and uniqueness of B

    $\Delta \not\vdash N : B_1 \uparrow$ — second sub^{3}case
    $\Delta \not\vdash R\,N : A \downarrow$ for any A — by inversion and uniqueness of B

  $B \neq B_1 \supset B_2$ for any B_1 and B_2 — second subsubcase
  $\Delta \not\vdash R\,N : A \downarrow$ for any A — by inversion and uniqueness of B

$\Delta \not\vdash R : B \downarrow$ for any B — second subcase
$\Delta \not\vdash R\,N : A \downarrow$ for any A — by inversion

Case: $M = R$.

$\Delta \vdash R : A' \downarrow$ for a unique A' or there is no such A' — by i.h.(ii) on R
Either $A = A'$ or $A \neq A'$ — by decidability of equality on propositions
  $A = A'$ — first subcase
  $\Delta \vdash R : A \uparrow$ — by rule $\uparrow\downarrow$

  $A \neq A'$ — second subcase
  $\Delta \not\vdash R : A \uparrow$ — by inversion and uniqueness of A'.

Case: $R = u$.

$\Delta \vdash u : A \downarrow$ iff $u : A \in \Delta$ — by hypothetical judgment
A is unique — because declarations $u{:}A$ in Δ are unique

□

From this proof, we can extract two functions of the following types in ML:

```
check : (var * prop) list -> chk_term -> prop -> bool
synth : (var * prop) list -> syn_term -> prop option
```

Here, the checkable terms have type `chk_term`, synthesizing terms have type `syn_term`, and propositions are represented in the type `prop`. A context is represented as a list of pairs of variables and their types. There are a lot of cases to consider, but exploiting the pattern-matching facilities in ML it remains small and manageable. In a realistic implementation, one would want to print error messages or return error code instead of just `false` (for `check`) and `NONE` (for `synth`), but this is an extra-logical refinement.

As an aside, in such a representation of terms we can not just include every synthesizable term as a checkable term, but we would need an explicit constructor that creates a checkable term from a synthesizable term. Such a constructor makes it less intuitive to write terms, so we can instead just have a single type of `term` and have the `check` and `synth` functions sort out which must be which.

3 Instrumenting a Theorem Prover

The next step will be to instrument some theorem prover so it can produce a proof term in case it succeeds. What helps us here is that (a) we already designed the sequent calculus as a purely bottom-up system for searching for a verification, and (b) more efficient search procedures (such as G4ip, which is in fact a decision procedure) are actually presented as refinements of the sequent calculus. As an example we consider here G4, which is the restricted sequent calculus from Lecture 11, Section 2 in Fig. 2. Recall that this is an optimization of the system which we obtained from translating

$$\frac{}{\Gamma, P \longrightarrow P}\ \mathsf{id}$$

$$\frac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow B}{\Gamma \longrightarrow A \wedge B}\ \wedge R \qquad \frac{\Gamma, A, B \longrightarrow C}{\Gamma, A \wedge B \longrightarrow C}\ \wedge L$$

$$\frac{}{\Gamma \longrightarrow \top}\ \top R \qquad \frac{\Gamma \longrightarrow C}{\Gamma, \top \longrightarrow C}\ \top L$$

$$\frac{\Gamma \longrightarrow A}{\Gamma \longrightarrow A \vee B}\ \vee R_1 \qquad \frac{\Gamma \longrightarrow B}{\Gamma \longrightarrow A \vee B}\ \vee R_2 \qquad \frac{\Gamma, A \longrightarrow C \quad \Gamma, B \longrightarrow C}{\Gamma, A \vee B \longrightarrow C}\ \vee L$$

$$\text{no } \bot R \text{ rule} \qquad \frac{}{\Gamma, \bot \longrightarrow C}\ \bot L$$

$$\frac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B}\ \supset R \qquad \frac{\Gamma, A \supset B \longrightarrow A \quad \Gamma, B \longrightarrow C}{\Gamma, A \supset B \longrightarrow C}\ \supset L$$

Figure 2: G4 rules

$$\begin{array}{c} A_1{\downarrow}, \ldots, A_n{\downarrow} \\ \vdots \\ C\,{\uparrow} \end{array} \qquad \text{to} \qquad \begin{array}{c} \vdots \\ A_1, \ldots, A_n \Longrightarrow C \end{array}$$

The assignment of synthesizing and checking terms to the verifications for natural deduction suggests the corresponding annotations on sequents.

$$\begin{array}{c} R_1{:}A_1{\downarrow}, \ldots, R_n{:}A_n{\downarrow} \\ \vdots \\ N : C\,{\uparrow} \end{array} \qquad \text{to} \qquad \begin{array}{c} \vdots \\ R_1{:}A_1, \ldots, R_n{:}A_n \Longrightarrow N : C \end{array}$$

Furthermore, since propositions $A\,{\downarrow}$, working downwards from hypotheses, are always fully justified as we are searching for a proof while $C\,{\uparrow}$ is unknown until we complete the proof, the theorem we are aiming for is:

