Chapter 4

Structured Proof

The natural deduction rules of JZF give us a rigorous and precise set of logical reasoning principles. When combined with the axioms of set theory, this gives you a rather stringent standard of proof for set theoretic propositions. Using these rules, you can construct proofs of hypothetical judgments as derivation trees. The system is quite small: one could even imagine encoding JZF proofs as a data structure in some programming language, along with functions that traverse them to ensure that they represent correct proofs. Such proof-checking machinery lies at the heart of mechanized proof assistants. Here we focus on paper proofs, but we want to know that at least in principle we can write precise enough proofs that we could teach a computer to check them.

For all of their rigour, though, the ergonomics of natural deduction leave much to be desired. For starters, even a proof of a small hypothetical judgment quickly balloons into a large tree that is difficult to typeset on a standard piece of paper. Even if we omit proofs of some premises (i.e. present only a proof sketch rather than a full proof), things get out of hand and are hard for humans to write and read. See Fig. 4.1 for an example

Furthermore, the rules of JZF are quite spartan. It's quite amazing that a single page of rules technically suffices to convey our conception of logical-deduction-as-information-processing, breaking down assumptions into pieces that can be used to subsequently verify propositions. This intuition is quite powerful. But JZF is more low-level than we would want to work with on a day-to-day basis. It amounts to an "assembly language" for proof. In computer programming, assembly language is also small, spartan and capable *in principle* of expressing all our desired programs. But for most programming we want to work in a higher-level language. The same applies to proof: we want higher-level abstractions for reasoning, including some that convey styles of reasoning may have already learned and taken for granted from previous exposure to logic or mathematics, but which are not directly embodied in the rules of JZF. In essence, we want a high-level proof language that "compiles" to JZF, but enables us to reason precisely and rigourously.

Although we want our proofs to be rigorous, we would like some control over how precise we must be when making arguments. The JZF rules are extremely precise, but that level of precision can obscure the essence of a proof, especially when communicating among humans. For instance, a vanilla JZF proof is littered with proofs of the `prop` and `set` judgments, many of which add no insightful content to a proof, especially when no definite descriptions are involved, or when the descriptions involved arise directly from the axioms of set theory. Sometimes these proofs are critical to conveying an argument, because it depends on the unique existence of some set: in those cases, an explicit proof can convey content. And when first learning the system, it's important to at least know where such proofs are technically required, if only so that one can convince oneself that it's not a problem and move on. More generally, as we get more comfortable with rigorous proof, and as we seek to communicate the core insightful content of proofs to other experts, it can be beneficial to elide details of proofs that detract from the essence of an argument, but that we have high confidence that we can fill in if asked to. Many computer proof assistants can automatically fill in proof components, though the possibility of automation does not necessarily determine which parts of a proof should be omitted. A proof is about communicating with a person, and their understanding may depend on some information that can indeed be automated.

Some philosophers, e.g. Avigad [2020], argue that an ideal rigorous proof is one that could be turned into a formal proof (e.g. a full JZF natural deduction) merely by filling in some missing details, including assumptions that are easy to establish so not proven, or even assumptions that are obvious to the reader so need not even be stated. One goal of this work is to help you get to the point where you can produce such proofs (which skip steps that you know are technically requisite, but are highly confident that they are achievable and that the audience for your proof will agree or be equipped to fill in the dots).

To address these needs (tractable, rigorous proof at varying levels of precision), we now introduce an abstraction layer over JZF: a syntax for proofs that lets us fit proofs on paper, appeal to higher-level proof techniques that we can rigorously explain, and elide details that could be filled in later. The style we introduce is called *structured proof*. It was pioneered by Turing Award winner Leslie Lamport, who sought a notation for writing rigorous proofs about computer algorithms as a means of catching errors in his proofs [Lamport, 1995, 2012].

Our approach to structured proof uses the overall structure that Lamport laid out, but makes some changes to accommodate JZF, especially our separation between judgments and propositions. As you'll see, we can interpret structured proofs as a language for describing the construction, manipulation, and transformation of natural deduction proofs of hypothetical judgments. But ultimately, structured proofs are a high-level language for writing and reading proofs directly, even though each step can be explained and justified in terms of how to "compile" it to a natural deduction proof.

Our metatheorems, which describe how to transform one JZF proof of one hypothetical judgment into a related proof of another judgment, will be treated as proof rules in structured proofs. This technique, which corresponds to *tactics* in mechanized proof assistants, will help us make proofs shorter and more focused.

Structured proofs will also allow us to describe the construction of the same JZF proof multiple different ways, depending on the point of emphasis. In essence, a structure proof describes how one might go about building a natural deduction proof, and the order in which that construction happens can have conceptual importance to the consumer of the proof (as well as the person writing the proof).

4.1 A Simple Structured Proof by Example

To demonstrate some of the features of structured proofs and how they relate to natural deduction proofs, we first present a natural deduction proof of a hypothetical judgment that exposes some of the interesting features of both systems.

Fig. 4.1 presents a proof of the judgment $A \;\mathsf{prop} \vdash \forall Q.\, (\exists S.\, S \in Q \wedge A) \Rightarrow (\exists S.\, S \in Q) \wedge A \;\mathsf{verif}$. The proof is partial: proofs of prop judgments have been omitted. Because the proposition A is hypothesized to be well-formed in the absence of assumptions about any well-formed sets, it follows that A does not depend on the universally quantified set S. Stated more intuitively, A cannot refer to S. This judgment (and its proof) demonstrates that under that assumption, we can hoist the verification of A out of the existential quantifier. Our substitution principle Prop 152 empowers us to prove any analogue of this judgment that replaces A with a proposition that is well-formed under no assumptions.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\Gamma_1 \vdash \exists S.\, S \in Q \land A\ \mathsf{use}}\,(\mathrm{hypU}^x)
\quad
\dfrac{
\dfrac{
\begin{array}{c}\vdots\\ \Gamma_2, S_2\ \mathsf{set} \vdash S_2 \in Q\ \mathsf{prop}\end{array}
\quad
\dfrac{}{\Gamma_2 \vdash S_1\ \mathsf{set}}\,(\mathrm{hypS}^{S_1})
\quad
\dfrac{
\dfrac{\dfrac{}{\Gamma_2 \vdash S_1 \in Q \land A\ \mathsf{use}}\,(\mathrm{hypU}^y)}{\Gamma_2 \vdash S_1 \in Q\ \mathsf{use}}\,(\land\mathrm{E1})
\quad
\begin{array}{c}\vdots\\ \Gamma_2 \vdash S_1 \in Q\ \mathsf{prop}\end{array}
}{\Gamma_2 \vdash S_1 \in Q\ \mathsf{verif}}\,(\mathrm{atomic})
}{\Gamma_2 \vdash \exists S.\, S \in Q\ \mathsf{verif}}\,(\exists\mathrm{I})
\quad
\dfrac{
\dfrac{\dfrac{}{\Gamma_2 \vdash S_1 \in Q \land A\ \mathsf{use}}\,(\mathrm{hypU}^y)}{\Gamma_2 \vdash A\ \mathsf{use}}\,(\land\mathrm{E2}
\quad
\dfrac{}{\Gamma_2 \vdash A\ \mathsf{prop}}\,(\mathrm{hypP}^A)
}{\Gamma_2 \vdash A\ \mathsf{verif}}\,(\mathrm{atomic})
}{(\Gamma_2 := \Gamma_1, S_1\ \mathsf{set}, y : S_1 \in Q \land A\ \mathsf{use}) \vdash (\exists S.\, S \in Q) \land A\ \mathsf{verif}}\,(\land\mathrm{I})
\quad
\begin{array}{c}\vdots\\ \Gamma_1 \vdash (\exists S.\, S \in Q) \land A\ \mathsf{prop}\end{array}
}{(\Gamma_1 := \Gamma_0, x : \exists S.\, S \in Q \land A\ \mathsf{use}) \vdash (\exists S.\, S \in Q) \land A\ \mathsf{verif}}\,(\exists\mathrm{E}^{S,y})
\quad
\begin{array}{c}\vdots\\ \Gamma_0 \vdash \exists S.\, S \in Q \land A\ \mathsf{prop}\end{array}
}{(\Gamma_0 := A\ \mathsf{prop}, Q\ \mathsf{set}) \vdash (\exists S.\, S \in Q \land A) \Rightarrow (\exists S.\, S \in Q) \land A\ \mathsf{verif}}\,(\Rightarrow\mathrm{I}^x)
}{A\ \mathsf{prop} \vdash \forall Q.\, (\exists S.\, S \in Q \land A) \Rightarrow (\exists S.\, S \in Q) \land A\ \mathsf{verif}}\,(\forall\mathrm{I}^Q)
$$

Figure 4.1: Natural Deduction Proof of $A\ \mathsf{prop} \vdash \forall Q.\, (\exists S.\, S \in Q \land A) \Rightarrow (\exists S.\, S \in Q) \land A\ \mathsf{verif}$

Fig. 4.2 presents a structured proof of the same hypothetical judgment. This proof amounts to a recipe for constructing the natural deduction proof from Fig. 4.1.[1]

It's worth highlighting some of the features of the structured proof compared to it's natural deduction counterpart. The natural deduction proof has a tree shape, and gets wide enough that we must flip the proof to landscape orientation and shrink it's font, even without including the proofs of prop judgments. In contrast, the structured proof has a nested indented structure and fits in portrait orientation without decreasing its font.

Next, both proof styles exhibit hierarchical structure, but in different ways. The natural deduction proof can be viewed as a tree of proofs. Each subtree of the overall proof is in and of itself a proof of a constituent hypothetical judgment. These proofs are composed using rules to build bigger proofs. In the structured proof, each numbered step counts as a distinct proof of a hypothetical judgment. A structured proof may consist of subproofs, which are numbered using a hierarchical numbering scheme (e.g. 4.1.2 is a subproof of 4.1), and using indentation to further indicate structure. This style is analogous to how one writes an outline in preparation for writing an essay.

Next, each proof style uses a different approach to enumerating assumptions and referring to them. A natural deduction proof explicitly enumerates every assumption available to each hypothetical judgment proven along the way to the proof of the overall hypothetical judgment. Assumptions are referred to by creating a standalone proof using one of the "hyp" rules, appealing either directly to the assumed entity, in the case of hypS and hypP, or to an alphabetic label in the case of hypU. For example, the proof labelled hypU^y appeals to the alphabetical y label to justify the conclusion $S_1 \in Q \wedge A\,\mathsf{use}$ of its hypothetical judgment. In contrast, structured proofs use *block structure*—whereby indentation and hierarchical numbering indicate nested "blocks" of proofs and subproofs—to implicitly determine which assumptions are available to each subproof. Also, for conciseness sake, structured proofs do not require an explicit appeal to a "hyp" rule to prove an assumption. Assumptions are uniformly referenced by appealing to the step number where an assumption was introduced, as well as to the numeric index given to the assumption, when more than one assumption is introduced at that step. For example, step 4.2.1 appeals to assumption 4.1:1, meaning assumption 1 introduced at step 4.1, for the assumption $S_1 \in Q \wedge A\,\mathsf{use}$, rather than creating an independent subproof of the judgment that appeals to that assumption (which is legal but unnecessary). A top-level statement of a proposition may have one or more assumptions, in which case the step index is empty: for example :5 refers to the fifth assumption of the overall judgment being proven. Assumptions in both structured proofs and natural deduction are *not* ordered, though they are *labelled* to give distinct identity to multiple assumptions of license to use the same proposition (as would arise when proving, e.g., $A \Rightarrow (A \Rightarrow A)\,\mathsf{verif}$). One could choose, for clarity of exposition, to explicitly state an assumption as a proof, possibly because it is useful for the reader to see it written down close by and unambiguously. Allowing "proof by reference" can help with reading proofs, rather than having to shuffle back and forth to see what proof step is being referenced. This is a matter of taste.

