The examples of the previous lecture show that finding evidence for a logical proposition isn't always easy, as it requires semantical reasoning about the structure of the components involved, i.e. a deep understanding of how an input may be used in order to construct the desired output. In general there is little machine support for semantical reasoning (although in the propositional case an algorithm based on type-analysis could be developed), which means that evidence terms would have to be constructed entirely by hand.

There is a different approach to solving proof and programming problems which makes the task of finding a solution easier. Instead of constructing the evidence term for a formula all at once, we consider the formula as *proof task* that needs to be solved and decompose this task into smaller *subtasks* until each of these subtasks can be solved in a single step. We then compose the evidence terms for these subtasks into evidence terms for the respective larger tasks until we have constructed the evidence term for the original formula. This process is called *proof by refinement*. We will illustrate it by a simple example.

Example 4.1 (Proof by refinement)

Consider the formula $P \Rightarrow (Q \Rightarrow (P \wedge Q))$. In order to prove the implication, we assume P and now have to show $Q \Rightarrow (P \wedge Q)$. In the next decomposition step we make Q our second assumption and have to show $P \wedge Q$. To prove $P \wedge Q$ we have to prove P and we have to prove Q. In both cases we have P and Q as assumption, so the solution is immediate.

In principle, this series of refinements is sufficient to assure the validity of $P \Rightarrow (Q \Rightarrow (P \wedge Q))$. But it also tells us how to assemble evidence for this formula. The proof of P under the assumptions P and Q yields as evidence a term $p : [P]$, which we get from the assumption P. Likewise the proof of Q yields as evidence a term $q : [Q]$, which we get from the assumption Q. Since both tasks were the result of refining $P \wedge Q$, we can assemble the two evidence terms into the pair $(p, q) : [P \wedge Q]$. Since the assumptions P and Q, which gave us the two terms p and q, resulted from refining $Q \Rightarrow (P \wedge Q)$ and $P \Rightarrow (Q \Rightarrow (P \wedge Q))$, the evidence term for the former must be $\lambda q.\, (p, q) : [Q \Rightarrow (P \wedge Q)]$ and the evidence term for the original formula is $\lambda p.\, (\lambda q.\, (p, q)) : [P \Rightarrow (Q \Rightarrow (P \wedge Q))]$. □

At a first glance the refinement approach to justifying the validity of a logical formula seems more complicated than constructing evidence terms directly. But it can be described in terms of formal refinement rules, which in turn can be implemented on a computer. A user of such an implementation would guide the proof process by selecting appropriate rules while the machine would execute each refinement step and generate the corresponding subtasks. Once the refinement is complete, the evidence terms can be assembled automatically.

We have seen in the above example that refinement proofs operate not just on formulas but on formulas that need to be proven and associated assumptions. We use the notation $H \vdash C$, called a *sequent*, to express that a *conclusion* C can be shown under the assumption that all the *hypotheses* in H are valid. In a sequent $H \vdash C$ the conclusion C is a single formula and H is a list of formulas[1].

[1]Some formalizations of sequents use sets of formulas as assumptions, since in propositional and first-order logic the order and multiplicity of hypotheses doesn't matter. By using lists instead of sets it becomes easier to address a specific hypothesis in the cours of a refinement. Some logics also permit the use of sequents with multiple goal formulas. The task is then to prove at least one of the goals. In this situation, however, it is not always possible to construct evidence for the original formula to be proven. We will discuss this issue in one of the future lectures.