Theorem 2 (Sequent Proof Annotation)**.**
For every deduction $A_1, \ldots, A_n \longrightarrow C$ and all hypotheses Δ with $\Delta \vdash R_1{:}A_1\,{\downarrow}$ and $\ldots, \Delta \vdash R_n{:}A_n\,{\downarrow}$ there exists a proof term N such that $R_1{:}A_1, \ldots, R_n{:}A_n \longrightarrow N : C$ and $\Delta \vdash N : C\,{\uparrow}$

Proof: By induction on the structure of $\overset{\mathcal{D}}{A_1, \ldots, A_n \longrightarrow C}$ □

Rather than showing the proof case by case, we develop the proof term annotation case by case. Let's start with

$$\frac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B} \supset R$$

We can annotate the antecedents in Γ with a sequence $\rho = (R_1, \ldots, R_n)$ of synthesizing terms, which we abbreviate by $\rho : \Gamma$. By induction hypothesis, and with a fresh variable u, we get some proof term N such that

$$\rho : \Gamma, u : A \longrightarrow N : B$$

from which we can glean that the annotated rule should be

$$\rho : \Gamma \longrightarrow (\mathsf{fn}\ \ u \Rightarrow N) : A \supset B$$

Moreover, since $\Delta, u{:}A\downarrow \vdash u : A \downarrow$, we have $\Delta, u{:}A\downarrow \vdash N : B \uparrow$ as $\mathsf{fn}\ \ u \Rightarrow N$ has type $A \supset B$ and hence $\Delta \vdash (\mathsf{fn}\ \ u \Rightarrow N) : A \supset B \downarrow$ by $\supset I$. This means our annotated rule should be

$$\frac{\rho : \Gamma, u : A \longrightarrow N : B}{\rho : \Gamma \longrightarrow (\mathsf{fn}\ \ u \Rightarrow N) : A \supset B} \supset R$$

As a second case, we consider $\supset L$. We have

$$\frac{\Gamma, A \supset B \longrightarrow A \quad \Gamma, B \longrightarrow C}{\Gamma, A \supset B \longrightarrow C} \supset L$$

We are also given $\rho : \Gamma, R : A \supset B$. By induction hypothesis, we get an M such that

$$\rho : \Gamma, R : A \supset B \longrightarrow M : A \quad \text{and} \quad \Delta \vdash M : A$$

In order to be able to apply the induction hypothesis to the second premise, we need some (synthesizing) term, denoting a proof of B. We have both $R : A \supset B$ and $M : A$, so we have $R\,M : B$ and obtain from the induction hypothesis some proof term N such that

$$\rho : \Gamma, R\,M : B \longrightarrow N : C \quad \text{and} \quad \Delta \vdash N : C$$

which allows us to choose $N : C$ in the conclusion as well. Summarizing this as a rule, we get

$$\frac{\rho : \Gamma, R : A \supset B \longrightarrow M : A \quad \rho : \Gamma, R\,M : B \longrightarrow N : C}{\rho : \Gamma, R : A \supset B \longrightarrow N : C} \supset L$$

As a final case, we consider the identity rule.

$$\frac{}{\Gamma, P \longrightarrow P} \mathsf{id}$$

We also have $\rho : \Gamma, R : P$ so we can choose the proof term $N = R$:

$$\frac{}{\rho : \Gamma, R : P \longrightarrow R : P}\ \mathsf{id}$$

and $\Delta \vdash R : P \downarrow$ implies $\Delta \vdash R : P \uparrow$ by judgmental rule $\downarrow\uparrow$.

We now summarize all the rules in Fig 3, but reusing the notation Γ for $\rho : \Gamma$ to make the rules more readable.

$$\frac{}{\Gamma, R : P \longrightarrow R : P}\ \mathsf{id}$$

$$\frac{\Gamma \longrightarrow M : A \quad \Gamma \longrightarrow N : B}{\Gamma \longrightarrow \langle M, N\rangle : A \wedge B}\ \wedge R \qquad \frac{\Gamma, \mathbf{fst}\, R : A, \mathbf{snd}\, R : B \longrightarrow N : C}{\Gamma, R : A \wedge B \longrightarrow N : C}\ \wedge L$$

$$\frac{}{\Gamma \longrightarrow (\,) : \top}\ \top R \qquad \frac{\Gamma \longrightarrow N : C}{\Gamma, R : \top \longrightarrow N : C}\ \top L$$

$$\frac{\Gamma \longrightarrow M : A}{\Gamma \longrightarrow \mathbf{inl}\, M : A \vee B}\ \vee R_1 \qquad \frac{\Gamma \longrightarrow N : B}{\Gamma \longrightarrow \mathbf{inr}\, N : A \vee B}\ \vee R_2$$