Next, the tree form of natural deduction proofs determines the entirety of the dependency order for proofs and sub-proofs. There is sense in which this presentation describes the "order" in which the proof tree was built. You can read it "bottom-up", as though the proof were written from conclusion to root; and you can read it "top-down", as though each complete sub-proof was built before considering what conclusions could be drawn from it. In contrast, the steps and nesting mechanisms of structured proofs provide much control over how the presentation of the proof is ordered and broken into chunks. Furthermore, the (SUFFICES) mechanism enables the "bottom-up" description of a "proof with holes", which can facilitate a more intuitive explanation of the essence of a proof. A single natural deduction proof can be described by many different structured proofs, each of which describes the construction of the same tree, but possibly in different orders.

Some useful facets of structured proofs are not demonstrated in Fig. 4.2 but are explained below. One may add "proof sketches" as in-line commentary to a structured proof. This lets you intersperse prose into a proof as guidance, much like comments in a computer program. On may also use LET to introduce notational definitions for propositions and sets. This corresponds to syntactic abstractions like macros in a programming language, which get substituted verbatim wherever they are used. And finally, structured proofs may appeal to metatheorems about JZF as proof steps. Conceptually this lets us use computation to construct the natural deduction proof we want from helper proofs that we build along the way. This

[1] *WARNING:* This structured proof is just for demonstration. You are not expected to be equipped to understand it before reading the rest of this chapter!

is a powerful means of abstraction that makes proofs much more tractable. It corresponds to treating metatheorems as admissible "rules" that could in principle be added to JZF but need not be.

As propositions and their proofs get more complex, it becomes increasingly tedious to write even structured proofs, especially because of the obligations to establish well-formedness of propositions and sets, but also because JZF rules are too rudimentary. Even more importantly, such detailed proofs can be difficult to read, since the essence of the proof can be mired in details. To address this, we add some simplifications, derived forms, and metatheorems to our process to justify *compact* structured proofs. The idea is that a compact structured proof ought to be straightforwardly expanded into a full structured proof. A compact structured proof should have enough detail that in principle the missing parts can be quite predictably and straightforwardly filled in (either by a knowledgeable human or a rather straightforward computer algorithm).

Fig. 4.3 presents a compact variant of the proof in Fig. 4.2. Most proof steps omit appeals to relevant prop and set judgments, except for identifying the appropriate set for use with ∃I in step 3, since the choice of set is significant for that rule. In addition, assumptions of use judgments are treated as though they had been used (pun intended) to deduce the corresponding verif judgment, implicitly assuming that the requisite prop can be deduced and that the identity theorem Prop. A.1 applies. The proof appeals to other metatheorems via INTROS and ∧L, and uses PICK as a meaningful abbreviation. We discuss these abbreviations and abstractions in subsequent sections.

4.2 Structured Proof in Detail

We now explain some facets of structured proof by delving in more detail into the above proof. We start with a proof of a rudimentary judgment.

Proposition 38.

Assume: *1.* A prop *2.* B prop *3.* $A \wedge B$ use
Prove: $B \wedge A$ verif

This is how we state our intent to construct a proof of the judgment

$$A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}.$$

The ASSUME keyword introduces antecedents into the context of the hypothetical judgment, and the PROVE keyword indicates the succedent. At this point, we have declared our intent to construct a proof of this judgment of the form:

$$\begin{array}{c} \vdots \\ A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif} \end{array}$$

Most propositions that we prove are simply a $\vdash \Phi$ verif judgment that has no explicit assumptions, though it may draw on previous propositions and definitions. In that case, the ASSUME and PROVE keywords are omitted and the proposition simply states the Φ that is to be verified.

Here is the entire proof:

Proof:
1. A verif
  1.1. A use by ∧E1 with :3
  1.2. Q.E.D. by atomic with 1.1, :1
2. B verif
  2.1. B use by ∧E2 with :3
  2.2. Q.E.D. by atomic with 2.1, :2
3. Q.E.D. by ∧I with 2, 1

Let's describe it in parts. Step 1:

$\vdots$
1. A verif

Proposition 36.

Assume: *1.* A prop

Prove: $\forall Q.\, (\exists S.\, S \in Q \wedge A) \Rightarrow (\exists S.\, S \in Q) \wedge A$ verif

Proof:

1. Suffices Assume: 1. Q set

 Prove: $(\exists S.\, S \in Q \wedge A) \Rightarrow (\exists S.\, S \in Q) \wedge A$ verif

 by $\forall$I

2. Suffices: 1. $\exists S.\, S \in Q \wedge A$ prop

 2. Assume: $\exists S.\, S \in Q \wedge A$ use

 Prove: $(\exists S.\, S \in Q) \wedge A$ verif

 by $\Rightarrow$I

3. $\exists S.\, S \in Q \wedge A$ prop *(Proof Omitted)*

4. Assume: 1. $\exists S.\, S \in Q \wedge A$ use

 Prove: $(\exists S.\, S \in Q) \wedge A$ verif

 4.1. Suffices Assume: 1. S_1 set

 2. $S_1 \in Q \wedge A$ use

 Prove: Q.E.D.

 4.1.1. $(\exists S.\, S \in Q) \wedge A$ prop *(Proof Omitted)*

 4.1.2. Q.E.D. by $\exists$E with 4:1, 4.1.1

 4.2. $(\exists S.\, S \in Q)$ verif

 4.2.1. Assume: S_2 set

 Prove: $S_2 \in Q$ prop

 (Proof Omitted)

 4.2.2. $S_1 \in Q$ verif

 4.2.2.1. $S_1 \in Q$ use by $\wedge$E with 4.1:2

 4.2.2.2. $S_1 \in Q$ prop *(Proof Omitted)*

 4.2.2.3. Q.E.D. by atomic with 4.2.2.1, 4.2.2.2

 4.2.3. Q.E.D. by $\exists$I with 4.2.1, 4.1:0, 4.2.2

 4.3. A verif

 4.3.1. A use by $\wedge$E with 4.1:2

 4.3.2. Q.E.D. by atomic with 4.3.1, :1

 4.4. Q.E.D. by $\wedge$I with 4.2, 4.3

5. Q.E.D. by discharging 2 with 3, 4

Figure 4.2: Structured Proof of A prop $\vdash \forall Q.\, (\exists S.\, S \in Q \wedge A) \Rightarrow (\exists S.\, S \in Q) \wedge A$ verif

Proposition 37.

Assume: A prop

Prove: $\forall Q.\, (\exists S.\, S \in Q \wedge A) \Rightarrow (\exists S.\, S \in Q) \wedge A$ verif

Proof:

1. Suffices Assume: 1. Q set

 2. $\exists S.\, S \in Q \wedge A$ use

 Prove: $(\exists S.\, S \in Q) \wedge A$ verif

 by INTROS

2. Pick

 1. New $S_1 \in Q$

 2. A use

 by $\exists$E with 1:2, $\wedge$L

3. $\exists S.\, S \in Q$ verif by $\exists$I with 2:1(set), 2:1(use)

4. Q.E.D. by $\wedge$I with 3, 2:2

Figure 4.3: Compact Structured Proof of A prop $\vdash \forall Q.\, (\exists S.\, S \in Q \wedge A) \Rightarrow (\exists S.\, S \in Q) \wedge A$ verif

   1.1. A use by $\wedge$E1 with :3
   1.2. Q.E.D. by atomic with 1.1, :1

$\vdots$

uses the available assumptions to prove an intermediate judgment, which at this point is not directly attached to the intended goal. We can visualize this as the following partial natural deduction proof:

$$\begin{array}{c}
\dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{prop}}\,(\mathrm{hypP}^{A}) \qquad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A \wedge B\ \mathsf{use}}\,(\mathrm{hypU}^{x})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{use}}\,(\wedge\mathrm{E1})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{verif}}\,(\mathrm{atomic}) \\
\vdots \\
A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}
\end{array}$$

A structured proof step can refer to any assumptions that were introduced by a surrounding step in which the current proof step is nested. One proof step surrounds another proof step if the name of the first proof step is a prefix of the second one. If a proof step introduces multiple assumptions, we use the notation ⟨step⟩:⟨asm⟩ to name the assumption, where ⟨step⟩ is the step name and ⟨asm⟩ is the assumption number. If a proof step introduces only one assumption, then we can omit assumption numbers. For example, Step 1.1, which proves A use, refers to the assumption $A \wedge B$ use introduced as part of the original proposition. The original proposition counts as the "outermost" proof step, and is given an empty step name, which is a prefix of any other step name. As such, :3 refers to the third assumption of the original proposition. So our natural deduction proof of A verif shares the same context as our goal judgment. Since subproofs only add new assumptions and never remove any, this can lead to much more concise proof statements: in programming languages terminology, assumptions are accessible according to *block structure*.

A proof step can also refer to any conclusion that was proven "above" it. One proof step is above a second proof step if either the first proof step surrounds the second (i.e., the name of the first proof step is a prefix of the second) or if the step names differ only in their trailing number and the trailing number of the first is less than the trailing number of the second. For example, Step 1.2 refers to Step 1.1, which is ok because both share the prefix "1." and the trailing number "1" of the referenced step is less than the trailing number "2" of the referencing step.

In Lamport's convention, a proof step can be labelled Q.E.D. when it is a proof of the goal judgment currently active in context. In Step 1.2, Q.E.D. abbreviates the judgment being proved, which was originally stated at Step 1: A verif. Q.E.D. is short for the Latin *quod erat demonstrandum* which translates literally to "which was to be demonstrated." Traditionally mathematicians have placed this acronym at the end of the proof of a theorem. It's worth observing that in modern French, the word for "proof" is "demonstration", so this acronym indicates the end of a proof. Here Step 1.2 is the proof of the judgment that completes the proof of the judgment stated at Step 1. As we will see, structured proofs often use Q.E.D., and each instance refers to some surrounding already-stated judgment. We could have instead repeated A verif, but using this abbreviation helps provide visual structure to a proof, and decreases the amount of repetition, which is especially nice when judgments get complicated, and when you want to quickly identify the structure of a proof.