Refinement proofs for a given formula A start with the *initial goal* $\vdash A$, i.e. the task of proving A without additional assumptions, and apply *refinement rules* that transform a *goal* sequent into *subgoal* sequents. Refinement rules are usually written as *rule schemes* of the form

$$\begin{array}{l} H \vdash G \\ \quad H_1 \vdash G_1 \\ \quad\quad \vdots \\ \quad H_n \vdash G_n \end{array}$$

where the formulas in the H_i and G_i may contain placeholders for logical propositions. Rules have to be designed in a way such that the provability of the original goal follows from that of the subgoals. The provability of $A \wedge B$, for instance, follows from the provability of A and that of B (without any changes to the hypostheses). Thus the rule for the refinement of a conclusion $A \wedge B$ will be written as

$$\begin{array}{l} H \vdash A \wedge B \\ \quad H \vdash A \\ \quad H \vdash B \end{array}$$

Since it operates on the right side of a sequent, we give it the name `andR`. Applying a refinement rule means matching $H \vdash G$ against the goal sequent and generating the appropriate instantiations of the $H_i \vdash G_i$ as subgoal sequents. Thus applying `andR` to the sequent $\vdash (P \Rightarrow Q) \wedge (R \Rightarrow Q)$ will instantiate A by $P \Rightarrow Q$ and B by $R \Rightarrow Q$ and generate the two subgoals $\vdash P \Rightarrow Q$ and $\vdash R \Rightarrow Q$. To simplify the presentation of formal proofs we write rule applications in a schematic fashion.

$\vdash (P \Rightarrow Q) \wedge (R \Rightarrow Q)$ `by andR`
1. $\vdash P \Rightarrow Q$
2. $\vdash R \Rightarrow Q$

This means that the application of the rule `andR` to the sequent $\vdash (P \Rightarrow Q) \wedge (R \Rightarrow Q)$ has generated two (numbered) subgoals that still need to be proven. A goal can be *closed* by applying a proof rule that does not generate subgoals. If all generated proof branches are closed, the refinement proof is complete. We will give examples of complete proofs in our discussion of the refinement rules for the propositional calculus.

Refinement rules may also associated with the construction of evidence[2]. The fact that the provability of the original goal must follow from that of the subgoals also means that evidence for the original goal can be constructed from evidence for the subgoals. The evidence for a goal $H \vdash A \wedge B$, for instance, is a pair (a, b) where a is evidence for $H \vdash A$ and b evidence for $H \vdash B$. We indicate that by writing the rule `andR` as

$H \vdash A \wedge B$ `ev =` (a, b) `by andR`
$H \vdash A$ `ev =` a
$H \vdash B$ `ev =` b

In propositional refinement logic, most rules follow immediately from an intuitive understanding of the logical connectives. As in our discussion of evidence semantic we will investigate each connective separately.

[2]This is not necessary because a complete refinement proof provides as much assurance for the validity of a proposition as an evidence term would do. But evidence construction provides a link between refinement proofs and evidence semantics. It also has an interesting side-effect on the mplementation of proof systems: it allows the extraction of programs from proofs of formulas that state the existence of certain algorithms and thus the construction of verified programs through theorem proving.

- Refinement rules for propositional variables

  A propositional variable A cannot be decomposed into smaller components because it is a placeholder for an unknown proposition. But if A is also one of the assumptions in the hypotheses, we can obviously prove A. We write this schematically as

  $$H, A, H' \vdash A$$

  to indicate that the formula A occurs somewhere in the list of assumptions. H and H' are lists of formulas. H is empty if A is the first hypothesis, H' is empty if A is the last one. We call the rule that proves A under these assumption `axiom`. It does not generate any subgoals, because the sequent is self-evident.

  What is the evidence constructed by this rule?

  Essentially the evidence needs to show that the hypothesis A was used to prove the conclusion A. To indicate that we need to associate hypotheses with labels.[3]

  So if we label the assumption A by a, then the label a is sufficient evidence for the validity of the conclusion A in the sequent $H, A, H' \vdash A$. If we integrate that information into the rule, using programming notation then the `axiom` rule appears as follows.

  $$H, a{:}A, H' \vdash A \quad \texttt{ev} = a \quad \texttt{by axiom}$$

- Refinement rules for $A \Rightarrow B$

  To prove an implication $\vdash A \Rightarrow B$ it is sufficient to assume A and then prove B. This means that A will be added to the hypotheses of the sequent and B becomes the new conclusion.

  $$\begin{array}{l} H \vdash A \Rightarrow B \\ \quad H, A \vdash B \end{array}$$

  We call the rule `impliesR`, as it decomposes an "implies" on the right side of the sequent.