$$\frac{\Gamma, u : A \longrightarrow N_1 : C \quad \Gamma, v : B \longrightarrow N_2 : C}{\Gamma, R : A \vee B \longrightarrow (\mathbf{case}\ R\ \ \mathbf{of}\ \ \mathbf{inl}\, u \Rightarrow N_1 \mid \mathbf{inr}\, v \Rightarrow N_2) : C}\ \vee L$$

$$\text{no } \bot R \text{ rule} \qquad \frac{}{\Gamma, R : \bot \longrightarrow \mathbf{abort}\, R : C}\ \bot L$$

$$\frac{\Gamma, u : A \longrightarrow N : B}{\Gamma \longrightarrow (\mathsf{fn}\ \ u \Rightarrow N) : A \supset B}\ \supset R$$

$$\frac{\Gamma, R : A \supset B \longrightarrow M : A \quad \Gamma, (R\, M) : B \longrightarrow N : C}{\Gamma, R : A \supset B \longrightarrow N : C}\ \supset L$$

Figure 3: Annotated sequent calculus proof rules

4 Justifying Cut

Providing a similar proof term assignment for full G4ip requires one further thought: how would we handle the rule of cut if it were needed? Recall that G4ip's derivations use helpful cuts. Let's come back to our inductive proof of Theorem 2 and consider the case that we would have cut as a rule (but we write it as the admissible rule it is):[1]

$$\frac{\Gamma \longrightarrow A \quad \Gamma, A \longrightarrow C}{\Gamma \longrightarrow C}\ \mathsf{cut}$$

[1]Note how important the absence of cut is for inversion arguments.

We know, by the fact that cut is admissible, that there is a cut-free proof of the conclusion. We could construct this using the proof of admissibility (which was constructive) and annotate the result. Unfortunately, the result could be quite large, since cut elimination can very significantly explode the size of the proof.

Alternatively, we can make up a new kind of proof term for this rule, standing in for what cut elimination might compute. First, since we have $\rho : \Gamma$ we can appeal to the induction hypothesis and construct an M such that

$$\rho : \Gamma \longrightarrow M : A$$

If only we could turn the checkable term M into a synthesizing term R, we could use this to justify the antecedent A in the second premise. For this we create a new construct $(M : A)$ in the syntax for synthesizing terms. It synthesized A (which is therefore unique) if M checks against A. But M (which we obtained from an appeal to the induction hypothesis) was a checkable term! Then, again by induction hypothesis we obtain

$$\rho : \Gamma, (M : A) : A \longrightarrow N : C$$

which we can use to conclude

$$\rho : \Gamma \longrightarrow N : C$$

as required. This leads to the rule

$$\frac{\Gamma \longrightarrow M : A \quad \Gamma, (M : A) : A \longrightarrow N : C}{\Gamma \longrightarrow N : C}\ \mathsf{cut}$$

We would need the new rule

$$\frac{\Delta \vdash M : A \uparrow}{\Delta \vdash (M : A) : A \downarrow}\ \uparrow\downarrow$$

For verifications, this cannot be a primitive rule (since it destroys the meaning explanation for the connectives), but we can use it in the type checker if we extend our syntax with $(M : A)$ as a new form of synthesizing term.

Alternatively, we could use the let form by justifying A by a variable u which is discharged using the verification of A in the conclusion:

$$\frac{\Gamma \vdash M : A \quad \Gamma, u : A \vdash N : C}{\Gamma \vdash (\mathbf{let}\ u : A = M\ \ \mathbf{in}\ \ N) : C}\ \mathsf{cut}$$

The let form here is necessary in the conclusion because otherwise the conclusion would still depend on u. This form is much more pleasant from a programming perspective. It also means we can type every term (not just normal terms) if we annotate the let form with its type. Additionally, this is the only form where we need a type. In some ways, this is the essence of bidirectional type-checking: only redexes need to be annotated with a type. If all redexes are expressed as let forms, this means only let forms need to be annotated, and really only if the term we are assigning is only checkable. If it were synthesizing, as in **let** $u = R$ **in** N, we could synthesize the type A of R and proceed to check N under the antecedent $u : A$.

Adding both of these alternatives to the syntax (even though only one is really required), we obtain this syntax that allows us to express arbitrary proofs, not just those annotating verifications.

$$\begin{array}{lll}
\text{Checkable terms} & M, N & ::= \langle M, N\rangle \mid (\mathsf{fn}\ u \Rightarrow M) \mid R \\
& & \quad \mid \mathbf{inl}\, M \mid \mathbf{inr}\, N \mid (\mathbf{case}\ R\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow M \mid v \Rightarrow N) \\
& & \quad \mid \mathbf{abort}\, R \\
& & \quad \mid (\mathbf{let}\ u : A = M\ \mathbf{in}\ N) \\
\text{Synthesizing terms} & R & ::= \mathbf{fst}\, R \mid \mathbf{snd}\, R \mid u \mid R\, M \mid (M : A)
\end{array}$$