Now let's add Step 2 to the proof:

$\vdots$

2. B verif
   2.1. B use by $\wedge$E2 with :3
   2.2. Q.E.D. by atomic with 2.1, :2

$\vdots$

This structured proof so far corresponds to the (partial) natural deduction proof:

$$\begin{array}{c}
\dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{prop}}\,(\mathrm{hypP}^{A}) \quad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A \wedge B\ \mathsf{use}}\,(\mathrm{hypU}^{x})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{use}}\,(\wedge\mathrm{E1})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{verif}}\,(\mathrm{atomic}) \;\; \cdots \;\; \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{prop}}\,(\mathrm{hypP}^{A}) \quad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A \wedge B\ \mathsf{use}}\,(\mathrm{hypU}^{x})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{use}}\,(\wedge\mathrm{E1})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{verif}}\,(\mathrm{atomic}) \\
\vdots \\
A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}
\end{array}$$

The partial natural deduction proof is made up of two completed subproofs, which are floating above the goal judgment. In general a structured proof can be viewed as a *forest* of proofs: a collection of distinct that are typically connected together into a single proof by later proof steps.[2] Our structured proof could have reordered the two subproofs, swapping Step 1 and Step 2, without changing the corresponding partial natural deduction proof. This flexibility empowers the proof author to choose the ordering and structure of proof that suits their style and perhaps helps a reader take in the pieces in a conceptually compelling order and narrative.

Finally, Step 3 introduces the last required rule that turns the forest into a unified proof tree:

⋮

3. Q.E.D. by ∧I with 2, 1

This step yields the completed natural deduction proof:

$$\dfrac{\dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{prop}}\,(\text{hypP}^A) \quad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A \wedge B\ \mathsf{use}}\,(\text{hypU}^x)}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{use}}\,(\wedge\text{E1})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{verif}}\,(\text{atomic}) \quad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{prop}}\,(\text{hypP}^A) \quad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A \wedge B\ \mathsf{use}}\,(\text{hypU}^x)}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{use}}\,(\wedge\text{E1})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{verif}}\,(\text{atomic})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}}\,(\wedge\text{I})$$

Now you can see three different steps marked Q.E.D., each of which refers to a different goal judgment stated earlier. This last rule simultaneously connects the goal to each of the subproofs, but also determines where the two subproofs are situated with respect to one another (each is a premise of the goal).

Though natural deduction proofs are always trees, a structured proof can refer to a single proof step multiple times in multiple proof steps. As such, structured proofs in general can form a directed acyclic graph (DAG). For example, to prove $A \wedge A$ verif, instead of $A \wedge B$ verif, we need only prove A verif once, but refer to it twice in the last step. Nonetheless the corresponding natural deduction tree would have two copies of the subproof, thereby expanding the DAG into a tree.

4.2.1 SUFFICES steps

The structured proof above demonstrates the most common structuring technique for a structured proof: construct complete individual subproofs as steps, each of which is possibly made up of nested subproofs in turn, and then those subproofs are used in the proofs of subsequent steps. Eventually those sub-proofs are connected to form a proof of the overall goal.

For the most part, you can view this as a "top-down" process, wherein every proof step is a complete proof of some judgment. This approach is flexible in the sense that the proof can be broken up at many levels of granularity: each top-level proof step could be a single JZF rule, with the rule stated as an in-line proof method; or each proof step could be one premises of the final rule that proves the overall goal; or anything in between.

But for the purposes of explanation, a top-down process is not always ideal. It can be beneficial to structure some parts of a proof "bottom-up", wherein one shows that some proof goal could be achieved if only first some other judgments could be proven. Once it is established that these other judgments *suffice* to prove the stated goal, those other judgments become the goal of the proof. To do this we use a SUFFICES proof step.

Here's a restructuring of our earlier proof in line with this approach.

PROOF:
1. SUFFICES: A verif
   1.1. B verif
      1.1.1. B use by ∧E2 with :3
      1.1.2. Q.E.D. by atomic with 1.1.1, :2
   1.2. Q.E.D. by ∧I with 1.1, 1
2. Q.E.D.
   2.1. A use by ∧E1 with :3
   2.2. Q.E.D. by atomic with 2.1, :1

[2] In principle a structured proof, like a program, could have "dead subtrees" that are never used to prove the original judgment, but that does not seem helpful for a completed proof.

The interesting part of this proof is Step 1, after which the corresponding partial natural deduction proof is as follows:

$$\dfrac{\dfrac{\vdots}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{verif}} \qquad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{prop}}\,(\mathrm{hypP}^A) \quad \dfrac{\dfrac{}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A \wedge B\ \mathsf{use}}\,(\mathrm{hypU}^x)}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{use}}\,(\wedge\mathrm{E1})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{verif}}\,(\mathrm{atomic})}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}}\,(\wedge\mathrm{I})$$

Whereas a typical step produces a completed subproof of a theorem, Step 1 produces a *partial* proof of the current goal $B \wedge A$ verif, which requires a subproof of A verif in order to form a complete proof. As such, the SUFFICES proof of Step 1 *changes the overall goal* to be A verif. As a result, Step 2's use of Q.E.D. refers to this new goal, as does Step 2.2 since it refers to Step 2's goal.

The step:

1. SUFFICES: A verif

means "given a proof of A verif, the current goal (Q.E.D., a.k.a. $B \wedge A$ verif) can be proven." Typically a mathematician will phrase this "to prove the current goal, it suffices to show that ...", hence the use of the keyword SUFFICES. The proof of Step 1 formally demonstrates why it indeed suffices to prove A verif. That proof now contains both the proof of B verif, as well as the final proof step that would link a proof of A verif to the overall goal. In other words, the body of the proof of Step 1 may appeal to the stated conclusion *as though it had been proven.* It is not an assumption, but rather a *pending proof obligation.* Upon arriving at Step 2, the current goal (i.e. that which Q.E.D. refers to) has changed, so the rest of the proof must discharge the remaining proof obligations to fill in this gap.

Using a SUFFICES step, we can also leave multiple unfinished proof obligations, which means that there are now multiple goals:

PROOF:
1. SUFFICES: 1. A verif
                    2. B verif
   by $\wedge$I with 1:2, 1:1
2. A verif [Q.E.D. 1:1]
   2.1. A use by $\wedge$E1 with :3
   2.2. Q.E.D. by atomic with 2.1, :1
3. B verif [Q.E.D. 1:2]
   3.1. B use by $\wedge$E2 with :3
   3.2. Q.E.D. by atomic with 3.1, :2

Here, step 1 corresponds to the partial natural deduction tree:

$$\dfrac{\dfrac{\vdots}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash A\ \mathsf{verif}} \quad \dfrac{\vdots}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B\ \mathsf{verif}}}{A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}}\,(\wedge\mathrm{I})$$

Step 1 introduces multiple goals using an enumeration. Then we dispatch each goal in a separate step that refers back to the relevant goal. For greater clarity, those sub-goal steps list both the full judgment name, and note that they dispatch outstanding goals by writing "[Q.E.D. xxx]" with xxx referring to the specific proof obligation. The same notation can be used for single-goal situations as well, since it may improve readability. Alternatively, or in addition, one may add an extra step 4 that proves "Q.E.D. by discharging 1 with 2, 3." In this case Step 1 lists all of the outstanding proof obligations, of which there are 2, and Steps 2 and 3 discharge them. If distinct steps state the remaining proof obligations, then they may be discharged by writing "Q.E.D. by discharging aaa with bbb, ccc with ddd, ..."

The use of SUFFICES in the first structured proof is just to demonstrate the feature: I think that the original proof has a better pedagogical structure. The second, where both premises are left as goals, is reasonable, though it's not obviously better exposition than the first. Nonetheless, this approach will prove helpful often for controlling the structure of how a proof is explained. Often you want to work bottom-up to swiftly dispatch some common but uninteresting details so that the rest of the proof can focus on the core. Other times, while constructing a proof you may find it helpful to start out in a goal-directed manner in order to try and uncover the right strategy for proving a theorem. Having more than one way to build up a proof is purely a bonus.

4.2.2 Adding Assumptions

In the natural deduction proof of our present judgment, every hypothetical judgment had the same hypotheses, namely $A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use}$. In general, though, proofs of judgments may consist of subproofs that have more hypotheses than those that appear in the overall goal. Furthermore, subproofs will never have fewer hypotheses.[3] Just like when stating the overall theorem, a subproof can use the ASSUME-PROVE form to introduce new assumptions that are available for scope of the relevant subproof.

To demonstrate this, let's prove a new theorem that builds on our old one:

Proposition 39.
Assume: *1.* $A\ \mathsf{prop}$
2. $B\ \mathsf{prop}$
Prove: $A \wedge B \Rightarrow B \wedge A\ \mathsf{verif}$

Proof:
1. Assume: $A \wedge B\ \mathsf{use}$
   Prove: $B \wedge A\ \mathsf{verif}$ *(Proof Omitted)*
2. Q.E.D. by ⇒I with 1

This partial structured proof corresponds to the partial natural deduction tree:

$$\dfrac{\begin{array}{c}\vdots\\ A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \wedge B\ \mathsf{use} \vdash B \wedge A\ \mathsf{verif}\end{array}}{A\ \mathsf{prop}, B\ \mathsf{prop}, A \wedge B\ \mathsf{use} \vdash A \wedge B \Rightarrow B \wedge A\ \mathsf{verif}}\ (\Rightarrow\mathrm{I}^x)$$

The premise of the last rule coincides with our previous proof, and has one extra assumption compared to the overall goal, $x : A \wedge B\ \mathsf{use}$. In the structured proof, the extra assumption is in scope in the body of the omitted proof, which would nearly match our previous proof, except that references to this new assumption would point to Step 1 rather than to the overall goal.

We can also combine ASSUME-PROVE with SUFFICES to construct the same proof using backward reasoning:

Proof:
1. Suffices Assume: $A \wedge B\ \mathsf{use}$
   Prove: $B \wedge A\ \mathsf{verif}$
   by ⇒I with 1

$\vdots$

In this proof, the proof of Step 1 demonstrates how to deduce the overall goal from stated judgment. However, the new assumption is *not* in scope for the subproof: it comes into scope for the remainder of the proof, here replaced with dots, where the goal has changed to $B \wedge A\ \mathsf{verif}$. Thus, the remainder of the proof is a carbon copy of our earlier proof, with the proof steps renumbered to follow Step 1, and with references to Step 1 when the assumption $A \wedge B\ \mathsf{use}$ are needed.

4.3 LET: Notational Definitions

Structured proofs support the introduction of notational definitions, using LET notation. It is thus possible to abbreviate a complex set expression or proposition expression as a symbol. Then, any use of that notational definition within its scope (following rules analogous to those for assumptions) will stand in for the replacement of the defined symbol with the text of its definition. For conciseness and abstraction, we can use Let binding to introduce syntactic sugar where needed. e.g.