  To describe the evidence constructed by this rule we have to associate the newly generated hypothesis A with a label a and assume that we have some evidence b for the the subgoal sequent $H, a{:}A \vdash B$. This means that we must have found a generic way to turn an evidence a for A into the evidence b for B. In other words, there is a function that on input a computes b and this function must be the evidence for $A \Rightarrow B$. Using the λ-notation for evidence terms introduced previously, we can now write the rule as

  $$\begin{array}{lll} H \vdash A \Rightarrow B & \texttt{ev} = \lambda a.b & \texttt{by impliesR} \\ \quad H, a{:}A \vdash B & \texttt{ev} = b & \end{array}$$

  Let us look at a few concrete examples

  - $P \Rightarrow P$: The refinement proof for this formula is straightforward. We decompose the implication and then apply the `axiom` rule to the generated subgoal:

    $$\begin{array}{lll} \vdash P \Rightarrow P & & \texttt{by impliesR} \\ \texttt{1.} \quad P \vdash P & & \texttt{by axiom} \end{array}$$

    If we include the generated evidence terms, then the hypothesis P in the subgoal will receive a label p and the evidence created by applying the `axiom` rule is p. Using the evidence constructor associated with `impliesR` we get the evidence $\lambda p.\, p$, which is

[3] Labels are easier to track than, for instance, the position of the formula A in a list of hypotheses.

exactly what we expected. If we integrate evidence construction into our presentation of the proof we get:[4]

$\vdash\ P \Rightarrow P$ ev = $\lambda p.\, p$ by impliesR
1. $p{:}P \vdash P$ ev = p by axiom

One should keep in mind that while the proof is constructed top-down the evidence will be constructed bottom-up *after* the refinement proof has been completed.

- $P \Rightarrow (Q \Rightarrow P)$: The refinement proof for this formula is straightforward.

$\vdash\ P \Rightarrow (Q \Rightarrow P)$ ev = $\lambda p.\, (\lambda q.\, p)$ by impliesR
1. $p{:}P \vdash Q \Rightarrow P$ ev = $\lambda q.\, p$ by impliesR
1.1. $p{:}P, q{:}Q \vdash P$ ev = p by axiom

In the schematic presentation, the index 1.1. indicates that the last subgoal is the first subgoal of subgoal number 1.

In addition to refining the conclusion of a sequent we also have to be able to decompose assumptions. A refinement proof of the formula $P \Rightarrow ((P \Rightarrow Q) \Rightarrow Q)$, for instance, will eventually generate the subgoal $P, P \Rightarrow Q \vdash Q$. The proof will only be able to proceed if we decompose the implication on the left side of the sequent. This will generate two goals. The first requires us to show that we are able to provide evidence for the left side *of the implication*. Once we have this evidence we may use the right side of the implication to prove the conclusion. The corresponding refinement rule is:

$H, A \Rightarrow B, H' \vdash C$ by impliesL
$H, A \Rightarrow B, H' \vdash A$
$H, B, H' \vdash C$

Note that we have to preserve the hypothesis $A \Rightarrow B$ because we may have to use it again.

For the construction of evidence let us assume that f is a label for $A \Rightarrow B$ in the main goal, that a is the evidence for A in the first subgoal, and that in the last subgoal b is a label for B and c the evidence for C. The latter means that there is a function that computes evidence c for C from arbitrary evidences b for B. Since $f(a)$ is a specific evidence for B we can apply this function, i.e. $\lambda b.c$, to $f(a)$ and get the desired evidence for C in the main goal. If we integrate evidence construction into the rule we get

$H, f{:}A \Rightarrow B, H' \vdash C$ ev = $(\lambda b.c)(f(a))$ by impliesL
$H, f{:}A \Rightarrow B, H' \vdash A$ ev = a
$H, b{:}B, H' \vdash C$ ev = c

Taking the reduction rules of the lambda-calculus into consideration the evidence could also be presented in its reduced form $c[f(a)/b]$, which often leads to shorter evidence terms.