[3] However, some logical systems rely on reasoning about fewer hypotheses in subproofs

Proposition 40.
Suffices Assume: 1. A set
Prove: $A \in \{ A, A, A, A, A \}$

Proof:
Let: $B := \{ A, A, A, A, A \}$
1. $A \in B$
  1.1. …

In the above sketch, it may be convenient for the sake of the proof to abbreviate the long set name $\{ A, A, A, A, A \}$ as B during the proof of the proposition, which occurs in step 1.[4] In fact, $\{ A, A, A, A, A \}$ is itself a notational definition for the extensional set notation introduced in Ch. 2.

A let binding is in scope for the subsequent lexical scope (all subsequent steps at the same level or deeper prior to the end of the level at which the let binding was introduced). A LET binding can appear as a non-numbered step (as below), or can appear in a numbered step, if it proves clarifying for a proof step to refer to the definition.

In addition to abbreviating complex set expressions, notational definitions can also be introduced for complex propositional expressions, like:

Let: $d \mid a := \exists c \in \mathbb{N}.\, dc = a$

which introduces the notation $d \mid a$ for "d divides a evenly." Both set and proposition notational definitions can be parameterized with respect to proposition and set arguments, as we have alluded to in previous chapters and in the above example. Here is an example of a set notation parameterized by propositions and sets:

Let: $\bigcup_{d|a} F(d) := \bigcup \{ F(d) \textbf{ for } d \in \{ n \in \mathbb{N} \mid (n \mid a) \} \}$

which introduces the notation $\bigcup_{d|a} F(d)$ to mean the set that consists of the union of all sets $F(d)$, where each $F(d)$ is produced from some number d that divides a evenly. What a mouthful for this concept! But once we understand the concept, the compact notation yields clarity. In fact we could abstract this more:

Let: $\bigcup_{\Phi(d)} F(d) := \bigcup \{ F(d) \textbf{ for } d \in \{ n \in \mathbb{N} \mid \Phi(n) \} \}$

and thereby generalize taking the union of sets $F(d)$ where unions of sets produced by numbers d that satisfy Φ. Generalizing this notation for sets other than numbers $n \in \mathbb{N}$ requires some care in order to ensure that the resulting expression properly describes a set.

LET can also be used as a simple proof step, in which case it simultaneously introduces a notational definition for a set expression, and asserts that the given set expression satisfies the set or prop judgment. The proof of the step establishes that. For instance, our proof above could be written:

Proof:
1. Let: $B := \{ A, A, A, A, A \}$
2. $A \in B$
  2.1. …

And any appeal to the judgment B set can refer to step 1.

4.4 Derived Forms

The constructs introduced so far, top-down (complete) proof steps, bottom-up (partial) proof steps, block-scoped assumptions, Q.E.D. proofs, and notational definitions, constitute the primitive elements of structured proofs. Structured proofs also have a few convenience mechanisms that either make things shorter, or help to write proofs in a style that more closely parallels prose proofs, evoking similar terminology. What follows are those derived forms.

[4] Step 1 could be phrased Q.E.D., but introducing the notational definition in context can help the reader.

4.4.1 Pick

Consider the JZF rule for existential elimination:

$$\dfrac{\Gamma \vdash \exists S_1.\, \Phi_1(S_1)\ \mathsf{use} \quad \Gamma, S_2\ \mathsf{set}, \ell_1 : \Phi_1(S_2)\ \mathsf{use} \vdash \Phi_2\ \mathsf{verif} \quad \Gamma \vdash \Phi_2\ \mathsf{prop}}{\Gamma \vdash \Phi_2\ \mathsf{verif}}\ (\exists\mathrm{E}^{S_2,\ell_1})$$

Given license to use an existentially quantified proposition, we can do so by assuming some set identifier that satisfies the proposition, and using that information to prove something that depends on it.

We can already do so using structured proofs, but structured proofs also provide the PICK proof step to concisely express the common case.

To explain, consider the following “conservation” proof:

$$\dfrac{\dfrac{}{\Gamma_0 \vdash \exists S.S \in Q\ \mathsf{use}}\,(\mathrm{hypU}^x) \quad \dfrac{\dfrac{\vdots}{\Gamma_1, S_3\ \mathsf{set} \vdash S_3 \in Q\ \mathsf{prop}}\,(\in\mathrm{P}) \quad \dfrac{}{\Gamma_1 \vdash S_2\ \mathsf{set}}\,(\mathrm{hypS}^{S_2}) \quad \dfrac{\dfrac{}{\Gamma_1 \vdash S_2 \in Q\ \mathsf{use}}\,(\mathrm{hypU}^y) \quad \dfrac{\vdots}{\Gamma_1 \vdash S_2 \in Q\ \mathsf{prop}}}{\Gamma_1 \vdash S_2 \in Q\ \mathsf{verif}}\,(\mathrm{atomic})}{(\Gamma_1 := \Gamma_0, S_2\ \mathsf{set}, y : S_2 \in Q\ \mathsf{use} \vdash \exists S.S \in Q\ \mathsf{verif}}\,(\exists\mathrm{I}) \quad \dfrac{\vdots}{\Gamma_0 \vdash \exists S.S \in Q\ \mathsf{prop}}\,(\exists\mathrm{P})}{(\Gamma_0 := Q\ \mathsf{set}, x : \exists S.\, S \in Q\ \mathsf{use}) \vdash \exists S.S \in Q\ \mathsf{verif}}\,(\exists\mathrm{E}^{S_2,x})$$

Working bottom-up, we could write a structured proof where the first step is:

Proposition 41.

Assume: *1.* Q set
2. $\exists S.\, S \in Q$ use
Prove: $\exists S.S \in Q$ verif

Proof:
1. Suffices Assume: 1. S_2 set
2. $S_2 \in Q$ use
Prove: Q.E.D.
   1.1. $\exists S.S \in Q$ use by assumption :2
   1.2. $\exists S.S \in Q$ prop (*Proof Omitted*)
   1.3. Q.E.D. by $\exists$E with 1.2, 1.3

2. $\vdots$

Step 1 uses bottom-up reasoning to reduce the proof to the hypothetical judgment that assumes a set with the appropriate properties. Then Step 2 proves the previous goal, but with new assumptions. Since this shape is so common, you can write this more concisely as follows:

1. Pick
   1. S_2 set
   2. $S_2 \in Q$ use

$\vdots$

The PICK form is meant to be specifically for unpacking an existential in-line and continuing to prove the pre-existing goal, but now with extra assumptions acquired from the existential. When the existential is a conjunction of properties, then the $\wedge$L metatheorem can be referenced to separate the conjunction into multiple assumptions, as was done in Fig. 4.3 (See Sec. 4.5.3).

4.4.2 NEW

The last example captures another common pattern: when working with quantified variables, it’s quite common to make two sequential assumptions: the availability of some set identifier as a set, followed by the assumption of its membership in some other set. These two assumptions can be collapsed and expressed using the NEW directive:

1. Pick
   1. New $S_2 \in Q$

$\vdots$

This single assumption can now be referenced when either the set judgment or the use judgment are needed. Such references can be disambiguated by following the reference with the relevant judgment in parentheses.

Because it's so common to introduce many sets simultaneously that are each elements of the same set (the "domain"), we can also write:

1. PICK
   1. NEW $S_1, S_2, S_3, \ldots, S_n \in Q$

$\vdots$

Of course doing this means that uses of the assumption label are less precise: one must determine which particular set is being referenced, and whether sethood, elementhood, or both are being used. But often the conciseness clarifies than the ambiguity obscures.

4.4.3 CASE

The use of disjunctive and existential propositions have a similar structure, in that both have elimination rules that require the deduction of a conclusions under new assumptions. As such, it makes sense that just as existentials receive custom structured support in the form of PICK, disjunctions receive custom support in the form of CASE, which corresponds to case analysis in prose proofs.

Consider the JZF rule for disjunction elimination:

$$\dfrac{\Gamma \vdash \Phi_1 \lor \Phi_2\ \mathsf{use} \quad \Gamma, \ell_1 : \Phi_1\ \mathsf{use} \vdash \Phi_3\ \mathsf{verif} \quad \Gamma, \ell_2 : \Phi_2\ \mathsf{use} \vdash \Phi_3\ \mathsf{verif} \quad \Gamma \vdash \Phi_3\ \mathsf{prop}}{\Gamma \vdash \Phi_3\ \mathsf{verif}}\ (\lor\mathrm{E}^{\ell_1,\ell_2})$$

Among its premises are two hypothetical judgments $\Gamma, \ell_1 : \Phi_1\ \mathsf{use} \vdash \Phi_3\ \mathsf{verif}$ and $\Gamma, \ell_2 : \Phi_2\ \mathsf{use} \vdash \Phi_3\ \mathsf{verif}$, each corresponding to one of the disjuncts. We can demonstrate its use with the bones of another conservation lemma:

$$\dfrac{\dfrac{}{\Gamma_0 \vdash A \lor B\ \mathsf{use}}\ (\mathrm{hypU}^x) \quad \begin{array}{c}\vdots\\ \Gamma_0, y : A\ \mathsf{use} \vdash A \lor B\ \mathsf{verif}\end{array} \quad \begin{array}{c}\vdots\\ \Gamma_0, z : B\ \mathsf{use} \vdash A \lor B\ \mathsf{verif}\end{array} \quad \begin{array}{c}\vdots\\ \Gamma_0 \vdash A \lor B\ \mathsf{prop}\end{array}}{(\Gamma_0 := A\ \mathsf{prop}, B\ \mathsf{prop}, x : A \lor B\ \mathsf{use}) \vdash A \lor B\ \mathsf{verif}}\ (\lor\mathrm{E}^{y,z})$$

In a structured proof of this judgment, we can introduce two ASSUME-PROVE steps, one for each of the core hypothetical judgments, as in the following sketch of a proof using SUFFICES.

Proposition 42.

ASSUME: *1.* $A\ \mathsf{prop}$
2. $B\ \mathsf{prop}$
3. $A \lor B\ \mathsf{use}$
PROVE: $A \lor B\ \mathsf{verif}$

PROOF:
1. SUFFICES: 1. ASSUME: $A\ \mathsf{use}$
PROVE: Q.E.D.
2. ASSUME: $B\ \mathsf{use}$
PROVE: Q.E.D.
1.1. $A \lor B\ \mathsf{use}$ by :3
1.2. $A \lor B\ \mathsf{prop}$ *Proof Omitted*
1.3. Q.E.D. by $\lor$E
2. ASSUME: $A\ \mathsf{use}$
PROVE: Q.E.D. [Q.E.D. 1:1]

$\vdots$

3. ASSUME: $B\ \mathsf{use}$
PROVE: Q.E.D. [Q.E.D. 1:2]

⋮

But we can abbreviate each case of ASSUME-PROVE Q.E.D. here with CASE, as in the following:

Proof:
1. Suffices: 1. Case: A use
   2. Case: B use
   1.1. $A \vee B$ use by :3
   1.2. $A \vee B$ prop *Proof Omitted*
   1.3. Q.E.D. by $\vee$E
2. Case: A use [Q.E.D. 1:1]

   ⋮

3. Case: B use [Q.E.D. 1:2]

   ⋮

CASE differs from PICK in that CASE abbreviates a top-down ASSUME-PROVE form, which is for constructing self-contained subproofs, whereas PICK abbreviates a bottom-up SUFFICES-ASSUME-PROVE form, which is for constructing partial proofs that change the goal(s) of surrounding proof to be those omitted premises that have not yet been proven. Indeed, CASE *could* be applied when using an existential proposition, but doing so less closely matches the structure of more typical prose proofs.