Using this rule we may now complete the proof of $P \Rightarrow ((P \Rightarrow Q) \Rightarrow Q)$

$\vdash\ P \Rightarrow ((P \Rightarrow Q) \Rightarrow Q)$ ev = $\lambda p.\, (\lambda h.\, (\lambda q.q)(h(p)))$ by impliesR
1. $p{:}P \vdash (P \Rightarrow Q) \Rightarrow Q$ ev = $\lambda h.\, (\lambda q.q)(h(p))$ by impliesR
1.1. $p{:}P, h{:}(P \Rightarrow Q) \vdash Q$ ev = $(\lambda q.q)(h(p))$ by impliesL
1.1.1. $p{:}P, h{:}(P \Rightarrow Q) \vdash P$ ev = p by axiom
1.1.2. $p{:}P, q{:}Q \vdash Q$ ev = q by axiom

Note that the evidence $\lambda p.\, (\lambda h.\, (\lambda q.q)(h(p)))$ is not identical to the evidence $\lambda p.\, (\lambda h.\, h(p))$ that we constructed when we investigated the formula in the previous section. However, if

[4] In an computerized proof system the impliesR rule has to provide the label for P and the axiom rule has to give the hypothesis number of P. Since this is an implementation detail we will ignore rule parameters in these notes.

we use the reduction rules of the lambda calculus to compute the result of the application of $\lambda q.q$ to $h(p)$ (i.e. $h(p)$) then the two terms become identical.

- Refinement rules for $A \wedge B$

  We already discussed the refinement of conjunctions on the right side of a sequent:

  $H \vdash A \wedge B$ ev = (a, b) by andR
  $H \vdash A$ ev = a
  $H \vdash B$ ev = b

  To decompose a conjunction $A \wedge B$ in the hypotheses we only need to split it into two separate assumptions A and B, leaving the rest unchanged.

  For the construction of evidence let us assume that x is a label for $A \wedge B$, a one for A, b one for B, and c the evidence for the subgoal sequent $H, A, B, H' \vdash C$. Then by construction a must be the same as x_1 and b the same as x_2 and the evidence for the main goal can be constructed by providing x_1 and x_2 as inputs to the function $\lambda a.\,(\lambda b.c)$.

  $H, x{:}A \wedge B, H' \vdash C$ ev = $((\lambda a.\,(\lambda b.c))(x_1))(x_2)$ by andL
  $H, a{:}A, b{:}B, H' \vdash C$ ev = c

  Again, the evidence could also be presented in a reduced form, which is $c[x_1, x_2/a, b]$. In the following examples we will make use of this form if it makes the extracted evidence terms simpler.

  - $P \Rightarrow (Q \Rightarrow (P \wedge Q))$: The refinement proof is fairly straightforward.

    $\vdash P \Rightarrow (Q \Rightarrow (P \wedge Q))$ ev = $\lambda p.\,(\lambda q.\,(p, q))$ by impliesR
    1. $p{:}P \vdash Q \Rightarrow (P \wedge Q)$ ev = $\lambda q.\,(p, q)$ by impliesR
    1.1. $p{:}P, q{:}Q \vdash P \wedge Q$ ev = (p, q) by andR
    1.1.1. $p{:}P, q{:}Q \vdash P$ ev = p by axiom
    1.1.2. $p{:}P, q{:}Q \vdash P$ ev = q by axiom

    The constructed evidence is the same as we had before.

  - $(P \wedge Q) \Rightarrow P$: The refinement proof is

    $\vdash (P \wedge Q) \Rightarrow P$ ev = $\lambda x.\,x_1$ by impliesR
    1. $x{:}(P \wedge Q) \vdash P$ ev = x_1 by andL
    1.1. $p{:}P, q{:}Q \vdash P$ ev = p by axiom

    Note that the evidence term $p[x_1, x_2/p, q]$ generated by andL is x_1 because this is the result of replacing every occurrence of p by x_1 and of q by x_2 in the term p.