4.4.4 Definitional (and Propositional) Reasoning

JZF is strict about how and when definite descriptions can be used. In particular, the (ddS) rule says that a definite description is not even a well-formed set expression unless unique existence of the described set has been verified, and the same holds for appeals to use of the defining property. Failure to prove unique existence all too often leads to errors in mathematical reasoning. To simultaneously accommodate these proof obligations and introduce abbreviated notation for descriptions, we use the Define : form. For example consider a proof fragment:[5]

1. Define: $\mathit{fact} := \iota F \in \mathbb{N} \to \mathbb{N}.\, F(0) = 1 \wedge \forall n \in \mathbb{N}.\, F(n+1) = (n+1)F(n)$
   Proof sketch: by the Principle of Recursion on $\mathbb{N}$

   ⋮

Step 1 introduces a notational definition *fact* for the factorial function, much like the Let form. Unlike Let, however, Define requires a proof of the corresponding unique existence proposition,

1. $\exists! F \in \mathbb{N} \to \mathbb{N}.\, F(0) = 1 \wedge \forall n \in \mathbb{N}.\, F(n+1) = (n+1)F(n)$ verif

As usual, one might mark the proof obligation as *omitted*, which at least makes clear and visible the necessity of a proof. Step 1 may now be referred to, either as evidence that *fact* set, or to use the defining property of *fact*.

In practice, we present function definitions using common informal notation:

1. Define: $\mathit{fact} : \mathbb{N} \to \mathbb{N}$
   $\mathit{fact}(0) = 1$
   $\mathit{fact}(n+1) = (n+1) * \mathit{fact}(n)$
   Proof sketch: by the Principle of Recursion on $\mathbb{N}$

   ⋮

We allow Defines that appeal to set notations to be given without proof, so long as the constituents are well-defined themselves (which had better be the case). For example:

1. Define: Even $:= \{\, n \in \mathbb{N} \mid \exists d \in \mathbb{N}.\, 2 * d = n \,\}$ *(Proof Omitted)*

   ⋮

[5] Here, $F(n)$ is shorthand for the explicitly annotated partial function application notation $F[\mathbb{N} \to \mathbb{N}](n)$, where the domain and codomain of the function are deduced from context.

This is the whole point of our notations.

After introducing a definition using set notation, proof steps may *use* the defining property of a definite description. These definitions, and those for functions, typically quantify over sets and then conjoin properties, often in the form of a bi-implication $\Psi \Leftrightarrow \Phi$. For example, the following proof explicitly extracts the reasoning principle from our set definition:

1. Define: $\textsc{Even} := \{\, n \in \mathbb{N} \mid \exists d \in \mathbb{N}.\, 2 * d = n \,\}$ *(Proof Omitted)*
2. $8 \in \textsc{Even}$ verif
   2.1. $8 \in \mathbb{N}$ verif
   2.2. $2 * 4 = 8$ verif
   2.3. $\exists d \in \mathbb{N}.\, 2 * d = 8$ verif by ∃I with 4 set, 2.2
   2.4. $8 \in \mathbb{N} \land \exists d \in \mathbb{N}.\, 2 * d = 8$ verif by ∧I with 2.1,2.3
   2.5. $\forall S.\, S \in \textsc{Even} \Leftrightarrow S \in \mathbb{N} \land \exists d \in \mathbb{N}.\, 2d = S$ use by dd with 1
   2.6. $8 \in \textsc{Even} \Leftrightarrow 8 \in \mathbb{N} \land \exists d \in \mathbb{N}.\, 2 * d = 8$ use by ∀E with 2.5, 8 set
   2.7. $8 \in \textsc{Even} \Leftarrow 8 \in \mathbb{N} \land \exists d \in \mathbb{N}.\, 2 * d = 8$ use by ∧E2 with 2.6
   2.8. $8 \in \textsc{Even}$ use by ⇒E with 2.7, 2.4
   2.9. Q.E.D. by atomic with 2.8,1,8 set

The one exception to omitting proof obligations is that any use of generalized replacement notation $\{\, B \leftarrow \Phi(A, B)\ \mathbf{for}\ A \in S \,\}$ requires a proof that $\forall A \in S.\, \exists! B.\, \Phi(A, B)$. The specialized functional replacement notation $\{\, F(A)\ \mathbf{for}\ A \in S \,\}$ typically requires no proof since the expression $F(A)$ should clearly denote one value for each set $A \in S$. But be careful: a set like $\{\, 5/n\ \mathbf{for}\ n \in N \,\}$ is ill-formed because $5/n$ is not defined for $n = 0$!

4.4.5 Algebraic Proof Steps and Synthetic Proof Rules

As we saw in the last proof, stating each individual proof step one by one can get pretty tedious, especially when they involve logical principles with which you have become quite familiar: breaking a bi-implication into two pieces so that you can apply just one of them, for example. Stating the intermediate result can be helpful for the new learner, but a distraction from the essence of a proof. Nonetheless it can be useful to capture the proof steps involved. To do so, one can rephrase proof steps *algebraically*, as though each proof step took arguments. For example, if one writes "by ∀E with 2.4, 8 set", then you can think of this as an operator that takes two arguments, the proof of step 2.4 as well as the (unstated) proof that 8 set. We could instead write this $\forall\mathrm{E}(2.4, 8\ \mathsf{set})$.

This expression itself, which serves as the proof of step 2.6 above, can be treated as an argument to another proof step, so "by ∧E2 with 2.6" could be rewritten "$\land\mathrm{E2}(\forall\mathrm{E}(2.4, 8\ \mathsf{set}))$."

Following this pattern, we can abbreviate the proof from the previous section as follows:

1. Define: $\textsc{Even} := \{\, n \in \mathbb{N} \mid \exists d \in \mathbb{N}.\, 2 * d = n \,\}$ *(Proof Omitted)*
2. $8 \in \textsc{Even}$
   2.1. $8 \in \mathbb{N}$
   2.2. $2 * 4 = 8$
   2.3. $\exists d \in \mathbb{N}.\, 2 * d = 8$ by ∃I with 4 set, 2.2
   2.4. Q.E.D. by $\mathrm{atomic}(\Rightarrow\mathrm{E}(\land\mathrm{E2}(\forall\mathrm{E}(\mathrm{dd}(1), 8\ \mathsf{set})), \land\mathrm{I}(2.1, 2.3)), 1, 8\ \mathsf{set})$

This structured proof is equivalent to the proof from the previous section, but Step 2.4 is justified by an algebraic proof that embodies Steps 2.4–2.9 of the prior version.

Warning: For now, we will *not* use algebraic proof steps to make things concise. Once we have mastered step-by-step proofs, we'll "unlock" this advance move.

However, for now it can be a convenient way to explain some higher-level abstractions that we *will* use before long, because a little bit of abstraction can become useful without undermining understanding, especially when capturing patterns of use of our axioms about equality and elementhood. For instance, we quickly discover that the separation axiom, which is stated as a bi-implication, naturally breaks into an "introduction" component, for establishing that one set is an element of another set that was defined using separation, as well as an "elimination" component, for deducing properties of one set that is already known to be an element of a separation-described set. So these rules "introduce" and "eliminate" propositions about membership in a set

Let's rewrite the above proof using such principles, and state the precise relationship.

1. DEFINE: EVEN $:= \{\, n \in \mathbb{N} \mid \exists d \in \mathbb{N}.\, 2 * d = n \,\}$ *(Proof Omitted)*
2. $8 \in$ EVEN
   - 2.1. $8 \in \mathbb{N}$
   - 2.2. $2 * 4 = 8$
   - 2.3. $\exists d \in \mathbb{N}.\, 2 * d = 8$ by $\exists$I with $4\,\mathsf{set}$, 2.2
   - 2.4. Q.E.D. by sepI with 1, $8\,\mathsf{set}$, 2.1, 2.3

This version of the proof provides a simple named abstraction of the previous combination of low-level logical operations, forming a high-level operation that is more closely aligned with informal proofs, but just as rigorous and precise as our earlier detailed proof. The name sepI is short for "Separation (Elementhood) Introduction", and has the style of our other rules. One may also imagine having an "overloaded" version of this rule that accepts a pre-existing conjunction (e.g. of 2.1 and 2.3). The number of arguments to these two synthetic rules could disambiguate them. A corresponding "Separation (Elementhood) Elimination" rule (sepE) could similarly be introduced.

As with the direct statement of $8\,\mathsf{set}$ as an argument to a proof, rather than stating it as an explicit proof step that has no explicit associated proof, one can imagine doing the same with any judgment. However we restrict ourselves to judgments that are as nearly immediately evident as possible, like basic set judgments. Otherwise, one risks making the dependency on a particular judgment invisible. This should only be done when the omission yields more clarity than its presence (as when some detail detracts from the essential facets of a proof).

4.4.6 Equational/Relational Reasoning

Many proofs about mathematical objects can be most conveniently structured as a sequence of binary relations, often equalities, that relate objects. One benefit of this style is its brevity, especially in that mathematical expressions need not be repeated once for its right-hand-side occurrence and once for its left-hand-side occurrence. To achieve this compression, a structured proof can use the $\bullet$ symbol on the left of a binary relation as an abbreviation for the right-hand object of the previous proof step at the same level, in which case the previous proof must be of a binary relation.

For instance:

1. $(1 + 1) + 1 = 2 + 1$ use *(Proof Omitted)*
2. $2 + 1 < 3 + 1$ use *(Proof Omitted)*
3. $3 + 1 = 4$ use *(Proof Omitted)*

can be rewritten equivalently as follows:

1. $(1 + 1) + 1 = 2 + 1$ use *(Proof Omitted)*
2. $\bullet < 3 + 1$ use *(Proof Omitted)*
3. $\bullet = 4$ use *(Proof Omitted)*

4.5 Generalizations

JZF is intentionally presented as a very small proof theory that captures the essence of our logical reasoning principles. It serves as a robust standard against which we can judge proofs to be rigorous. But it's radical minimality makes it rather impractical to use for day-to-day proof. It is much like an intermediate language of a compiler for a programming language. Languages are often implemented by translating to a small intermediate language that is easy for a computer to manipulate and generate good code. That way, the compiler writer can, for instance, implement the compilation of a single looping construct, and then the compiler front-end can translate source programs, which may use many different looping constructs, into this one construct that gets handled uniformly by the compiler.

Similarly, JZF's size makes it amenable to expressing our core reasoning principles, but for the sake of proving theorems, we want a richer "source" language that can be understood in terms of "compiling" to JZF. As it happens, the various metatheorems that we stated about JZF, which describe how the existence of some JZF proofs entail the existence of other related JZF proofs, describe convenience operations we can exploit in structured proofs. Then our proof rules appeal to those metatheorems, which, instead of just building up a proof by adding rules, "compute" a new proof that is related to the old proof. We show some examples here.