- Refinement rules for $A \vee B$

  Decomposing a disjunction $A \vee B$ in the conclusion of a sequent requires us to make a choice. In order to prove $A \vee B$ we may either prove A or we may prove B. As a result there will be two rules, one that directs the proof towards proving the left disjunct and another that directs it towards proving the right one. COnsequently, the first rule will wrap the evidence for the subgoal with inl and the other one with inr:

  $H \vdash A \vee B$ ev = inl(a) by orR1
  $H \vdash A$ ev = a
  $H \vdash A \vee B$ ev = inr(b) by orR2
  $H \vdash B$ ev = b

Refine a disjunction $A \vee B$ in the hypotheses of a proof will cause the proof to branch. If we want to prove a conclusion C under the assumption $A \vee B$, then it is sufficient to show how to prove C assuming A and how to prove C if we assume B. Since we have evidence for either A or for B these two conditions will tell us how to prove C in either case.

$H, A \vee B, H' \vdash C$ `by orL`
$H, A, H' \vdash C$
$H, B, H' \vdash C$

For the construction of evidence let us assume that that x is a label for $A \vee B$ in the main goal, that in the first subgoal a is a label for A and c_1 the evidence for C, and that in the second subgoal b is a label for B and c_2 the evidence for C. Then by construction x is either $\texttt{inl}(a)$ or $\texttt{inr}(b)$ and therefore the evidence for the main goal is constructed by the case analysis term for disjunctions, i.e. $\texttt{case}\ x\ \texttt{of}\ \texttt{inl}(a) \to c_1 \mid \texttt{inr}(b) \to c_2$.

$H, x{:}A \vee B, H' \vdash C$ $\texttt{ev} = \texttt{case}\ x\ \texttt{of}\ \texttt{inl}(a) \to c_1 \mid \texttt{inr}(b) \to c_2$ `by orL`
$H, a{:}A, H' \vdash C$ $\texttt{ev} = c_1$
$H, b{:}B, H' \vdash C$ $\texttt{ev} = c_2$

Again let us look at concrete examples

- $P \Rightarrow (P \vee Q)$: The refinement proof is

$\vdash P \Rightarrow (P \vee Q)$ $\texttt{ev} = \lambda p.\, \texttt{inl}(p)$ `by impliesR`
`1.` $p{:}P \vdash P \vee Q$ $\texttt{ev} = \texttt{inl}(p)$ `by orR1`
`1.1.` $p{:}P \vdash P$ $\texttt{ev} = p$ `by axiom`

- $(P \vee Q) \Rightarrow (Q \vee P)$: The refinement proof is

$\vdash (P \vee Q) \Rightarrow (Q \vee P)$ $\texttt{ev} = \lambda x.\, (\texttt{case}\ x\ \texttt{of}\ \texttt{inl}(p) \to \texttt{inr}(p) \mid \texttt{inr}(q) \to \texttt{inl}(q))$ `by impliesR`
`1.` $x{:}(P \vee Q) \vdash Q \vee P$ $\texttt{ev} = \texttt{case}\ x\ \texttt{of}\ \texttt{inl}(p) \to \texttt{inr}(p) \mid \texttt{inr}(q) \to \texttt{inl}(q)$ `by orL`
`1.1.` $p{:}P \vdash Q \vee P$ $\texttt{ev} = \texttt{inr}(p)$ `by orR2`
`1.1.1.` $p{:}P \vdash P$ $\texttt{ev} = p$ `by axiom`
`1.2.` $p{:}P \vdash Q \vee P$ $\texttt{ev} = \texttt{inl}(q)$ `by orR1`
`1.2.1.` $q{:}Q \vdash Q$ $\texttt{ev} = q$ `by axiom`

- Refinement rules for $\neg A$

Since we already explained the relation between negation and implication we should expect the rules to be similar. To prove a negation $\vdash \neg A$ one usually assumes A and then prove a contradiction, which we denoted by the term f. For the construction of the associated evidence we assume that a is a label for A in the subgoal and b is the evidence for that subgoal (if the hypotheses are contradictory we will be able to construct that evidence). The, using the same argument as before, $\lambda a.b$ will be the evidence for the main goal.