4.5.1 Conservativity and composition

When sticking with the core of JZF, using a proposition to verify a related proposition can be brutally tedious. Take for instance:

Proposition 43.
Assume: *1.* A prop
2. B prop
3. $A \vee B$ use
Prove: $(A \vee B) \wedge \top$ verif

The complete proof is painful:

1. Assume: A use
   Prove: Q.E.D.
   by *Proof Omitted*
   1.1. A verif by atomic with :1, 1
   1.2. $A \vee B$ verif by $\vee$I1 with :2,1.1
   1.3. $\top$ verif by $\top$I
   1.4. Q.E.D. by $\wedge$I with 1.1, 1.3
2. Assume: B use
   Prove: Q.E.D.
   by *Proof Omitted: analogous to 1*
3. Q.E.D. by $\vee$E with :3, 1, 2

JZF distinguishes between use and verification, and requires that verifications be built up from the most basic propositional components. Luckily our conservativity metatheorem (Prop. A.1) ensures that this can always be done, so long as the relevant proposition is well-formed. This motivates our ability to in roughly one step transition from use to verif:

1. $A \vee B$ prop by $\vee$P with :1, :2
2. $A \vee B$ verif by assumption with 1, :3
3. $\top$ verif by $\top$I
4. Q.E.D. by $\wedge$I with 2, 3

We use the rule name "assumption" to describe appealing to a propositional assumption, and allow the result to be a verif rather than a use by including evidence that the proposition is prop. In simple cases like the above, where no definite descriptions are involved, one may simply identify the relevant propositional variable assumptions (:1, :2) rather than constructing the actual proof of prop.

4.5.2 Arbitrary-arity Conjunctions and Disjunctions

The JZF rules for conjunctions and disjunctions are strictly binary, and arguably the rules for $\top$ and $\bot$ can be viewed as the zero-ary analogues. Such rules make for a concise and clear exposition of the logical principles involved, but we regularly wish to work with arbitrary finite conjunctions and disjunctions. These can be carefully encoded using some choice of associativity, and the corresponding introduction and elimination principles can be shown to be derivable.

In our structured proofs, we assume the ability to perform such arbitrary conjunctions and disjunctions. We can assume that they translate to an appropriate encoding in JZF, or similarly that JZF is enriched with arbitrary arity conjunctions and disjunctions. Thus we can write $\Phi_1 \wedge \Phi_2 \wedge \cdots \Phi_n$ without concern for associativity or nesting, and extract a conjunct using $\wedge$Ei for any appropriate index i, possibly omitting the index when it can be inferred from context. We can do the same for disjunction. Introducing an n-ary disjunction requires establishing that $n-1$ of the disjuncts can be judged prop.

When eliminating an implication whose premise is an n-ary conjunct, it suffices to appeal to the individual components of the premise, rather than explicitly constructing a conjunction and then later applying it.

4.5.3 Assumption Introduction: INTROS

When working in set theory, we often find ourselves proving theorems that have a form similar to the following:

$$\forall S_1 \in Q_1.\, A \wedge B \Rightarrow \forall S_2 \in Q_2.\, C \wedge D \Rightarrow E.$$

This particular proposition involves the iteration of universal quantifications $\forall S_i.\Phi(S_i)$ and implications $\Phi_1 \Rightarrow \Phi_2$, where the premise of an implication may be a conjunction $\Phi_1 \wedge \Phi_2$. Thanks to one of our notational definitions, we also have $\forall S \in Q.\Phi(S) \equiv \forall S.S \in Q \Rightarrow \Phi(S)$, so each of the above universal quantifications is hiding more implications.

This pattern is common because it amounts to making set and prop assumptions, although wrapped up in a single proposition. Using bottom-up reasoning according to the ∀I, ⇒I rules and the ∧L metatheorem, we can decompose this proposition into a hypothetical judgment with the assumptions we want in scope. [6] However, doing so with one SUFFICES step per construct would be tedious, and would obscure the fundamental structure of a proof. Instead, one can appeal to INTROS as an abbreviation for the appropriate iteration of these rules, e.g.:[7]

1. SUFFICES ASSUME: 1. S_1 set
2. $S_1 \in Q_1$ use
3. S_2 set
4. $S_2 \in Q_2$ use
5. A use
6. B use
7. C use
8. D use

PROVE: E verif

by INTROS

2. ⋮

By inspecting the new goal, one could reconstruct the appropriate iteration of rules and metatheorems needed to introduce the assumptions above. The new assumptions can be reordered freely since JZF assumptions have no set order. However, as usual, when applying ∀I, one may need to rename the set in order to satisfy the uniqueness of the names of assumed sets.

4.5.4 Structured Proofs and Proof Sketches in Practice

Now that we have the basic concepts of structured proof down, let's consider how structured proofs can be used in practice. The basic concepts give us sufficient precision to prove in gory detail how any theorem follows from the rules, judgments, and axioms that make up JZF. That access to detailed proof can give us substantial confidence in our arguments: they can be built up, piece by piece, from the foundations our our reasoning up to the penthouse suite of domain-relevant propositions.

In practice, though, that level of detail can detract from the communication of the essence of our results, burying the important ideas under layers of logical bureaucracy. Expert theoreticians typically deal in high-level proof sketches, guided by the shared understanding[8] that the author or reader of the proof sketch could fill in the missing details to produce a fully precise proof. This material is intended to instill in you the ability to produce fully precise proofs, but also the ability to write communicative proof sketches by eliding bureaucracy. The structured proof format can support both goals. The prior parts of this document focused on the former goal, but this section focuses on the latter.

Proofs and their Dependencies In the course of working, a typical proof of a theorem involves appeals to principles of set theory that derive from the axioms. In particular, we typically make use of the reasoning principles of our set notations without fully fleshing out the proofs. Furthermore, many proofs appeal to theorems that were proven earlier in the course of a development. Of interest is that almost always, those

[6] Technically we make use here of a "contracting" form of Conjunction Left, which discards the conjunction assumption if you only wish to keep the individual conjuncts as assumptions, as we do in the upcoming example.

[7] The name INTROS is adopted from the `intros` tactic of the Rocq/Coq mechanized proof assistant.

[8] possibly misguided unfortunately

proofs are of hypothetical verifications, but we still use them as though they were hypothetical uses, thanks to the admissibility of elimination rules on hypothetical verifications.

We also sometime abbreviate proofs, not showing all the details. In a well-formed structured proof, further details can be filled in by adding another proof level to flesh out an argument. The beauty of structured proofs is that one may express a proof at a variety of levels of abstraction. Hiding routine details can be better for communicating results to other humans, while fleshing out every last detail is sometimes necessary to convince a mechanized proof assistant that your proposition indeed holds. Leslie Lamport's advice for developing correct structured proofs is to add as many levels of proof as needed to convince yourself of the proof's correctness, and then add one level more. At times we will ask you to write a proof using a high-level of detail, primarily as a means to familiarize you with the structure of correct and precise reasoning. At other times, we will express a proof at a higher level of abstraction, where in principle a number of subsidiary proof steps are technically needed to flesh out the proof, but those steps may be relatively routine but perhaps tedious for a seasoned human prover.

In an ideal world, our proofs would be displayed in an interactive format, where details could be made visible or hidden by interacting with individual proof steps. Unfortunately we do not have such an interactive tool for structured proof.

Vapid Revisited In Ch. 1, we presented a formal model of the Vapid language, and presented, without explanation, a few structured proofs about it. If we reconsider those propositions and proofs, we will notice some surface differences from this chapter's description of structured proof. Those differences are mostly meant as conveniences for communication: each of those propositions and proofs could be refined to match this description.

First and easiest, we often omit proofs of the well-formedness judgments $\mathcal{E}\ \mathsf{set}$ and $\Phi\ \mathsf{prop}$, except when they involve some non-routine reasoning steps. Much of the time, we rely on the guarantees of our set builder notations to ensure that our sets are well-formed, and in turn that our propositions are well-formed. Only in cases where bespoke logical reasoning is needed do we explicitly state and prove well-formedness judgments. If we exhibited all such trivial proofs, then the essence of our logical arguments would be lost under a morass of administrative detail, or we would have to expand the rules of our system substantially in order to suppress them. Nonetheless a learnèd formal reasoner should always remember that these obligations exist, and look out for any nontrivial obligations that have been omitted.

Next, the propositions do not explicitly state which judgment they are proving about a proposition: they just say Φ instead of saying $\Phi\ \mathsf{verif}$ or $\Phi\ \mathsf{use}$. The reason for that is that we have demonstrated through our development of JZF that we can blur the distinction between the two without sacrificing logical rigour. We started out by only applying elimination rules to prove $\Phi\ \mathsf{use}$ judgments, and only applying introduction rules to prove $\Phi\ \mathsf{verif}$ judgments. This gives our *core* notion of formal proof a clear structure and a sense of proof-as-information-processing. But after introducing our judgments and rules, we proved that by the identity property, we can morally apply an introduction rule to a use judgment, and by the composition property, we can morally apply an elimination rule to a verif judgment.

For example, consider the following propositions and proofs:

Proposition 44.
Assume: *1.* $A\ \mathsf{use}$
2. $B\ \mathsf{use}$
Prove: $A \wedge B\ \mathsf{verif}$

1. $A\ \mathsf{verif}$ by identity
2. $B\ \mathsf{verif}$ by identity
3. $A \wedge B\ \mathsf{verif}$ by $\wedge$I with 1,2

Proposition 45.
Assume: *1.* $A\ \mathsf{verif}$
2. $B\ \mathsf{verif}$
Prove: $A \wedge B\ \mathsf{verif}$

1. $A \wedge B\ \mathsf{verif}$ by $\wedge$I with 1,2

In the proof of Prop. 44, the uses of the identity principle add no interesting *problem-specific* reasoning compared to the proof of Prop. 45. Nonetheless, we know that the underlying JZF proof is larger, needing extra steps to transform use into verif. In practice, we blur this administrative difference by writing the proposition and proof in the more concise form without committing to which judgment is at play, as though introduction rules could be applied just as easily to either judgment. Of course the proof can be appropriately elaborated in either case, so it justifies the argument for either judgment.

Proposition 46.
Assume: *1.* A
2. B
Prove: $A \wedge B$

1. $A \wedge B$ by $\wedge$I with 1,2

A similar relationship holds for elimination rules, ultimately thanks to the composition property. For example, consider the following proposition and proof:

Proposition 47.
Assume: *1.* $A \wedge B$ verif
Prove: A verif

1. Assume: 1. $A \wedge B$ use
   Prove: A use
   by $\wedge$E1 with 1:1
2. Q.E.D. by composition with 1 and 1

Prop. 47 proves that conjunction elimination is admissible for verif judgments, but hidden inside the proof is a use of conjunction elimination for use judgments. This strategy works for every elimination rule in JZF, so it makes for a nice abstraction to blur the distinction between uses of admissibility and direct elimination, as in the following:

Proposition 48.
Assume: *1.* $A \wedge B$
Prove: A

1. Q.E.D. by $\wedge$E1 with 1

In short, for convenience, compression, and clarity, we simply omit the judgment and overload our use of introduction and elimination rule names for both actual rules and their admissible counterparts.

An equivalent strategy taken in most presentations of similar logics is to simply have one judgment (possibly without explicit notation) from the start, such that both introduction and elimination rules apply to the same judgment. In that style, however, establishing many of the properties of such a logic in essence calls for introducing machinery that is equivalent to our two-judgment approach, but without making as explicit the computational "information-processing" perspective that we emphasize.

As a final, albeit perhaps obvious in retrospect, convenience, when proving a (judgment about a) proposition, our proofs can refer to previously proven (judgments about) propositions as though they had been proven in an earlier step of the proof. In essence we treat our entire development as one big structured proof, where distinct proposition numbers count as top-level structured proof "steps."

4.5.5 LaTeX Support

To typeset structured proofs, we use a modified version of a LaTeX style file developed by Leslie Lamport [Lamport, 2011]. Here we briefly discuss some of the features of the original, as well as some additions to the style for making local labels and references and hyperrefs.

Here is the LaTeX code, using pf2.sty for the proof from Fig. 4.3:

```
\begin{outerProof}
  \pf
  \step{1}{
      \sassume{
        \begin{pfenum}
        \item $Q\jset$
        \item \asmlabel{e} $\exists S.\,S \in Q \land A\juse$
        \end{pfenum}
      }
      \prove{$(\exists S.\,S \in Q)\land A\jverif$}
      by INTROS
    }
    \step{4.1}{
      \pick{}
      \begin{pfenum}
      \item \asmlabel{asm:s1} \pfnew{} $S_1 \in Q$
      \item \asmlabel{asm:a} $A\juse$
      \end{pfenum}
      by $\rexistsEs$ with \asmref{1}{e}, $\randL$
    }

    \step{4.2}{
      $\exists S.\,S \in Q\jverif$
      by $\rexistsI$ with \asmref{4.1}{asm:s1}($\jset$),
      \asmref{4.1}{asm:s1}($\juse$)
    }
    \step{4.4}{Q.E.D. by $\randI$ with \stepref{4.2}, \asmref{4.1}{asm:a}}
  \end{outerProof}
```

Some commands simply typeset terminology. `\pf` expands to Proof; `\pfnew` expands to New; `\pick` expands to Pick.

The `\step` command is used to typeset one step of a proof. The first argument is a label for the particular step. That label can be arbitrary: here the label corresponds to the step number from the larger proof in Fig. 4.2. Refactoring the proof does not require renumbering, but doing so can be helpful for editing (Lamport has developed a command-line application for updating step labels in LaTeX files). The `\stepref` command is used to refer to a step by label. Only steps that are in scope can be referenced: block structure of steps is enforced by the combination of `\step` and `\stepref`.

The `\pflevel` environment is used to create subproofs. For instance, here is a fragment of the LaTeXfor Fig. 4.2:

```
    \step{4}{
      \assume{
        \begin{pfenum}
        \item \asmlabel{e} $\exists S.\,S \in Q \land A\juse$
        \end{pfenum}
      }
      \prove{$(\exists S.\,S \in Q)\land A\jverif$}
    }
    \begin{pflevel}
      \step{4.1}{
        \sassume{
          \begin{pfenum}
          \item \asmlabel{asm:s1} $S_1\jset$
          \item \asmlabel{asm:x} $S_1 \in Q \land A\juse$
          \end{pfenum}
        }
        \prove{Q.E.D.}
      }
      \begin{pflevel}
        \step{4.1.1}{
          $(\exists S.\,S \in Q) \land A \jprop$
          \emph{(Proof Omitted)}
        }
        \step{4.1.2}{Q.E.D. by $\rexistsEs$ with \asmref{4}{e}, \stepref{4.1.1}}
      \end{pflevel}
```

When a proof step requires a complex proof, we use `\pflevel` to typeset this nested proof, with proper indentation and numbering.

`\suffices` is used to construct a SUFFICES proof step. The sufficient judgment is its argument, e.g.:

```
\begin{pflevel}
 \step{1}{\suffices{
   $A \jverif$
  }
 }
\end{pflevel}
```

multiple judgments can be accommodated using `\pfenum`.

`\assume` and `\prove` (or `\sassume` and `\prove`) are used together to assert a judgment that introduces additional assumptions to the context. Often the assumptions are presented as an enumerated list using the `\pfenum` form, which takes the proof structure into account when formatting.

`pf2.sty` is quite clever about enforcing block structure on step references: each use of `\stepref` can only refer to a step label that is in the surrounding scope. But often we want to reference not just individual steps, but individual enumerated assumptions that arise as a result of combining ASSUME (or SUFFICES-ASSUME) with `\pfenum`. Furthermore, we would like those labels to only be in scope for a single entire proof, so that the proof author need not invent new globally-unique reference names themselves. We introduce 4 new features to support this.

`\outerProof` behaves exactly like `\pflevel` except that it creates a new assumption label context, meaning that any assumption labels defined within the outerProof can only be referenced inside the outerProof itself. A better implementation of this feature would provide more block-structure control to ensure that only in-scope assumptions could be referenced, much like stepref, but since `\asmref` indicates the step as well as the assumption number, the failure of the step reference to properly resolve (appearing as question marks **??**) indicates a problem. Beware: `outerProof` blocks do not nest. Each one creates a new scope, so nesting will likely do the wrong thing. That's why it's called "outer". Use `\pflevel` for nesting proof levels. You *can* use `\pflevel` for an outer proof level, but local labels will likely behave inappropriately.

`\asmlabel` works like LaTeX's standard `\label`, but it creates a label whose scope only includes the surrounding `\outerProof` block. This is primarily meant for putting labels on assumptions so that they can be referenced precisely in proofs. But sometimes it is useful to reference enumerated proof statements, as when disambiguating multiple outstanding SUFFICES proof obligations.

`\asmref` takes two arguments: the label of the step that contains the referenced assumption, and the assumption for the label itself. Both are typeset with the format described earlier, but the entire label is wrapped in a hyperref that points to the asmlabel.

`\asmreftop` is to be used when referencing an assumption that appears outside of a proof step, as in the statement of a proposition. It is meant to be used with traditional `\label` labels, which exist outside of a proof. The difference from `\ref` is about presentation (including the colon to clarify that it is referencing an assumption) combined with support for hyperref (so the colon rests inside the hyperlink). It will not work properly if used to reference an `\asmlabel`. As an example of its use, the reference :3 points to the third assumption of Prop. 42. It is expected that in context there will be no need to textually reference the relevant Proposition since the overall proof must be associated with its corresponding proposition already.

When introducing a large number of assumptions, it can be nice to take advantage of horizontal space. This can be done using the `multicol` package. For example:

```
\begin{pflevel}
 \step{1}{
  \sassume{
   \begin{multicols}{4}
   \begin{pfenum}
   \item $S_1 \jset$
   \item $S_1 \in Q_1 \juse$
```

```
      \item $S_2 \jset$
      \item $S_2 \in Q_2 \juse$
      \item $A \juse$
      \item $B \juse$
      \item $C \juse$
      \item $D \juse$
      \end{pfenum}
      \end{multicols}
    }
    \vspace{-1em}
    \prove{
      $E\jverif$
    }
    by INTROS
  }
  \step{2}{\mbox{}\vdots}
\end{pflevel}
```

Simply wrapping the `pfenum` environment in a `multicols` environment uses 4 columns (instead of 1) to typeset the list. However, for some reason a gap is left before the PROVE. So we add `\vspace{-1em}` to re-compress. Since `multicols` spreads the columns evenly across the screen, you may need to provide more columns than enumerated items to make the spacing more compact if all of the items can fit on one line.

4.5.6 Additional Hypothesis Management

By exploiting metatheoretic properties of JZF, the structure of many of our proofs can be comfortably extended.

Using Previously Proven Propositions Often, when proving a proposition, we are wise to take a "divide-and-conquer" approach of breaking it up into multiple distinct propositions and then combining them. However, we may find ourselves applying a proposition in a context that has *more* assumptions in scope than the original proposition. In informal practice, We take this for granted, but our principle of composition, as stated, does not naturally allow it:

If $\Gamma \vdash \Phi_1\ \mathsf{verif}$ and $\Gamma, \Phi_1\ \mathsf{use} \vdash \Phi_2\ \mathsf{verif}$ then $\Gamma \vdash \Phi_2\ \mathsf{verif}$

The first judgment above corresponds to a previously proven proposition, and the second corresponds to a use of the first, but the assumptions in scope, Γ are *exactly the same* in both. However, given any previously proven proposition, JZF allows *weakening*:

If $\Gamma \vdash \mathcal{J}$ then $\Gamma, \mathcal{U} \vdash \mathcal{J}$

We can always add more assumptions without compromising the provability of the judgment (removing assumptions, on the other hand, can definitely compromise provability!). One might take this for granted, but some logics do not support weakening, especially logics where each assumption is treated as an ephemeral resource. If we were to use structured proofs to construct proofs in those systems, we would have to be more careful about how assumptions interact. Thankfully we don't. Such a system would also require additional features for assumption management when stating and proving hypothetical judgments within the scope of other assumptions. Our structured proofs can keep around all assumptions in scope, but a system without weakening may need to restrict the assumptions in scope for a moment, not simply add more assumptions.

4.6 Example: Return of the Empty Set!

In Ch. 3, we ended with a proof that the empty set is unique among all sets. Here we provide a structured proof of the same theorem, so as to compare and contrast between our new higher-level proof system and the underlying JZF system on which it depends.

Proposition 49 (The Empty Set)**.** $\exists! S . \forall Q . Q \notin S$

1. Suffices: 1. $\exists S . \forall Q . Q \notin S$
   2. $\forall S_1, S_2 . \, (\forall Q . Q \notin S_1) \wedge (\forall Q . Q \notin S_2) \Rightarrow S_1 = S_2$
   by $\wedge$I
2. $\exists S . \forall Q . Q \notin S$ by axiom of the empty set
3. $\forall S_1, S_2 . \, (\forall Q . Q \notin S_1) \wedge (\forall Q . Q \notin S_2) \Rightarrow S_1 = S_2$

3.1. Suffices Assume: 1. S_1 set 3. $\forall Q . Q \notin S_1$
2. S_2 set 4. $\forall Q . Q \notin S_2$
Prove: $S_1 = S_2$
by $\wedge$I with 2,3 and INTROS with 3

3.2. Suffices Assume: 1a. Q set
Prove: 1. Assume: $Q \in S_1$
Prove: $Q \in S_2$
2. Assume: $Q \in S_2$
Prove: $Q \in S_1$

3.2.1. $(\forall Q . \, Q \in S_1 \Leftrightarrow Q \in S_2) \Rightarrow S_1 = S_2$

3.2.1.1. $\forall S_1, S_2 . \, (\forall Q . \, Q \in S_1 \Leftrightarrow Q \in S_2) \Rightarrow S_1 = S_2$ by axiom of extensionality

3.2.1.2. Q.E.D. by $\forall$E with 3.2.1.1 and 3.1:1 and then $\forall$E with 3.1:2

3.2.2. $\forall Q . \, Q \in S_1 \Leftrightarrow Q \in S_2$

3.2.2.1. $Q \in S_1 \Rightarrow Q \in S_2$ by $\Rightarrow$I with 3.2:1

3.2.2.2. $Q \in S_2 \Rightarrow Q \in S_1$ by $\Rightarrow$I with 3.2:2

3.2.2.3. Q.E.D. by $\wedge$I with 3.2.2.1, 3.2.2.2 and $\forall$I with 3.2:4.6

3.2.3. Q.E.D. by $\Rightarrow$E with 3.2.1, 3.2.2

3.3. Assume: $Q \in S_1$
Prove: $Q \in S_2$ [Q.E.D. 3.2:1]

3.3.1. $Q \notin S_1$ by $\forall$E with 3.1:3

3.3.2. $\perp$ by $\Rightarrow$E with 3.3.1 and 3.3

3.3.3. Q.E.D. by $\perp$E with 3.3.2

3.4. Assume: $Q \in S_2$
Prove: $Q \in S_1$ [Q.E.D. 3.2:2]
Proof sketch: Analogous to 3.3

3.5. Q.E.D. by 3.3 and 3.4

One upside of this proof is that it fits on one page in full-size font without turning pages to landscape. The biggest savings comes from using lexical scope to determine which judgments are available in the context at any step.

One downside is that one might need to work a little harder to be sure that there are no missing or errant steps compared to derivation trees. But with some care, one can inspect such a proof line-by-line and confirm the high-level structure of the argument, as well as ensuring that the gaps appear in clear locations (marked with Proof sketch : or some such) and could be filled in if desired.

Bibliography

P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, 1977.

J. Avigad. Reliability of mathematical inference. *Synthese*, 2020. doi: 10.1007/s11229-019-02524-y. `https://doi.org/10.1007/s11229-019-02524-y`.

F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-45520-0.

J. Bagaria. Set theory. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. The Metaphysics Research Lab, winter 2014 edition, 2014. URL `http://plato.stanford.edu/archives/win2014/entries/set-theory/`.

R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3):50:1–50:39, May 2018. ISSN 0360-0300. doi: 10.1145/3182657. URL `http://doi.acm.org/10.1145/3182657`.

H. P. Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985. ISBN 978-0-444-86748-3.

A. Bauer. Proof of negation and proof by contradiction. Mathematics and Computation Blog, March 2010. `http://math.andrej.com/2010/03/29/proof-of-negation-and-proof-by-contradiction/`.

L. Crosilla. Set Theory: Constructive and Intuitionistic ZF. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2020 edition, 2020.

M. Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, 1991.

M. Felleisen and D. P. Friedman. A syntactic theory of sequential state. *Theoretical Computer Science*, 69(3):243–287, 1989. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(89)90069-8. URL `http://www.sciencedirect.com/science/article/pii/0304397589900698`.

M. Felleisen, R. B. Findler, and M. Flatt. *Semantics Engineering with PLT Redex*. MIT Press, 1st edition, 2009.

J. Ferreirós. The early development of set theory. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2019 edition, 2019. URL `https://plato.stanford.edu/archives/sum2019/entries/settheory-early/`.

G. Gentzen. Investigations into logical deduction. *American Philosophical Quarterly*, 1(4):288–306, 1964.

D. Grossman, G. Morrisett, T. Jim, M. Hicks, Y. Wang, and J. Cheney. Region-based memory management in cyclone. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, PLDI '02, pages 282–293, New York, NY, USA, 2002. ACM. ISBN 1-58113-463-0. doi: 10.1145/512529.512563. URL `http://doi.acm.org/10.1145/512529.512563`.

P. R. Halmos. *Naive Set Theory*. Springer-Verlag, first edition, Jan. 1960. ISBN 0387900926.
A classic introductory textbook on set theory.

P. R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2012. URL `http://www.cs.cmu.edu/%7Erwh/plbook/1sted-revised.pdf`.

D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124 (2):103–112, Feb. 1996. ISSN 0890-5401.

G. Kahn. Natural semantics. In *4th Annual Symposium on Theoretical Aspects of Computer Sciences on STACS 87*, pages 22–39, London, UK, UK, 1987. Springer-Verlag.

R. Krebbers. *The C standard formalized in Coq*. PhD thesis, Radboud University Nijmegen, December 2015. URL `https://robbertkrebbers.nl/thesis.html`.

C. Kuratowski. Sur la notion de l'ordre dans la théorie des ensembles. *Fundamenta Mathematicae*, 2(1): 161–171, 1921. doi: 10.4064/fm-2-1-161-171. `https://web.archive.org/web/20190429103938/http://matwbn.icm.edu.pl/ksiazki/fm/fm2/fm2122.pdf`.

L. Lamport. How to write a proof. *The American Mathematical Monthly*, 102(7):600–608, Aug 1995. ISSN 0002-9890. doi: 10.1080/00029890.1995.12004627. URL `https://doi.org/10.1080/00029890.1995.12004627`.

L. Lamport. The pf2 package, 2011. URL `https://lamport.azurewebsites.net/latex/pf2.pdf`.

L. Lamport. How to write a 21st century proof. *Journal of Fixed Point Theory and Applications*, 11(1): 43–63, Mar 2012. ISSN 1661-7746. doi: 10.1007/s11784-012-0071-6. URL `https://doi.org/10.1007/s11784-012-0071-6`.

P. J. Landin. The mechanical evaluation of expressions. *Comput. J.*, 6(4):308–320, 1964. doi: 10.1093/comjnl/6.4.308. URL `https://doi.org/10.1093/comjnl/6.4.308`.

X. Leroy and H. Grall. Coinductive big-step operational semantics. *Inf. Comput.*, 207(2):284–304, Feb. 2009. ISSN 0890-5401.

W. Lovas and K. Crary. Structural normalization for classical natural deduction. https://www.cs.cmu.edu/ wlovas/papers/clnorm.pdf, December 2006.

P. Maddy. Believing the axioms. I. *The Journal of Symbolic Logic*, 53(02):481–511, 1988.
An interesting (though complicated) analysis of why set theorists believe in their axioms.

P. Maddy. *Defending the Axioms: On the Philosophical Foundations of Set Theory*. Oxford University Press, 01 2011. ISBN 9780199596188. doi: 10.1093/acprof:oso/9780199596188.001.0001. URL `https://doi.org/10.1093/acprof:oso/9780199596188.001.0001`.

P. Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Commun. ACM*, 3(4):184–195, 1960.

J. McCarthy. Towards a mathematical science of computation. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*, pages 21–28. North-Holland, 1962.

J. H. J. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, Feb. 1969. URL `http://hdl.handle.net/1721.1/64850`.

F. Pfenning. Church and curry: Combining intrinsic and extrinsic typing. In C. Benzmueller, C. E. Brown, and J. Siekmann, editors, *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on His 70th Birthday*, volume 17 of *Mathematical Logic and Foundations*. College Publications, December 2008.

F. Pfenning. Constructive logic: Course notes. `https://www.cs.cmu.edu/~fp/courses.html#15-317`, 2009.

A. M. Pitts. *Nominal sets: Names and symmetry in computer science*. Cambridge University Press, 2013.

G. D. Plotkin. The origins of structural operational semantics. *J. Log. Algebr. Program.*, 60-61:3–15, 2004a. doi: 10.1016/j.jlap.2004.03.009. URL `http://dx.doi.org/10.1016/j.jlap.2004.03.009`.

G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004b.

B. Popik. ”pull yourself up by your bootstraps”. Weblog entry, September 2012. `https://www.barrypopik.com/index.php/new_york_city/entry/pull_yourself_up_by_your_bootstraps/`.

K. Rashidi. Eliminability of definite descriptions. Bachelor’s thesis, University of British Columbia, 2025. URL `http://hdl.handle.net/2429/91182`.

E. Reck and G. Schiemer. Structuralism in the Philosophy of Mathematics. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2020 edition, 2020.

B. Russell. On denoting. *Mind*, 56(14):479–493, 1905.

B. Russell. *Introduction to Mathematical Philosophy*. George Allen & Unwin, Ltd., London, May 1919. ISBN 9781003308881.

D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4): 15:1–15:41, May 2009. ISSN 0164-0925.

W. Sieg and S. Cittadini. Normal natural deduction proofs (in non-classical logics). In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, pages 169–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32254-2. doi: 10.1007/978-3-540-32254-2_11. URL `https://doi.org/10.1007/978-3-540-32254-2_11`.

W. Sieg and D. Schlimm. Dedekind’s analysis of number: Systems and axioms. *Synthese*, 147(1):121–170, Oct 2005.

R. J. Simmons. Structural focalization. *ACM Trans. Comput. Logic*, 15(3), Sept. 2014. ISSN 1529-3785. doi: 10.1145/2629678. URL `https://doi.org/10.1145/2629678`.

J. Spolsky. The law of leaky abstractions. Joel on Software Blog, November 2002. `https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/`.

G. L. Steele. Debunking the “expensive procedure call”” myth or, procedure call implementations considered harmful or, lamdba: The ultimate goto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1977. URL `http://dspace.mit.edu/handle/1721.1/5753`.

S. Stenlund. Descriptions in intuitionistic logic. In S. Kanger, editor, *Proceedings of the Third Scandinavian Logic Symposium*, volume 82 of *Studies in Logic and the Foundations of Mathematics*, pages 197 – 212. Elsevier, 1975. URL `http://www.sciencedirect.com/science/article/pii/S0049237X08707328`.

M. Tiles. Book Review: Stephen Pollard. Philosophical Introduction to Set Theory. *Notre Dame Journal of Formal Logic*, 32(1):161–166, 1990.
A brief introduction to the philosophical issues underlying set theory as a foundation for mathematics.

C. Urban and M. Norrish. A formal treatment of the Barendregt variable convention in rule inductions. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding*, MERLIN '05, page 25–32, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930728. doi: 10.1145/1088454.1088458. URL `https://doi.org/10.1145/1088454.1088458`.

C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *Proceedings of the 21st International Conference on Automated Deduction: Automated Deduction*, CADE-21, page 35–50, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 9783540735946. doi: 10.1007/978-3-540-73595-3_4. URL `https://doi.org/10.1007/978-3-540-73595-3_4`.

D. van Dalen. *Logic and structure (3. ed.).* Universitext. Springer, 1994. ISBN 978-3-540-57839-0.

A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, Nov. 1994.

B. Zimmer. figurative "bootstraps". email to linguistlist mailing list, August 2005. `http://listserv.linguistlist.org/pipermail/ads-l/2005-August/052756.html`.