$H \vdash \neg A$ $\texttt{ev} = \lambda a.b$ `by notR`
$H, a{:}A \vdash \mathsf{f}$ $\texttt{ev} = b$

The rule for decomposing a hypothesis of the form $\neg A$ is a simplified version of `impliesL`. The only way to make use of the negation is trying to prove A. If we can do that we have a contradiction (we have both A and $\neg A$), which means that there is nothing else to prove.

For the construction of evidence assume that f is a label for $\neg A$ in the main goal and that a is the evidence for A in the subgoal. This means that applying f to a will result in an

left		right	
impliesL	$H, f{:}A \Rightarrow B, H' \vdash C$ ev = $c[f(a)/b]$ $H, f{:}A \Rightarrow B, H' \vdash A$ ev = a $H, b{:}B, H' \vdash C$ ev = c	$H \vdash A \Rightarrow B$ ev = $\lambda a.b$ $H, a{:}A \vdash B$ ev = b	impliesR
andL	$H, x{:}A \wedge B, H' \vdash C$ ev = $c[x_1, x_2/a, b]$ $H, a{:}A, b{:}B, H' \vdash C$ ev = c	$H \vdash A \wedge B$ ev = (a, b) $H \vdash A$ ev = a $H \vdash B$ ev = b	andR
orL	$H, x{:}A \vee B, H' \vdash C$ ev = case x of inl$(a) \to c_1$ | inr$(b) \to c_2$ $H, a{:}A, H' \vdash C$ ev = c_1 $H, b{:}B, H' \vdash C$ ev = c_2	$H \vdash A \vee B$ ev = inl(a) $H \vdash A$ ev = a $H \vdash A \vee B$ ev = inr(b) $H \vdash B$ ev = b	orR1 orR2
notL	$H, f{:}\neg A, H' \vdash C$ ev = any$(f(a))$ $H, f{:}\neg A, H' \vdash A$ ev = a	$H \vdash \neg A$ ev = $\lambda a.b$ $H, a{:}A \vdash$ f ev = b	notR
		$H, a{:}A, H' \vdash A$ ev = a	axiom

Table 1: Refinement rules of the propositional refinement calculus

element of the empty type, which by definition cannot exist. We can utilize this information to claim that anything is valid now[5] and that anything could serve as evidence for the main goal. We could use the term axiom. But we want to indicate that $f(a)$ is the reason for the contradictory situation and will therefore make it a part of the evidence that we generate. For this purpose we introduce a new function any which, when applied to an element of the type $\{\}$, will serve as evidence for anything. With this special term we get the following rule.

$H, f{:}\neg A, H' \vdash C$ ev = any$(f(a))$ by notL
$H, f{:}\neg A, H' \vdash A$ ev = a

Again we will consider two examples

- $P \Rightarrow \neg\neg P$: overall evidence is $\lambda p.\,(\lambda h.\,h(p))$.

$\vdash P \Rightarrow \neg\neg P$ ev = $\lambda p.\,(\lambda h.\,\text{any}(h(p)))$ by impliesR
1. $p{:}P \vdash \neg\neg P$ ev = $\lambda h.\,\text{any}(h(p))$ by notR
1.1. $p{:}P, h{:}(\neg P) \vdash$ f ev = any$(h(p))$ by notL
1.1.1. $p{:}P, h{:}(\neg P) \vdash P$ ev = p by axiom

- $\neg(P \vee Q) \Rightarrow \neg P$: overall evidence is $\lambda h.\,(\lambda p.\,h(\text{inl}(p)))$.

$\vdash \neg(P \vee Q) \Rightarrow \neg P$ ev = $\lambda h.\,(\lambda p.\text{any}(h(\text{inl}(p)))$ by impliesR
1. $h{:}\neg(P \vee Q) \vdash \neg P$ ev = $\lambda p.\text{any}(h(\text{inl}(p))$ by notR
1.1. $h{:}\neg(P \vee Q), p{:}P \vdash$ f ev = any$(h(\text{inl}(p))$ by notL
1.1.1. $h{:}\neg(P \vee Q), p{:}P \vdash P \vee Q$ ev = inl(p) by orR1
1.1.1.1. $h{:}\neg(P \vee Q), p{:}P \vdash P$ ev = p by axiom

Note that in both examples the evidence does not show that the rule notL has been applied to a goal with the conclusion f. In that special situation the element of type $\{\}$ could have been used directly as evidence for f instead of being wrapped by any[6].

[5]The principle *ex falso quodlibet* (from a false proposition, anything follows) has been disputed by philosophers for centuries. In mathematics, it makes sense to adopt this principle, since in a contradictory theory we may be able to prove propositions like 0=1 or *all numbers are equal*, which makes as much sense as accepting that everything is true.

[6]Had we *defined* negation $\neg A$ as abbreviation for $A \Rightarrow$ f, dropped the negation rules, and created a rule falseL for f, we would have had the option of proving H, f $\vdash$ f bei either axiom, resulting in the evidence term $\lambda x.\,x$, or falseL, resulting in any.

Table 1 summarizes all the rules of the propositional refinement calculus. As we can see, there is a correspondence between right rules and the construction of evidence for a given connective, and between left rules and terms for decomposing evidence for that connective.

We conclude our investigation of refinement proofs by giving a precise definition of sequents, proofs, and the extraction mechanism. These definitions will help in the discussion of issues like correctness (*is every proven formula semantically valid?*) and completeness (*can every semantically valid formula be proved*) of the refinement calculus and also serve as a foundation for implementations.

Definition 4.2 (Sequents)

(1) *A declaration has the form $x:A$ where x is a (term) variable and A is a formula. It declares x to be evidence for A.*

(2) *A hypotheses list H is a list of declarations, written as $x_1{:}A_1, ldots, x_n{:}A_n$, where all the x_i are distinct.*

(3) *A sequent (or goal) has the form $H \vdash C$ where H is a hypotheses list and C (the conclusion) is a formula.*

(4) *An evidence term for a sequent $S = x_1{:}A_1, ldots, x_n{:}A_n \vdash C$ is a term whose variables are declared in the hypotheses and which is evidence for the conclusion C whenever its variables x_i are instantiated by evidence for the A_i.*

A sequent $S = x_1{:}A_1, ldots, x_n{:}A_n \vdash C$ is usually intended to express that *the conclusion C is valid under the assumption that all the formulas in the hypothesis list are valid*. If this is actually the case then the sequent is valid itself. Using evidence we can give a precise definition of the validity of sequents.

Definition 4.3 (Validity of sequents)

(1) *An evidence term for a sequent $S = x_1{:}A_1, \ldots, x_n{:}A_n \vdash C$ is a term whose variables are declared in the hypotheses and which is evidence for the conclusion C whenever its variables x_i are instantiated by evidence for the A_i.*

(2) *A sequent is valid if there is an evidence term for it.*

To prove the validity of a sequent, refinement rules *decompose* sequents into subgoal sequents and provide a *validation* mechanism that constructs evidence terms for the main goal from evidence terms for the subgoals.

Definition 4.4 (Refinement rules)

(1) *A decomposition (or refinement) is a mapping from a sequent (the main goal) into a into a list of sequents, called the subgoals.*

(2) *A validation ist a mapping from a list of sequents and terms into a term*[7].

(3) *A rule is a pair (dec, val) where dec is a decomposition and val a validation.*

[7]To construct an evidence term for a sequent a validation needs both the evidence terms of the subgoal sequents and the variables declared in them.

The rule schemes that we have used so far describe both the decomposition and the validation.

$$\begin{array}{ll} H \vdash G & \texttt{ev} = g \\ \quad H_1 \vdash G_1 & \texttt{ev} = g_1 \\ \quad \vdots & \\ \quad H_n \vdash G_n & \texttt{ev} = g_n \end{array}$$

The corresponding decomposition is a function that matches $H \vdash G$ against the goal sequent, instantiates the $H_i \vdash G_i$ accordingly and returns them as subgoals. The validation takes the generated subgoals, matches the terms $g_1, .., g_n$ against the evidence constructed for these subgoals and computes the evidence term by instantiating the term g.

Obviously, a proof calculus should only use *correct* inference rules where the correctness of a rule $r = (dec, val)$ can be explained as follows.

> Let $S = H \vdash A$ be an arbitrary sequent, $S_i = H_i \vdash A_i$ be the subgoal sequents created by applying dec to S, and a_i be evidence terms for the S_i. Then the term a resulting from applying val to the list of all pairs S_i, a_i is an evidence term for S.

Proving that the inference rules of the propositional calculus presented here are in fact correct is a straightforward but somewhat tedous task. We leave this as an exercise for the reader.

A proof of a formula C is generated by applying inference rules to the sequent $\vdash C$, to the generated subgoals, their subgoals, etc. This mechanism eventually generates a tree whose nodes are marked with sequents and rules such that the successors of a node are marked with the subgoals resulting from applying the (decomposition of the) rule to the sequent. An incomplete tree If the tree contains nodes marked with a sequent but not yet with a rule. This means that a proof can only be complete, if its leafs are marked with rules that do not generate subgoals.

Definition 4.5 (Proofs)

(1) *Proofs are recursively defined as follows*

- *Any sequent $S = H \vdash C$ is an incomplete proof with root S.*
- *Let $S = H \vdash A$ be a sequent, $r = (dec, val)$ a correct inference rule, $S_1, \ldots, S_n$ be the subgoal sequents created by applying dec to S, and π_i be proofs with roots S_i. Then the tuple $(S, r, [\pi_1, \ldots, \pi_n])$ is a proof with root S.*

(2) *A proof is complete if it does not contain any incomplete subproofs.*

(3) *A theorem is a complete proof whose root is an initial sequent of the form $\vdash C$.*

Since all the inference rules used in the construction of a proof are required to be correct, all the sequents in the proof must be valid. The evidence terms for the leaf sequents are constructed from scratch by the validations of their inference rules. The evidence terms of the remaining nodes are constructed iteratively from their successor's evidence terms, again by applying the corresponding validations. Since the root of the proof tree is an initial sequent $\vdash C$, the evidence term of that sequent is also the evidence for the formula C. Since evidence terms are generated only after a proof has been completed, they are often viewed as *extracted* from the proof.

Definition 4.6 (Extract terms)

The extract term of a complete proof $\pi = (S, (dec, val), [\pi_1, \ldots, \pi_n])$ is recursively defined as $ext(\pi) = val([(S_1, ext(\pi_1)), \ldots, (S_n, ext(\pi_n))])$, where the S_i are the roots of the subproofs π_i.

Note that this recursive definition does not require a base case. Since the leaf nodes of a complete tree do not have any subgoals ($\pi = (S, (dec, val), [])$) the validation is applied to an empty list.

Exercises

As an exercise the following problems should be investigated in groups. For each of the formulas below the group should find a refinement proof and construct the evidence from the proof. In cases where no proof can be found, try to explain why the proof has to get stuck.

- $(P \wedge Q) \Rightarrow (P \vee Q)$ (give two different proofs):
- $(P \Rightarrow Q) \Rightarrow ((R \Rightarrow Q) \Rightarrow (R \Rightarrow P))$:
- $(P \vee Q) \Rightarrow (P \vee Q)$:
- $(P \vee Q) \Rightarrow ((P \Rightarrow R) \Rightarrow ((Q \Rightarrow R) \Rightarrow R))$:
- $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$:
- $(P \Rightarrow Q) \Rightarrow (\neg P \Rightarrow \neg Q)$:
- $\neg(P \vee Q) \Rightarrow (\neg P \wedge \neg Q)$:
- $(\neg P \wedge \neg Q) \Rightarrow \neg(P \vee Q)$:
- $\neg(\neg P \wedge \neg Q) \Rightarrow (P \vee Q)$:
- $\neg\neg P \Rightarrow P$:
- $\neg P \vee P$: