A TLA$^+$ Proof System

Kaustuv Chaudhuri, INRIA
Damien Doligez, INRIA
Leslie Lamport, Microsoft Research
Stephan Merz, INRIA & Loria

Abstract

We describe an extension to the TLA$^+$ specification language with constructs for writing proofs and a proof environment, called the Proof Manager (PM), to checks those proofs. The language and the PM support the incremental development and checking of hierarchically structured proofs. The PM translates a proof into a set of independent proof obligations and calls upon a collection of back-end provers to verify them. Different provers can be used to verify different obligations. The currently supported back-ends are the tableau prover Zenon and Isabelle/TLA$^+$, an axiomatisation of TLA$^+$ in Isabelle/Pure. The proof obligations for a complete TLA^{+2} proof can also be used to certify the theorem in Isabelle/TLA$^+$.

1 Introduction

TLA$^+$ is a language for specifying the behavior of concurrent and distributed systems and asserting properties of those systems [11]. However, it provides no way to write proofs of those properties. We have designed an extended version of the language that allows writing proofs, and we have begun implementing a system centered around a *Proof Manager* (PM) that invokes existing automated and interactive proof systems to check those proofs. For now, the new version of TLA$^+$ is called TLA^{+2} to distinguish it from the current one. We describe here the TLA^{+2} proof constructs and the current state of the proof system.

The primary goal of TLA^{+2} and the proof system is the mechanical verification of systems specifications. The proof system must not only support the modal and temporal aspects of TLA needed to reason about system properties, but must also support ordinary mathematical reasoning in the underlying logic. Proofs in TLA^{+2} are natural deduction proofs written in a hierarchical style that we have found to be good for ordinary mathematics [9] and crucial for managing the complexity of correctness proofs of systems [6].

The PM computes proof obligations that establish the correctness of the proof and sends them to one or more back-end provers to be verified. Currently, the back-end provers are Isabelle/TLA$^+$, a faithful axiomatization of TLA$^+$ in Isabelle/Pure, and Zenon [2], a tableau prover for classical first-order logic with equality. The PM first sends a proof obligation to Zenon. If Zenon succeeds, it produces an Isar script that the PM sends to Isabelle to check. Otherwise, the PM outputs an Isar script that uses one of Isabelle's automated tactics. In both cases, the obligations are certified by Isabelle/TLA$^+$. The system architecture easily accommodates other back-end provers; if these are proof-producing, then we can use their proofs to certify the obligations in Isabelle/TLA$^+$, resulting in high confidence in the overall correctness of the proof.

The TLA^{+2} proof constructs are described in Section 2. Section 3 describes the proof obligations generated by the PM, and Section 4 describes how the PM uses Zenon and Isabelle to verify them. The conclusion summarizes what we have done and not yet done and briefly discusses related work.

2 TLA$^+$ and its Proof Language

2.1 TLA

The TLA$^+$ language is based on the Temporal Logic of Actions (TLA) [10], a linear-time temporal logic. The rigid variables of TLA are called *constants* and the flexible variables are called simply *variables*. TLA assumes an underlying ordinary (non-modal) logic for constructing expressions. Operators of that logic are called *constant* operators. A *state function* is an expression built from constant operators and

TLA constants and variables. The elementary (non-temporal) formulas of TLA are *actions*, which are formulas built with constant operators, constants, variables, and expressions of the form f', where f is a state function. (TLA also has an operator that is used in expressing fairness, but we ignore it for brevity.) An action is interpreted as a predicate on pairs of states that describes a set of possible state transitions, where state functions refer to the starting state and primed state functions refer to the ending state. Because priming distributes over constant operators and because c' is equal to c for any constant c, an action can be reduced to a formula built from constant operators, constants, variables, and primed variables.

TLA is practical for describing systems because all the complexity of a specification is in the action formulas. Temporal operators are essentially used only to assert liveness properties, including fairness of system actions. Most of the work in a TLA proof is in proving action formulas; temporal reasoning occurs only in proving liveness properties and is limited to propositional temporal logic and to applying a handful of proof rules whose main premises are action formulas. Because temporal reasoning is such a small part of TLA proofs, we have deferred its implementation. The PM now handles only action formulas. We have enough experience mechanizing TLA's temporal reasoning [4] to be fairly confident that it will not be hard to extend the PM to support it.

A formula built from constant operators, constants, variables, and primed variables is valid iff it is a valid formula of the underlying logic when constants, variables, and primed variables are treated as distinct variables of the logic—that is, if v and v' are considered to be two distinct variables of the underlying logic, for any TLA variable v. Since any action formula is reducible to such a formula, action reasoning is immediately reducible to reasoning in the underlying logic. We therefore ignore variables and priming here and consider only constant formulas.

2.2 TLA+

The TLA+ language adds the following to the TLA logic:

- An underlying logic that is essentially ZFC set theory plus classical untyped first-order logic with Hilbert's ε [13]. The major difference between this underlying logic and traditional ZFC is that functions are defined axiomatically rather than being represented as sets of ordered pairs.
- A mechanism for defining operators, where a user-defined operator is essentially a macro that is expanded syntactically. (TLA+ permits recursive function definitions, but they are translated to ordinary definitions using Hilbert's ε.)
- Modules, where one module can import definitions and theorems from other modules. A module is parameterized by its declared variables and constants, and it may be instantiated in another module by substituting expressions for its parameters. The combination of substitution and the operator introduces some complications, but space limitations prevent us from discussing them, so we largely ignore modules in this paper.

TLA+ has been extensively documented [11]. Since we are concerned only with reasoning about its underlying logic, which is a very familiar one, we do not bother to describe TLA+ in any detail. All of its nonstandard notation that appears in our examples is explained.

2.3 The Proof Language

The major new feature of TLA+2 is its proof language. (For reasons having nothing to do with proofs, TLA+2 also introduces recursive operator definitions, which we ignore here for brevity.) We describe the basic proof language, omitting a few constructs that concern aspects such as module instantiation that we are not discussing. TLA+2 also adds constructs for naming subexpressions of a definition or theorem, which is important in practice for writing proofs but is orthogonal to the concerns of this paper.

The goal of the language is to make proofs easy to read and write for someone with no knowledge of how the proofs are being checked. This leads to a mostly declarative language, built around the uses and proofs of assertions rather than around the application of proof-search tactics. It is therefore more akin to Isabelle/Isar [17] than to more operational interactive languages such as Coq's Vernacular [16]. Nevertheless, the proof language does include a few operational constructs that can eliminate the repetition of common idioms, albeit with some loss of perspicuity.

At any point in a TLA+ proof, there is a current obligation that is to be proved. The obligation contains a *context* of known facts, definitions, and declarations, and a *goal*. The obligation claims that the goal is logically entailed by the context. Some of the facts and definitions in the context are marked (explicitly or implicitly) as *usable* for reasoning, while the remaining facts and definitions are *hidden*.

Proofs are structured hierarchically. The leaf (lowest-level) proof asserts that the current goal follows easily from the usable facts and definitions. The leaf proof

$$e_1,\ldots,e_m \qquad o_1,\ldots,o_n$$

asserts that the current goal follows easily from the usable facts and definitions together with (i) the facts e_i that must themselves follow easily from the context and (ii) the known definitions of o_j. Whether a goal follows easily from definitions and facts depends on who is trying to prove it. For each leaf proof, the PM sends the corresponding *leaf obligation* to the back-end provers, so in practice "follows easily" means that a back-end prover can prove it. A non-leaf proof is a sequence of *steps*, each consisting of a begin-step token and a proof construct. For some constructs (including a simple assertion of a proposition) the step takes a subproof, which may be omitted. The final step in the sequence simply asserts the current goal, which is represented by the token . A begin-step token is either a *level token* of the form $\langle n\rangle$ or a *label* of the form $\langle n\rangle l$, where n is a level number that is the same for all steps of this non-leaf proof, and l is an arbitrary name. The hierarchical structure is deduced from the level numbers of the begin-step tokens, a higher level number beginning a subproof.

Some steps make declarations or definitions or change the current goal and do not require a proof. Other steps make assertions that become the current goals for their proofs. An omitted proof (or one consisting of the token) is considered to be a leaf proof that instructs the assertion to be accepted as true. Of course, the proof is then incomplete. From a logical point of view, an omitted step is the same as an additional assumption added to the theorem; from a practical point of view, it doesn't have to be lifted from its context and stated at the start. Omitted steps are intended to be used only in the intermediate stages of writing a proof.

Following a step that makes an assertion (and the step's proof), until the end of the current proof (after the step), the contexts contain that assertion in their sets of known facts. The assertion is marked usable iff the begin-step token is a level token; otherwise it can be referred to by its label in a proof or made usable with a step.

The hierarchical structure of proofs not only aids in reading the finished proof but is also quite useful in incrementally writing proofs. The steps of a non-leaf proof are first written with all proofs but that of the step omitted. After checking the proof of the step, the proofs omitted for other steps in this or earlier levels are written in any order. When writing the proof, one may discover facts that are needed in the proofs of multiple steps. Such a fact is then added to the proof as an earlier step, or added at a higher level. It can also be removed from the proof of the theorem and proved separately as a lemma. However, the hierarchical proof language encourages facts relevant only for a particular proof to be kept within the proof, making the proof's structure easier to see and simplifying maintenance of the proof. For correctness proofs of systems, the first few levels of the hierarchy are generally determined by the structure of the formula to be proved—for example, the proof that a formula implies a conjunction usually consists of steps asserting that it implies each conjunct.

As an example, we incrementally construct a hierarchical proof of Cantor's theorem, which states that there is no surjective function from a set to its powerset. It is written in TLA+ as:

$$\forall S : \forall f \in [S \rightarrow \quad S] : \exists A \in \quad S : \forall x \in S : f[x] \neq A$$

where function application is written using square brackets, S is the powerset of S, and $[S \rightarrow T]$ is the set of functions from S to T.

The statement of the theorem is the current goal for its top-level proof. A goal of the form $\forall v : e$ is proved by introducing a generic constant and proving the formula obtained by substituting it for the bound identifier. We express this as follows, using the / construct of TLA+2:

$\forall S : \forall f \in [S \rightarrow \quad S] : \exists A \in \quad S : \forall x \in S : f[x] \neq A$
⟨1⟩1. S,
$f \in [S \rightarrow \quad S]$
$\exists A \in \quad S : \forall x \in S : f[x] \neq A$
⟨1⟩2. ⟨1⟩1

Although we could have used labels such as ⟨1⟩*one* and ⟨1⟩*last* instead of ⟨1⟩1 and ⟨1⟩2, we have found that proofs are easier to read when steps at the same level are labeled with consecutive numbers. One typically starts using consecutive step numbers and then uses labels like ⟨3⟩2a for inserting additional steps. When the proof is finished, steps are renumbered consecutively. (A planned user interface will automate this renumbering.)

Step ⟨1⟩1 asserts that for any constants S and f with $f \in [S \rightarrow \quad S]$, the proposition to the right of the is true. More precisely, the current context for the (as yet unwritten) proof of ⟨1⟩1 contains the declarations of S and f and the usable fact $f \in [S \rightarrow \quad S]$, and the assertion is its goal. The step states that the original goal (the theorem) follows from the assertion in step ⟨1⟩1.

We tell the PM to check this (incomplete) proof, which it does by having the back-end provers verify the proof obligation for the step. The verification succeeds, and we now continue by writing the proof of ⟨1⟩1. (Had the verification failed because ⟨1⟩1 did not imply the current goal, we would have caught the error before attempting to prove ⟨1⟩1, which we expect to be harder to do.)

We optimistically start with the proof , but it is too hard for the back-end to prove, and the PM reports a timeout. Often this means that a necessary fact or definition in the context is hidden and we merely have to make it usable with a step or a proof. In this case we have no such hidden assumptions, so we must refine the goal into simpler goals with a non-leaf proof. We let this proof have level 2 (we can use any level greater than 1). Since the goal itself is existentially quantified, we must supply a witness. In this case, the witness is the classic diagonal set, which we call T.

⟨1⟩1. S,
$f \in [S \rightarrow \quad S]$
$\exists A \in \quad S : \forall x \in S : f[x] \neq A$
⟨2⟩1. $T \triangleq \{z \in S : z \notin f[z]\}$
⟨2⟩2. $\forall x \in S : f[x] \neq T$
⟨2⟩3. ⟨2⟩2

Because definitions made within a proof are usable by default, the definition of T is usable in the proofs of ⟨2⟩2 and ⟨2⟩3. Once again, the proof of the step is automatically verified, so all that remains is to prove ⟨2⟩2. (The step requires no proof.)

The system accepts as the proof of ⟨2⟩2 because the only difficulty in the proof of ⟨1⟩1 is finding the witness. However, suppose we want to add another level of proof for the benefit of a human reader. The universal quantification is proved as above, by introducing a fresh constant:

$\langle 2\rangle 2.\ \forall x \in S : f[x] \neq T$
$\quad\langle 3\rangle 1.\quad x \in S \quad f[x] \neq T$
$\quad\langle 3\rangle 2.\quad \langle 3\rangle 1$

Naturally, the step is verified. Although the system accepts as the proof of ⟨3⟩1 (remember that it could verify ⟨2⟩2 by itself), we can provide more detail with yet another level of proof. We write this proof the way it would seem natural to a person—by breaking it into two cases:

$\langle 3\rangle 1.\quad x \in S \quad f[x] \neq T$
$\quad\langle 4\rangle 1.\quad x \in T$
$\quad\langle 4\rangle 2.\quad x \notin T$
$\quad\langle 4\rangle 3.\quad \langle 4\rangle 1, \langle 4\rangle 2$

The (omitted) proof of the statement ⟨4⟩1 has as its goal $f[x] \neq T$ and has the additional usable fact $x \in T$ in its context.

We continue refining the proof in this way, stopping with an or proof when a goal is obvious enough for the back-end prover or for a human reader, depending on who the proof is being written for. A statement can guide the prover or the human reader by listing helpful obvious consequences of known facts. For example, the proof of ⟨4⟩1 might be $x \notin f[x]$. The proof is now finished: it contains no omitted sub-proofs. For reference, the complete text of the proof is given in Appendix B.

Our experience writing hand proofs makes us expect that proofs of systems could be ten or more levels deep, with the first several levels dictated by the structure of the property to be proved. Our method of numbering steps makes such proofs manageable, and we are not aware of any good alternative.

This example illustrates how the proof language supports the hierarchical, non-linear, and incremental development of proofs. The proof writer can work on the most problematic unproved steps first, leaving the easier ones for later. Finding that a step cannot be proved (for example, because it is invalid) may require changing other steps, making proofs of those other steps wasted effort. We intend to provide an interface to the PM that will make it easy for the user to indicate which proofs should be checked and will avoid unnecessarily rechecking proofs.

The example also shows how already-proved facts are generally not made usable, but are invoked explicitly in proofs. Global definitions are also hidden by default and the user must explicitly make them usable. This makes proofs easier to read by telling the reader what facts and definitions are being used to prove each step. It also helps constrain the search space for an automated back-end prover, leading to more efficient verification. Facts and definitions can be switched between usable and hidden by and steps, which have the same syntax as . As noted above, omitting the label from a step's starting token (for example, writing ⟨4⟩ instead of ⟨4⟩2) makes the fact it asserts usable. This might be done for compactness at the lowest levels of a proof.

The example also indicates how the current proof obligation at every step of the proof is clear, having been written explicitly in a parent assertion. This clear structure comes at the cost of introducing many levels of proof, which can be inconvenient. One way of avoiding these extra levels is by using an assertion of the form A, which asserts that proving A proves the current goal, and makes A the new current goal in subsequent steps. In our example proof, one level in the proof of step ⟨2⟩2 can be eliminated by writing the proof as:

$\langle 2\rangle 2.\ \forall x \in S : f[x] \neq T$
$\quad\langle 3\rangle 1.\quad x \in S \quad f[x] \neq T$

$\quad\langle 3\rangle 2.\quad x \in T$
$\quad\langle 3\rangle 3.\quad x \notin T$
$\quad\langle 3\rangle 4.\quad \langle 3\rangle 2, \langle 3\rangle 3$

where the proofs of the steps are the same as before. The statement changes the current goal of the level-3 proof to $f[x] \neq T$ after adding a declaration of x and the usable fact $x \in S$ to the context. This way of proving a universally quantified formula is sufficiently common that TLA^{+2} provides a construct that allows the assertion $\langle 3 \rangle 1$ and its proof to be written $x \in S$.

There is a similar construct, $f \in S$ for proving an existentially quantified goal $\exists x \in S : e$, which changes the goal to $e[x := f]$. For implicational goals $e \Rightarrow f$, the construct e changes the goal to f. No other constructs in the TLA^{+2} proof language change the form of the current goal. We advise that these constructs be used only at the lowest levels of the proof, since the new goal they create must be derived instead of being available textually in a parent assertion. (As a check and an aid to the reader, one can at any point insert a redundant step that simply asserts the current goal.)

The final TLA^{+2} proof construct is $x : e$, which introduces a new symbol x that satisfies e. The goal of the proof of this step is $\exists x : e$, and it changes the context of subsequent steps by adding a declaration of x and the fact e. A more formal summary of the language appears in Appendix A.

The semantics of a TLA^{+2} proof is independent of any back-end prover. Different provers will have different notions of what "follows easily", so an proof may be verified by one prover and not another. In practice, many provers such as Isabelle must be directed to use decision procedures or special tactics to prove some assertions. For this purpose, special standard modules will contain dummy theorems for giving directives to the PM. Using such a theorem (with a step or proof) will cause the PM not to use it as a fact, but instead to generate special directives for back-end provers. It could even cause the PM to use a different back-end prover. (If possible, the dummy theorem will assert a true fact that suggests the purpose of the directive.) For instance, using the theorem *Arithmetic* might be interpreted as an instruction to use a decision procedure for integers. We hope that almost all uses of this feature will leave the TLA^{+2} proof independent of the back-end provers. The proof will not have to be changed if the PM is reconfigured to replace one decision procedure with a different one.

3 Proof Obligations

The PM generates a separate *proof obligation* for each leaf proof and orchestrates the back-end provers to verify these obligations. Each obligation is independent and can be proved individually. If the system cannot verify an obligation within a reasonable amount of time, the PM reports a failure. The user must then determine if it failed because it depends on hidden facts or definitions, or if the goal is too complex and needs to be refined with another level of proof. (Hiding facts or definitions might also help to constrain the search space of the back-end provers.)

When the back-end provers fail to find a proof, the user will know which obligation failed—that is, she will be told the obligation's usable context and goal and the leaf proof from which it was generated. We do not yet know if this will be sufficient in practice or if the PM will need to provide the user with more information about why an obligation failed. For example, many SAT and SMT solvers produce counterexamples for an unprovable formula that can provide useful debugging information.

The PM will also mediate the *certification* of the TLA^{+2} theorem in a formal axiomatization of TLA^{+2} in a trusted logical framework, which in the current design is Isabelle/TLA$^+$ (described in Section 4.2). Although the PM is designed generically and can support other similar frameworks, for the rest of this paper we will limit our attention to Isabelle/TLA$^+$. Assuming that Isabelle/TLA$^+$ is sound, once it has certified a theorem we know that an error is possible only if the PM incorrectly translated the statement of the theorem into Isabelle/TLA$^+$.

After certifying the proof obligations generated for the leaf proofs, called the *leaf obligations*, certification of the theorem itself is achieved in two steps. First, the PM generates a *structure lemma* (and its Isabelle/TLA$^+$ proof) that states simply that the collection of leaf obligations implies the theorem.

Then, the PM generates a proof of the theorem using the already-certified obligations and structure lemma. If Isabelle accepts that proof, we are assured that the translated version of the theorem is true in Isabelle/TLA+, regardless of any errors made by the PM.

Of course, we expect the PM to be correct. We now explain why it should be by describing how it generates the leaf obligations from the proof of a theorem. (Remember that we are considering only TLA+2 formulas with no temporal operators.) Formally, a theorem in TLA+2 represents a closed proof obligation in the TLA+2 meta-logic of the form $(\Gamma \Vdash e)$, where Γ is a *context* containing all the declarations, definitions, facts (previous assumptions or theorems) and the assumptions introduced in the theorem using an clause (if present), and e is a TLA+2 formula that is the *goal* of the theorem.

A closed obligation $(\Gamma \Vdash e)$ is *true* if e is entailed by Γ in the formal semantics of TLA+ [11]. It is said to be *provable* if we have a proof of e from Γ in Isabelle/TLA+. Because we assume Isabelle/TLA+ to be sound, we consider any provable obligation to be true. A *claim* is a sentence of the form $\pi : (\Gamma \Vdash e)$, where π is a TLA+2 proof. This claim represents the verification task that π is a proof of the proof obligation $(\Gamma \Vdash e)$. The PM generates the leaf obligations of a claim by recursively traversing its proof, using its structure to refine the obligation of the claim. For a non-leaf proof, each proof step modifies the context or the goal of its obligation to produce an obligation for its following step, and the final step proves the final form of the obligation. More precisely, every step defines a *transformation*, written $\sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)$, which states that the *input* obligation $(\Gamma \Vdash e)$ is *refined* to the obligation $(\Delta \Vdash f)$ by the step $\sigma.\tau$. A step is said to be *meaningful* if the input obligation matches the form of the step. (An example of a meaningless claim is one that involves a step whose input obligation does not have a universally quantified goal.) A claim is meaningful if every step in it is meaningful.

The recursive generation of leaf obligations for meaningful claims and transformations is specified using inference rules, with the interpretation that the leaf obligations generated for the claim or transformation at the conclusion of a rule is the union of those generated by the claims and transformations in the premises of the rule. For example, the following rule is applied to generate the leaf obligations for a claim $\pi : (\Gamma \Vdash e)$ when π is a sequence of n steps, for $n > 1$.

$$\frac{\sigma_1.\tau_1 : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f) \qquad \sigma_2.\tau_2 \ \cdots \ \sigma_n.\tau_n : (\Delta \Vdash f)}{\sigma_1.\tau_1 \ \ \sigma_2.\tau_2 \ \cdots \ \sigma_n.\tau_n : (\Gamma \Vdash e)}$$

The leaf obligations of the claim in the conclusion are the union of those of the claim and transformation in the premises. As an example of leaf obligations generated by a transformation, here is a rule for the step $\sigma.\tau$ where σ is the begin-step level token $\langle n \rangle$ and τ is the proposition p with proof π.

$$\frac{\pi : (\Gamma, [\neg e] \Vdash p)}{\langle n \rangle.\ p \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, p \Vdash e)}$$

The rule concludes that the refinement in this step is to add p to the context of the obligation, assuming that the sub-proof π is able to establish it. The leaf obligations generated by this transformation are the same as those of the claim in the premise of the rule. The goal e is negated and added to the context as a hidden fact (the square brackets indicate hiding). We can use $\neg e$ in a proof or statement, and doing so can simplify subproofs. (Because we are using classical logic, it is sound to add $\neg e$ to the known facts in this way.) The full set of such rules for every construct in the TLA+2 proof language is given in appendix A.

A claim is said to be *complete* it its proof contains no omitted subproofs. Starting from a complete meaningful claim, the PM first generates its leaf obligations and *filters* the hidden assumptions from their contexts. (Filtration amounts to deleting hidden facts and replacing hidden operator definitions with declarations.) The PM then asks the back-end provers to find proofs of the filtered obligations, which are used to certify the obligations in Isabelle/TLA+. The PM next writes an Isar proof of the obligation of the complete meaningful claim that uses its certified filtered leaf obligations. The following meta-theorem (proved in Appendix A.4) ensures that the PM can do this for all complete meaningful claims.

Theorem 1 (Structural Soundness Theorem). *If $\pi : (\Gamma \Vdash e)$ is a complete meaningful claim and every leaf obligation it generates is provable after filtering hidden assumptions , then $(\Gamma \Vdash e)$ is provable.*

Isabelle/TLA$^+$ then uses this proof to certify the obligation of the claim. From the assumptions that the Isabelle/TLA$^+$ axiomatization is faithful to the semantics of TLA^{+2} and that the embedding of TLA^{+2} into Isabelle/TLA$^+$ is sound, it follows that the obligation is true.

4 Verifying Proof Obligations

Once the PM generates the leaf obligations, it must send them to the back-end provers. The one non-obvious part of doing this is deciding whether definitions should be expanded by the PM or by the prover. This is discussed in Section 4.1. We then describe the state of our two current back-end provers, Isabelle/TLA$^+$ and Zenon.

4.1 Expanding Definitions

Expansion of usable definitions cannot be left entirely to the back-end prover. The PM itself must do it for two reasons:

- It must check that the current goal has the right form for a , , or step to be meaningful, and this can require expanding definitions.
- The encoding of TLA$^+$ in the back-end prover's logic would be unsound if a modal operator like prime ($'$) were encoded as a non-modal operator. Hence, encoding a definition like $O(x) \triangleq x'$ as an ordinary definition in the prover's logic would be unsound. All instances of such operators must be removed by expanding their definitions before a leaf obligation is sent to the back-end prover. Such operator definitions seldom occur in actual TLA$^+$ specifications, but the PM must be able to deal with them.

Another reason for the PM to handle definition expansion is that the Isabelle/TLA$^+$ object logic does not provide a direct encoding of definitions made within proofs. We plan to reduce the amount of trusted code in the PM by lambda-lifting all usable definitions out of each leaf obligation and introducing explicit operator definitions using Isabelle's meta equality ($\equiv$). These definitions will be expanded before interacting with Isabelle.

4.2 Isabelle/TLA$^+$

The core of TLA^{+2} is being encoded as a new object logic Isabelle/TLA$^+$ in the proof assistant Isabelle [14]. One of Isabelle's distinctive features that similar proof assistants such as Coq [16] or HOL [7, 8] lack is genericity with respect to different logics. The base system Isabelle/Pure provides the trusted kernel and a framework in which the syntax and proof rules of object logics can be defined. We have chosen to encode TLA^{+2} as a separate object logic rather than add it on top of one of the existing logics (such as ZF or HOL). This simplifies the translation and makes it easier to interpret the error messages when Isabelle fails to prove obligations. A strongly typed logic such as HOL would have been unsuitable for representing TLA^{+2}, which is untyped. Isabelle/ZF might seem like a natural choice, but differences between the way it and TLA$^+$ define functions and tuples would have made the encoding awkward and would have prevented us from reusing existing theories. Fortunately, the genericity of Isabelle helped us not only to define the new logic, but also to instantiate the main automated proof methods, including rewriting, resolution- and tableau provers, and case-based and inductive reasoning. Adding support for more specialized reasoning tools such as proof-producing SAT solvers [5] or SMT solvers such as haRVey [3] will be similarly helped by existing generic interfaces.

The current encoding supports only a core subset of TLA+2, including propositional and first-order logic, elementary set theory, functions, and the construction of natural numbers. Support for arithmetic, strings, tuples, sequences, and records is now being added; support for the modal part of TLA+2 (variables, priming, and temporal logic) will be added later. Nevertheless, the existing fragment can already be used to test the interaction of the PM with Isabelle and other back-end provers. As explained above, Isabelle/TLA+ is used both as a back-end prover and to check proof scripts produced by other back-end provers such as Zenon. If it turns out to be necessary, we will enable the user to invoke one of Isabelle's automated proof methods (such as `auto` or `blast`) by using a dummy theorem, as explained at the end of Section 2.3. If the method succeeds, one again obtains an Isabelle theorem. Of course, Isabelle/TLA+ can also be used independently of the PM, which is helpful when debugging tactics.

4.3 Zenon

Zenon [2] is a tableau prover for classical first-order logic with equality that was initially designed to output formal proofs checkable by Coq [16]. Zenon outputs proofs in an automatically-checkable format and it is easily extensible with new inference rules. One of its design goals is predictability in solving simple problems, rather than high performance in solving some hard problems. These characteristics make it well-suited to our needs.

We have extended Zenon to output Isar proof scripts for Isabelle/TLA+ theorems, and the PM uses Zenon as a back-end prover, shipping the proofs it produces to Isabelle to certify the obligation. We have also extended Zenon with direct support for the TLA+2 logic, including definitions and rules about sets and functions. Adding support in the form of rules (instead of axioms) is necessary because some rules are not expressible as first-order axioms, notably the rules about the set constructs:

$$\frac{e \in S \qquad P[x := e]}{e \in \{x \in S : P\}}\ \mathit{subsetOf} \qquad\qquad \frac{\exists y \in S : e = d[x := y]}{e \in \{d : x \in S\}}\ \mathit{setOfAll}$$

Even for the rules that are expressible as first-order axioms, adding them as rules makes the proof search procedure much more efficient in practice. The most important example is extensionality: when set extensionality and function extensionality are added as axioms, they apply to every equality deduced by the system, and pollute the search space with large numbers of irrelevant formulas. By adding them as rules instead, we can use heuristics to apply them only in cases where they have some chance of being useful.

Adding support for arithmetic, strings, tuples, sequences, and records will be done in parallel with the corresponding work on Isabelle/TLA+, to ensure that Zenon will produce proof scripts that Isabelle/TLA+ will be able to check. Temporal logic will be added later. We also plan to interface Zenon with Isabelle, so it can be called by a special Isabelle tactic the same way other tools are. This will simplify the PM by giving it a uniform interface to the back-end provers. It will also allow using Zenon as an Isabelle tactic independently of TLA+.

5 Conclusions and Future Work

We have presented a hierarchically structured proof language for TLA+. It has several important features that help in managing the complexity of proofs. The hierarchical structure means that changes made at any level of a proof are contained inside that level, which helps construct and maintain proofs. Leaf proofs can be omitted and the resulting incomplete proof can be checked. This allows different parts of the proof to be written separately, in a non-linear fashion. The more traditional linear proof style, in which steps that have not yet been proved can be used only if explicitly added as hypotheses, encourages proofs that use many separate lemmas. Such proofs lack the coherent structure of a single hierarchical proof.

The proof language lets the user freely and repeatedly make facts and definitions usable or hidden. Explicitly stating what is being used to prove each step makes the proof easier for a human to understand. It also aids a back-end prover by limiting its search for a proof to ones that use only necessary facts.

There are other declarative proof languages that are similar to TLA+2. Isar [17] is one such language, but it has significant differences that encourage a different style of proof development. For example, it provides an *accumulator* facility to avoid explicit references to proof steps. This is fine for short proofs, but in our experience does not work well for long proofs that are typical of algorithm verification that TLA+2 targets. Moreover, because Isabelle is designed for interactive use, the effects of the Isar proof commands are not always easily predictable, and this encourages a linear rather than hierarchical proof development style. The Focal Proof Language [1] is essentially a subset of the TLA+2 proof language. Our experience with hierarchical proofs in Focal provides additional confidence in the attractiveness of our approach. We know of no declarative proof language that has as flexible a method of using and hiding facts and definitions as that of TLA+2.

The PM transforms a proof into a collection of proof obligations to be verified by a back-end prover. Its current version handles proofs of theorems in the non-temporal fragment of TLA+ that do not involve module instantiation (importing of modules with substitution). Even with this limitation, the system can be useful for many engineering applications. We are therefore concentrating on making the PM and its back-end provers handle this fragment of TLA+ effectively before extending them to the complete language. The major work that remains to be done on this is to complete the Zenon and Isabelle inference rules for reasoning about the built-in constant operators of TLA+. There are also a few non-temporal aspects of the TLA+2 language that the PM does not yet handle, such as subexpression naming. We also expect to extend the PM to support additional back-end provers, including decision procedures for arithmetic and for propositional temporal logic.

We do not anticipate that any major changes will be needed to the TLA+2 proof language. We do expect some minor tuning as we get more experience using it. For example, we are not sure whether local definitions should be usable by default. A graphical user interface is being planned for the TLA+ tools, including the PM. It will support the non-linear development of proofs that the language and the proof system allow.

References

[1] P. Ayrault, M. Carlier, D. Delahaye, C. Dubois, D. Doligez, L. Habib, T. Hardin, M. Jaume, C. Morisset, F. Pessaux, R. Rioboo, and P. Weis. Secure software within Focal. In *Computer & Electronics Security Applications Rendez-vous*, December 2008.

[2] Richard Bonichon, David Delahaye, and Damien Doligez. Zenon : An extensible automated theorem prover producing checkable proofs. In N. Dershowitz and A. Voronkov, editors, *Proc. 14th LPAR*, pages 151–165, 2007.

[3] David Déharbe, Pascal Fontaine, Silvio Ranise, and Christophe Ringeissen. Decision procedures for the formal analysis of software. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *Intl. Coll. Theoretical Aspects of Computing (ICTAC 2007)*, volume 4281 of *Lecture Notes in Computer Science*, pages 366–370, Tunis, Tunisia, 2007. Springer. See also `http://harvey.loria.fr/`.

[4] Urban Engberg, Peter Grønning, and Leslie Lamport. Mechanical verification of concurrent systems with TLA. In G. v. Bochmann and D. K. Probst, editors, *Proc. 4th CAV*, volume 663 of *Lecture Notes in Computer Science*, pages 44–55. Springer-Verlag, June 1992.

[5] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *Proc. 12th TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181, Vienna, Austria, 2006. Springer Verlag.

[6] Eli Gafni and Leslie Lamport. Disk Paxos. *Distributed Computing*, 16(1):1–20, 2003.

[7] Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL: a theorem proving environment for higher-order logic*. Cambridge University Press, 1993.

[8] John Harrison. The HOL Light theorem prover. `http://www.cl.cam.ac.uk/~jrh13/hol-light/index.html`.

[9] Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, August 1993.

[10] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[11] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2003.

[12] Leslie Lamport. TLA+2: A preliminary guide. Draft manuscript, April 2008. `http://research.microsoft.com/users/lamport/tla/tla2-guide.pdf`.

[13] A. C. Leisenring. *Mathematical Logic and Hilbert's ε-Symbol*. Gordon and Breach, New York, 1969.

[14] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1994.

[15] Piotr Rudnicki. An overview of the mizar project. In *Workshop on Types for Proofs and Programs*, Gothenburg, Sweden, 1992. Bastad. `http://www.mizar.org`.

[16] The Coq Development Team (Project TypiCal). The Coq proof assistant reference manual, 2008. `http://coq.inria.fr/V8.1pl3/refman/index.html`.

[17] Makarius Wenzel. The Isabelle/Isar reference manual, June 2008. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle/doc/isar-ref.pdf`.

A Details of the PM

We shall now give a somewhat more formal specification of the PM and prove the key Structural Soundness Theorem 1. We begin with a quick summary of the abstract syntax of TLA+2 proofs, ignoring the stylistic aspects of their concrete representation. (See [12] for a more detailed presentation of the proof language.)

Definition 2 (TLA+2 Proof Language). *TLA+2* proofs*,* non-leaf proofs*,* proof steps *and* begin-step *tokens have the following syntax, where n ranges over natural numbers, l over labels, e over expressions, Φ over lists of expressions, o over operator definitions, Ψ over sets of operator names, $\vec{\beta}$ over lists of binders (i.e., constructs of the form x and $x \in e$ used to build quantified expressions), and α over expressions or* … *forms.*

$$
\begin{array}{llll}
\textit{(Proofs)} & \pi & ::= & \quad | \quad | \quad \Phi \quad \Psi \mid \Pi \\
\textit{(Non-leaf proofs)} & \Pi & ::= & \sigma. \quad \pi \\
 & & & \mid \sigma.\,\tau \quad \Pi \\
\textit{(Proof steps)} & \tau & ::= & \quad \Phi \quad \Psi \mid \quad \Phi \quad \Psi \mid \quad o \\
 & & & \mid \quad e \mid \quad \vec{\beta} \mid \quad \Phi \\
 & & & \mid \alpha \quad \pi \mid \quad \alpha \quad \pi \mid \quad \vec{\beta} : e \quad \pi \\
\textit{(Begin-step tokens)} & \sigma & ::= & \langle n \rangle \mid \langle n \rangle l
\end{array}
$$

A proof that is not a non-leaf proof is called a leaf proof. *The level numbers of a non-leaf proof must all be the same, and those in the subproof of a step (that is, the π in α π, etc.) must be strictly greater than that of the step itself.*

A.1 The Meta-Language

The PM uses proofs in the TLA+2 proof language (Definition 2) to manipulate constructs in the meta-language of TLA+2. This meta-language naturally has no representation in TLA+2 itself; we define its syntax formally as follows.

Definition 3 (Meta-Language). *The TLA+2 meta-language consists of* obligations, assumptions *and* definables *with the following syntax, where e ranges over TLA+2 expressions, x and o over TLA+2 identifiers, and $\vec{x}$ over lists of TLA+2 identifiers.*

$$
\begin{array}{llll}
\textit{(Obligations)} & \phi & ::= (h_1,\ldots,h_n \Vdash e) & (n \geq 0) \\
\textit{(Assumptions)} & h & ::= \quad x \mid o \triangleq \delta \mid \phi \mid [o \triangleq \delta] \mid [\phi] & \\
\textit{(Definables)} & \delta & ::= \phi \mid \quad \vec{x} : e &
\end{array}
$$

The expression after $\Vdash$ in an obligation is called its goal. *An assumption written inside square brackets [ ] is said to be* hidden*; otherwise it is* usable. *For any assumption h, we write $\overline{h}$ (read: h* made usable*) to stand for h with its brackets removed if it is a hidden assumption, and to stand for h if it is not hidden. A list of assumptions is called a* context*, with the empty context written as $\cdot$; we let Γ, Δ and Ω range over contexts, with Γ,Δ standing for the context that is the concatenation of Γ and Δ. The context $\overline{\Gamma}$ is Γ with all its hidden assumptions made usable. The obligation $(\cdot \Vdash e)$ is written simply as e. The assumptions x, $o \triangleq \delta$ and $[o \triangleq \delta]$* bind *the identifiers x and o respectively. We write $x \in \Gamma$ if x is bound in Γ and $x \notin \Gamma$ if x is not bound in Γ. The context Γ, h is considered syntactically well-formed iff h does not bind an identifier already bound in Γ.*

An obligation is a statement that its goal follows from the assumptions in its context. TLA+2 already defines such a statement using ... , but the contexts in such statements have no hidden assumptions or definitions. (To simplify the presentation, we give the semantics of a slightly enhanced proof language where proof steps are allowed to mention obligations instead of just TLA+2 ... statements.) We define an embedding of obligations into Isabelle/TLA+ propositions, which we take as the ultimate primitives of the TLA+2 meta-logic.

Definition 4. *The Isabelle/TLA+* embedding $(-)_{\mathrm{Isa}}$ *of obligations, contexts and definables is as follows:*

$$
\begin{array}{rclrcl}
 & & & (\cdot)_{\mathrm{Isa}} & = & \\
(\Gamma \Vdash e)_{\mathrm{Isa}} & = & (\overline{\Gamma})_{\mathrm{Isa}}\, e & (\Gamma, \quad x)_{\mathrm{Isa}} & = & (\Gamma)_{\mathrm{Isa}} \bigwedge x. \\
(\quad \vec{x} : e)_{\mathrm{Isa}} & = & \lambda \vec{x}.\, e & (\Gamma, o \triangleq \delta)_{\mathrm{Isa}} & = & (\Gamma)_{\mathrm{Isa}} \bigwedge o.\, (o \equiv (\delta)_{\mathrm{Isa}}) \Longrightarrow \\
 & & & (\Gamma, \phi)_{\mathrm{Isa}} & = & (\Gamma)_{\mathrm{Isa}}\, ((\phi)_{\mathrm{Isa}}) \Longrightarrow
\end{array}
$$

For example, $(\quad P, [(\quad x \Vdash P(x))] \Vdash \forall x : P(x))_{\mathrm{Isa}} = \bigwedge P.\ (\bigwedge x.\ P(x)) \Longrightarrow \forall x : P(x)$. Note that usable and hidden assumptions are treated identically for the provability of an obligation.

The embedding of ordinary TLA+2 expressions is the identity because Isabelle/TLA+ contains TLA+2 expressions as part of its object syntax. Thus, we do not have to trust the embedding of ordinary TLA+2 expressions, just that of the obligation language. In practice, some aspects of TLA+2 expressions, such as the indentation-sensitive conjunction and disjunction lists, are sent by the PM to Isabelle using an indentation-insensitive encoding.While Isabelle/TLA+ can implicitly generalize over the free identifiers in a lemma, we shall be explicit about binding and consider obligations provable only if they are closed.

Definition 5 (Well-Formed Obligations). *The obligation* $(\Gamma \Vdash e)$ *is said to be* well-formed *iff it is closed and* $(\Gamma \Vdash e)_{\mathrm{Isa}}$ *is a well-typed proposition of Isabelle/TLA+.*

Definition 6 (Provability). *The obligation* $(\Gamma \Vdash e)$ *is said to be* provable *iff it is well-formed and* $(\Gamma \Vdash e)_{\mathrm{Isa}}$ *is certified by the Isabelle kernel to follow from the axioms of the Isabelle/TLA+ object logic.*

We trust Isabelle/TLA+ to be sound with respect to the semantics of TLA+2, and therefore provability to imply truth. Formally, we work under the following *trust* axiom.

Axiom 7 (Trust). *If* ϕ *is provable, then it is true.*

We state a number of useful facts about obligations (which are all theorems in Isabelle/TLA+), omitting their trivial proofs. The last one (Fact 13) is true because TLA+ is based on classical logic.

Fact 8 (Definition). *If* $(\Gamma, \quad o, \Delta \Vdash e)$ *is provable, then* $(\Gamma, o \triangleq \delta, \Delta \Vdash e)$ *is provable if it is well-formed.*

Fact 9 (Weakening). *If* $(\Gamma, \Delta \Vdash e)$ *is provable, then* $(\Gamma, h, \Delta \Vdash e)$ *is provable if it is well-formed.*

Fact 10 (Expansion). *If* $(\Gamma, o \triangleq \delta, \Delta \Vdash e)$ *is provable, then* $(\Gamma, o \triangleq \delta, \Delta[o := \delta] \Vdash e[o := \delta])$ *is provable.*

Fact 11 (Strengthening). *If* $(\Gamma, \quad o, \Delta \Vdash e)$ *or* $(\Gamma, o \triangleq \delta, \Delta \Vdash e)$ *is provable and* o *is not free in* $(\Delta \Vdash e)$*, then* $(\Gamma, \Delta \Vdash e)$ *is provable.*

Fact 12 (Cut). *If* $(\Gamma, \Delta \Vdash e)$ *is provable and* $(\Gamma, (\Delta \Vdash e), \Omega \Vdash f)$ *is provable, then* $(\Gamma, \Omega \Vdash f)$ *is provable.*

Fact 13. *If* $(\Gamma, \neg e, \Delta \Vdash e)$ *is provable, then* $(\Gamma, \Delta \Vdash e)$ *is provable.*

The / steps change the visibility of definitions in a context (Definition 14 below). Note that changing the visibility of a definition does not affect the provability of an obligation because the Isabelle embedding (Definition 4) makes all hidden definitions usable.

Definition 14. *If* Γ *is a context and* Ψ *a set of operator names, then:*

1. Γ with Ψ made usable*, written* $\Gamma \quad \Psi$*, is constructed from* Γ *by replacing all assumptions of the form* $[o \triangleq \delta]$ *in* Γ *with* $o \triangleq \delta$ *for every* $o \in \Psi$*.*
2. Γ with Ψ made hidden*, written* $\Gamma \quad \Psi$*, is constructed from* Γ *by replacing all assumptions of the form* $o \triangleq \delta$ *in* Γ *with* $[o \triangleq \delta]$ *for every* $o \in \Psi$*.*

A sequence of binders $\vec{\beta}$ in the TLA+2 expressions $\forall \vec{\beta} : e$ or $\exists \vec{\beta} : e$ can be reflected as assumptions.

Definition 15 (Binding Reflection). *If* $\vec{\beta}$ *is a list of binders with each element of the form* x *or* $x \in e$*, then the* reflection *of* $\vec{\beta}$ *as assumptions, written* $\|\vec{\beta}\|$*, is given inductively as follows.*

$$\|\cdot\| = \cdot \qquad\qquad \|\vec{\beta}, x\| = \|\vec{\beta}\|, \quad x \qquad\qquad \|\vec{\beta}, x \in e\| = \|\vec{\beta}\|, \quad x, x \in e$$

A.2 Interpreting Proofs

Let us recall some definitions from section 3.

Definition 16 (Claims and Transformations). *A* claim *is a judgement of the form* $\pi : (\Gamma \Vdash e)$ *where* π *is a TLA+2 proof. A* transformation *is a judgement of the form* $\sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)$ *where* σ *is a begin-step token and* τ *a proof step. A claim (respectively, transformation) is said to be* complete *if its proof (respectively, proof step) does not contain any occurrence of the leaf proof .*

The PM generates leaf obligations for a claim using two mutually recursive procedures, *checking* and *transformation*, specified below using the formalism of a *primitive derivation*.

Definition 17. *A* primitive derivation *is a derivation constructed using inferences of the form*

$$\frac{\mathcal{D}_1 \qquad \cdots \qquad \mathcal{D}_n}{E} \qquad (n \geq 0)$$

where E is either a claim or a transformation, and $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are primitive derivations or obligations. An obligation at the leaf of a primitive derivation is called a leaf obligation*.*

Definition 18 (Checking and Transformation). *The primitive derivations of a claim or transformation are constructed using the following* checking *and* transformation *rules.*

1. Checking *rules*

$$\frac{(\Gamma \Vdash e)}{\quad : (\Gamma \Vdash e)} \qquad \frac{}{\quad : (\Gamma \Vdash e)}$$

$$\frac{\langle 0 \rangle. \quad \Phi \qquad \Psi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f) \qquad (\Delta \Vdash f)}{\Phi \qquad \Psi : (\Gamma \Vdash e)}$$

$$\frac{\pi : (\Gamma \Vdash e)}{\sigma. \qquad \pi : (\Gamma \Vdash e)} \qquad \frac{\sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f) \qquad \Pi : (\Delta \Vdash f)}{\sigma.\tau \quad \Pi : (\Gamma \Vdash e)} \text{ non-}$$

2. Transformation

$$\frac{\sigma. \qquad \Phi : (\Gamma \qquad \Psi \Vdash e) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad \Phi \qquad \Psi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)}$$

$$\frac{\sigma. \qquad \Phi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad \Phi \qquad \Psi : (\Gamma \Vdash e) \longrightarrow (\Delta \qquad \Psi \Vdash f)}$$

$$\frac{}{\sigma. \qquad o \triangleq \delta : (\Gamma \Vdash e) \longrightarrow (\Gamma, [o \triangleq \delta] \Vdash e)} \qquad (o \notin \Gamma)$$

$$\frac{}{\sigma. \qquad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)} \; 0 \qquad \frac{}{\sigma. \qquad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)} \; 0$$

$$\frac{\sigma. \qquad \Phi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f) \qquad (\overline{\Delta}, \Gamma_0 \Vdash e_0)}{\sigma. \qquad \Phi, (\Gamma_0 \Vdash e_0) : (\Gamma \Vdash e) \longrightarrow (\Delta, (\Gamma_0 \Vdash e_0) \Vdash f)} \; 1$$

$$\frac{\sigma. \qquad \Phi : (\Gamma_0, [\phi], \Gamma_1 \Vdash e) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad \Phi, \phi : (\Gamma_0, \phi, \Gamma_1 \Vdash e) \longrightarrow (\Delta \Vdash f)} \; 1$$

$$\frac{}{\sigma. \qquad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)} \; 0 \qquad \frac{}{\sigma. \qquad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)} \; 0$$

$$\frac{\sigma. \qquad \vec{\beta} : (\Gamma, \qquad u \Vdash e[x := u]) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad u, \vec{\beta} : (\Gamma \Vdash \forall x : e) \longrightarrow (\Delta \Vdash f)} \; 1$$

$$\frac{(\Gamma \Vdash S \subseteq T) \qquad \sigma. \qquad \vec{\beta} : (\Gamma, \qquad u, u \in T \Vdash e[x := u]) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad u \in T, \vec{\beta} : (\Gamma \Vdash \forall x \in S : e) \longrightarrow (\Delta \Vdash f)} \; 2$$

$$\frac{\sigma. \qquad \Omega : (\Gamma \Vdash e[x := w]) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad w, \Omega : (\Gamma \Vdash \exists x : e) \longrightarrow (\Delta \Vdash f)} \; 1$$

$$\frac{(\Gamma \Vdash T \subseteq S) \qquad (\Gamma \Vdash w \in T) \qquad \sigma. \qquad \Omega : (\Gamma, w \in T \Vdash e[x := w]) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad w \in T, \Omega : (\Gamma \Vdash \exists x \in S : e) \longrightarrow (\Delta \Vdash f)} \; 2$$

$$\frac{(\Gamma, e \Vdash g)}{\sigma. \qquad g : (\Gamma \Vdash e \Rightarrow f) \longrightarrow (\Gamma, g \Vdash f)}$$

$$\frac{\pi : (\Gamma, [\neg e], \Delta \Vdash f)}{\langle n \rangle. \; (\Delta \Vdash f) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, (\Delta \Vdash f) \Vdash e)} \; 1$$

$$\frac{\pi : (\Gamma, \langle n \rangle l \triangleq (\Delta \Vdash f), [\neg e], \Delta \Vdash f)}{\langle n \rangle l.\ (\Delta \Vdash f) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, \langle n \rangle l \triangleq (\Delta \Vdash f), [\langle n \rangle l] \Vdash e)} \quad 2$$

$$\frac{\sigma.\ (g \Vdash e) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)}{\sigma. \qquad g \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)}$$

$$\frac{\pi : (\Gamma, (\Delta \Vdash f) \Vdash e)}{\langle n \rangle. \qquad (\Delta \Vdash f) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, [\neg e], \Delta \Vdash f)} \quad 1$$

$$\frac{\pi : (\Gamma, \langle n \rangle l \triangleq (\Delta \Vdash f), [\langle n \rangle l] \Vdash e)}{\langle n \rangle l. \qquad (\Delta \Vdash f) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, \langle n \rangle l \triangleq (\Delta \Vdash f), [\neg e], \Delta \Vdash f)} \quad 2$$

$$\frac{\pi : \left(\Gamma \Vdash \exists \vec{\beta} : p\right)}{\sigma. \qquad \vec{\beta} : p \qquad \pi : (\Gamma \Vdash e) \longrightarrow \left(\Gamma, \left\|\vec{\beta}\right\|, p \Vdash e\right)}$$

The inference rules in the above definition are deterministic: the conclusion of each rule uniquely determines the premises. However, the rules are partial; for example, there is no rule that concludes a transformation of the form $\sigma. \qquad x \in S : (\Gamma \Vdash B \wedge C) \longrightarrow (\Delta \Vdash f)$.

Definition 19. *A claim or a transformation is said to be* meaningful *if it has a primitive derivation.*

Definition 20 (Generating Leaf Obligations). *A meaningful claim or transformation is said to* generate *the leaf obligations of its primitive derivation.*

In the rest of this appendix we limit our attention to complete meaningful claims and transformations.

A.3 Correctness

If the leaf obligations generated by a complete meaningful claim are provable, then the obligation in the claim itself ought to be provable. In this section we prove this theorem by analysis of the checking and transformation rules.

Definition 21 (Provability of Claims and Transformation).

1. *The claim $\pi : (\Gamma \Vdash e)$ is* provable *iff it is complete and meaningful and the leaf obligations it generates are all provable.*
2. *The transformation $\sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)$ is* provable *iff it is complete and meaningful and the leaf obligations it generates are all provable.*

Theorem 22 (Correctness).

(1) If $\pi : (\Gamma \Vdash e)$ is provable, then $(\Gamma \Vdash e)$ is provable.
(2) If $\sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)$ is provable and $(\Delta \Vdash f)$ is provable, then $(\Gamma \Vdash e)$ is provable.

Proof. Let $\mathcal{D}$ be the primitive derivation for the claim in (1) and let $\mathcal{E}$ be the primitive derivation for the transformation in (2). The proof will be by lexicographic induction on the structures of $\mathcal{D}$ and $\mathcal{E}$, with a provable transformation allowed to justify a provable claim.

⟨1⟩1. If $\pi : (\Gamma \Vdash e)$ is provable, then $(\Gamma \Vdash e)$ is provable.

⟨2⟩1. *Case* π is , *i.e.*, $\mathcal{D} = \dfrac{(\Gamma \Vdash e)}{: (\Gamma \Vdash e)}$. *Obvious*

⟨2⟩2. *Case* π is is impossible because $\pi : (\Gamma \Vdash e)$ is complete.

⟨2⟩3. *Case* π is $\Phi \qquad \Psi$, *i.e.*,

$$\mathcal{D} = \frac{\langle 0 \rangle. \qquad \Phi \qquad \overset{\mathcal{E}_0}{\Psi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)} \qquad (\Delta \Vdash f)}{\Phi \qquad \Psi : (\Gamma \Vdash e)} \quad .$$

⟨3⟩1. $(\Delta \Vdash f)$ is provable *By* Definition 21.
⟨3⟩2. *Qed* *By* ⟨3⟩1, i.h. (inductive hypothesis) for $\mathcal{E}_0$.

⟨2⟩4. *Case* π is $\sigma.$ π_0, *i.e.*, $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_0\\ \pi_0 : (\Gamma \Vdash e)\end{array}}{\sigma. \qquad \pi_0 : (\Gamma \Vdash e)}$. *By* i.h. for $\mathcal{D}_0$.

⟨2⟩5. *Case* π is $\sigma.\tau \;\; \Pi$, *i.e.*,

$$\mathcal{D} = \frac{\begin{array}{c}\mathcal{E}_0\\ \sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)\end{array} \qquad \begin{array}{c}\mathcal{D}_0\\ \Pi : (\Delta \Vdash f)\end{array}}{\sigma.\tau \;\; \Pi : (\Gamma \Vdash e)} \text{ non-} \quad .$$

⟨3⟩1. $(\Delta \Vdash f)$ is provable *By* i.h. for $\mathcal{D}_0$.
⟨3⟩3. *Qed* *By* ⟨3⟩1, i.h. for $\mathcal{E}_0$.

⟨2⟩6. *Qed* *By* ⟨2⟩1, ..., ⟨2⟩5.

⟨1⟩2. If $\sigma.\tau : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)$ is provable and $(\Delta \Vdash f)$ is provable, then $(\Gamma \Vdash e)$ is provable.

⟨2⟩1. *Case* τ is $\quad \Phi \quad\quad \Psi$, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \sigma. \quad\; \Phi : (\Gamma \quad\quad \Psi \Vdash e) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad\; \Phi \quad\quad \Psi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)} \quad .$$

⟨3⟩1. $(\Gamma \quad\quad \Psi \Vdash e)$ is provable *By* i.h. for $\mathcal{E}_0$.
⟨3⟩2. *Qed* *By* ⟨3⟩1, Definition 14.

⟨2⟩2. *Case* τ is $\quad \Phi \quad\quad \Psi$, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \sigma. \quad\; \Phi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad\; \Phi \quad\quad \Psi : (\Gamma \Vdash e) \longrightarrow (\Delta \quad\quad \Psi \Vdash f)} \quad .$$

⟨3⟩1. $(\Delta \Vdash f)$ is provable *By* provability of $(\Delta \quad\quad \Psi \Vdash f)$ and Definition 14.
⟨3⟩2. *Qed* *By* ⟨3⟩1, i.h. for $\mathcal{E}_0$.

⟨2⟩3. *Case* τ is $\quad o \triangleq \delta$ with $o \notin \Gamma$, *i.e.*,

$$\mathcal{E} = \frac{}{\sigma. \quad\quad o \triangleq \delta : (\Gamma \Vdash e) \longrightarrow (\Gamma, [o \triangleq \delta] \Vdash e)} \quad .$$

⟨3⟩1. o is not free in e *By* $o \notin \Gamma$ and closedness of $(\Gamma \Vdash e)$.
⟨3⟩2. *Qed* *By* ⟨3⟩1, strengthening (Fact 11).

⟨2⟩4. *Case* τ is $\quad \bullet$, *i.e.*, $\mathcal{E} = \dfrac{}{\sigma. \quad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)} \quad 0.$ *Obvious*

⟨2⟩5. *Case* τ is $\quad \bullet$, *i.e.*, $\mathcal{E} = \dfrac{}{\sigma. \quad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)} \quad 0.$ *Obvious*

⟨2⟩6. *Case* τ is $\quad \Phi, \phi$, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \sigma. \quad\; \Phi : (\Gamma \Vdash e) \longrightarrow (\Delta_0 \Vdash f)\end{array} \qquad (\overline{\Delta_0}, \Gamma_0 \Vdash e_0)}{\sigma. \quad\; \Phi, (\Gamma_0 \Vdash e_0) : (\Gamma \Vdash e) \longrightarrow (\Delta_0, (\Gamma_0 \Vdash e_0) \Vdash f)} \quad 1$$

⟨3⟩1. $(\overline{\Delta_0}, \Gamma_0 \Vdash e_0)$ is provable *By* Definition 21.
⟨3⟩2. $(\Delta_0, \Gamma_0 \Vdash e_0)$ is provable *By* ⟨3⟩1, Definition 4.
⟨3⟩3. $(\Delta_0 \Vdash f)$ is provable *By* provability of $(\Delta_0, (\Gamma_0 \Vdash e_0) \Vdash f)$, ⟨3⟩2, cut (Fact 12).
⟨3⟩4. *Qed* *By* ⟨3⟩3, i.h. for $\mathcal{E}_0$

⟨2⟩7. *Case* τ is $\quad \Phi, \phi$, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \sigma. \quad\; \Phi : (\Gamma_0, [\phi], \Gamma_1 \Vdash e) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad\; \Phi, \phi : (\Gamma_0, \phi, \Gamma_1 \Vdash e) \longrightarrow (\Delta \Vdash f)} \quad 1.$$

⟨3⟩1. $(\Gamma_0, [\phi], \Gamma_1 \Vdash e)$ is provable *By* provability of $(\Delta \Vdash f)$, i.h. for $\mathcal{E}_0$.
⟨3⟩2. *Qed* *By* ⟨3⟩1, $(\Gamma_0, [\phi], \Gamma_1 \Vdash e)_{\mathrm{Isa}} = (\Gamma_0, \phi, \Gamma_1 \Vdash e)_{\mathrm{Isa}}$ (Definition 4).

⟨2⟩8. *Case* τ is •, *i.e.*, $\mathcal{E} = \dfrac{}{\sigma. \quad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)}\ {}_0.$ *Obvious*

⟨2⟩9. *Case* τ is •, *i.e.*, $\mathcal{E} = \dfrac{}{\sigma. \quad \bullet : (\Gamma \Vdash e) \longrightarrow (\Gamma \Vdash e)}\ {}_0.$ *Obvious*

⟨2⟩10. *Case* τ is $u, \vec{\beta}$, *i.e.*,

$$\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_0 \\ \sigma. \quad \vec{\beta} : (\Gamma, \quad u \Vdash e[x := u]) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad u, \vec{\beta} : (\Gamma \Vdash \forall x : e) \longrightarrow (\Delta \Vdash f)}\ {}_1.$$

⟨3⟩1. $(\Gamma, \quad u \Vdash e[x := u])$ is provable *By* i.h. for $\mathcal{E}_0$.
⟨3⟩2. *Qed* *By* ⟨3⟩1 and predicate logic.

⟨2⟩11. *Case* τ is $\sigma. \quad u \in T$, *i.e.*,

$$\mathcal{E} = \dfrac{(\Gamma \Vdash S \subseteq T) \qquad \begin{array}{c}\mathcal{E}_0 \\ \sigma. \quad \vec{\beta} : (\Gamma, \quad u, u \in T \Vdash e[x := u]) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad u \in T, \vec{\beta} : (\Gamma \Vdash \forall x \in S : e) \longrightarrow (\Delta \Vdash f)}\ {}_2.$$

⟨3⟩1. $(\Gamma, \quad u, u \in T \Vdash e[x := u])$ is provable *By* i.h on $\mathcal{E}_0$.
⟨3⟩2. $(\Gamma, \quad u, u \in S \Vdash u \in T)$ is provable
⟨4⟩1. $(\Gamma, \quad u \Vdash S \subseteq T)$ is provable *By* Definition 21, weakening (Fact 9).
⟨4⟩2. *Qed* *By* ⟨4⟩1, Definition of $\subseteq$.
⟨3⟩3. $(\Gamma, \quad u, u \in S \Vdash e[x := u])$ is provable *By* ⟨3⟩1, ⟨3⟩2, cut (Fact 12).
⟨3⟩4. *Qed* *By* ⟨3⟩3 and predicate logic.

⟨2⟩12. *Case* τ is w, Ω, *i.e.*,

$$\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_0 \\ \sigma. \quad \Omega : (\Gamma \Vdash e[x := w]) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad w, \Omega : (\Gamma \Vdash \exists x : e) \longrightarrow (\Delta \Vdash f)}\ {}_1.$$

⟨3⟩1. $(\Gamma \Vdash e[x := w])$ is provable *By* i.h. for $\mathcal{E}_0$.
⟨3⟩2. *Qed* *By* ⟨3⟩1.

⟨2⟩13. *Case* τ is $w \in T, \Omega$ and:

$$\mathcal{E} = \dfrac{(\Gamma \Vdash T \subseteq S) \qquad (\Gamma \Vdash w \in T) \qquad \begin{array}{c}\mathcal{E}_0 \\ \sigma. \quad \Omega : (\Gamma, w \in T \Vdash e[x := w]) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \quad w \in T, \Omega : (\Gamma \Vdash \exists x \in S : e) \longrightarrow (\Delta \Vdash f)}\ {}_2.$$

⟨3⟩1. $(\Gamma, w \in T \Vdash e[x := w])$ is provable *By* i.h. for $\mathcal{E}_0$.
⟨3⟩2. $(\Gamma \Vdash w \in T)$ is provable *By* Definition 21.
⟨3⟩3. $(\Gamma \Vdash e[x := w])$ is provable *By* ⟨3⟩1, ⟨3⟩2, cut (Fact 12).
⟨3⟩4. $(\Gamma \Vdash w \in S)$ is provable
⟨4⟩1. $(\Gamma, w \in T \Vdash w \in S)$ is provable *By* Definition 21, Definition of $\subseteq$.
⟨4⟩2. *Qed* *By* ⟨4⟩1, ⟨3⟩2, cut (Fact 12).
⟨3⟩5. *Qed* *By* ⟨3⟩3, ⟨3⟩4, and predicate logic.

⟨2⟩14. τ is g, *i.e.*,

$$\mathcal{E} = \dfrac{(\Gamma, e \Vdash g)}{\sigma. \quad g : (\Gamma \Vdash e \Rightarrow f) \longrightarrow (\Gamma, g \Vdash f)}\ .$$

⟨3⟩1. $(\Gamma, e, g \Vdash f)$ is provable *By* weakening (Fact 9).
⟨3⟩2. $(\Gamma, e \Vdash g)$ is provable *By* Definition 21.

⟨3⟩3. $(\Gamma, e \Vdash f)$ is provable *By* ⟨3⟩1, ⟨3⟩2, cut (Fact 12).
⟨3⟩4. $(\Gamma \Vdash e \Rightarrow f)$ is provable *By* ⟨3⟩3.

⟨2⟩15. $\sigma.\tau$ is $\langle n \rangle.\ (\Omega \Vdash g)$ π, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{D}_0 \\ \pi : (\Gamma, [\neg e], \Omega \Vdash g)\end{array}}{\langle n \rangle.\ (\Omega \Vdash g) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, (\Omega \Vdash g) \Vdash e)} \quad 1.$$

⟨3⟩1. $(\Gamma, [\neg e], (\Omega \Vdash g) \Vdash e)$ is provable *By* weakening (Fact 9).
⟨3⟩2. $(\Gamma, [\neg e], \Omega \Vdash g)$ is provable *By* i.h. for $\mathcal{D}_0$.
⟨3⟩3. $(\Gamma, [\neg e] \Vdash e)$ is provable *By* ⟨3⟩1, ⟨3⟩2, cut (Fact 12).
⟨3⟩4. *Qed* *By* ⟨3⟩3, Fact 13.

⟨2⟩16. *Case* $\sigma.\tau$ is $\langle n \rangle l.\ (\Omega \Vdash g)$ π, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{D}_0 \\ \pi : (\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], \Omega \Vdash g)\end{array}}{\langle n \rangle l.\ (\Omega \Vdash g) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\langle n \rangle l] \Vdash e)} \quad 2.$$

⟨3⟩1. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], [\langle n \rangle l] \Vdash e)$ is provable
By provability of $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\langle n \rangle l] \Vdash e)$, weakening (Fact 9).
⟨3⟩2. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], [(\Omega \Vdash g)] \Vdash e)$ is provable *By* ⟨3⟩1, expansion (Fact 10).
⟨3⟩3. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], \Omega \Vdash g)$ is provable *By* i.h. for $\mathcal{D}_0$.
⟨3⟩4. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e] \Vdash e)$ is provable *By* ⟨3⟩2, ⟨3⟩3, cut (Fact 12).
⟨3⟩5. $(\Gamma, [\neg e] \Vdash e)$ is provable *By* ⟨3⟩4, strengthening (Fact 11).
⟨3⟩6. *Qed* *By* ⟨3⟩5, Fact 13.

⟨2⟩17. τ is g π, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0 \\ \sigma.\ (g \Vdash e) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)\end{array}}{\sigma. \qquad g \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Delta \Vdash f)} \quad .$$

By i.h. for $\mathcal{E}_0$.

⟨2⟩18. τ is $\langle n \rangle.$ $(\Omega \Vdash g)$ π, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{D}_0 \\ \pi : (\Gamma, (\Omega \Vdash g) \Vdash e)\end{array}}{\langle n \rangle. \qquad (\Delta \Vdash f) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, [\neg e], \Omega \Vdash g)} \quad 1.$$

⟨3⟩1. $(\Gamma, [\neg e], (\Omega \Vdash g) \Vdash e)$ is provable *By* i.h. for $\mathcal{D}_0$, weakening (Fact 9).
⟨3⟩2. $(\Gamma, [\neg e] \Vdash e)$ is provable *By* provability of $(\Gamma, [\neg e], \Omega \Vdash g)$, ⟨3⟩1, cut (Fact 12).
⟨3⟩3. *Qed* *By* ⟨3⟩2, Fact 13.

⟨2⟩19. $\sigma.\tau$ is $\langle n \rangle l.$ $(\Omega \Vdash g)$ π, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{D}_0 \\ \pi : (\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\langle n \rangle l] \Vdash e)\end{array}}{\langle n \rangle l. \qquad (\Omega \Vdash g) \qquad \pi : (\Gamma \Vdash e) \longrightarrow (\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], \Omega \Vdash g)} \quad 2.$$

⟨3⟩1. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], [\langle n \rangle l] \Vdash e)$ is provable *By* i.h. for $\mathcal{D}_0$, weakening (Fact 9).
⟨3⟩2. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], [(\Omega \Vdash g)] \Vdash e)$ is provable *By* ⟨3⟩1, expansion (Fact 10).
⟨3⟩3. $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e] \Vdash e)$ is provable
By ⟨3⟩2, provability of $(\Gamma, \langle n \rangle l \triangleq (\Omega \Vdash g), [\neg e], \Omega \Vdash g)$, cut (Fact 12).
⟨3⟩4. $(\Gamma, [\neg e] \Vdash e)$ is provable *By* ⟨3⟩3, strengthening (Fact 11).
⟨3⟩5. *Qed* *By* ⟨3⟩4, Fact 13.

⟨2⟩20. *Case* τ is $\vec{\beta} : p$ π, *i.e.*,

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{D}_0 \\ \pi : \left(\Gamma \Vdash \exists \vec{\beta} : p\right)\end{array}}{\sigma. \qquad \vec{\beta} : p \qquad \pi : (\Gamma \Vdash e) \longrightarrow \left(\Gamma, \left\|\vec{\beta}\right\|, p \Vdash e\right)} \quad .$$

⟨3⟩1. $\left(\Gamma, \exists \vec{\beta} : p \Vdash e\right)$ is provable — *By* provability of $\left(\Gamma, \|\vec{\beta}\|, p \Vdash e\right)$, predicate logic.
⟨3⟩2. $\left(\Gamma \Vdash \exists \vec{\beta} : p\right)$ is provable — *By* i.h. for $\mathcal{D}_0$.
⟨3⟩3. *Qed* — *By* ⟨3⟩1, ⟨3⟩2, cut (Fact 12).

⟨2⟩21. *Qed* — *By* ⟨2⟩1, . . . , ⟨2⟩20

⟨1⟩3. *Qed* — *By* ⟨1⟩1, ⟨1⟩2.

□

A.4 Constrained Search

The correctness theorem (22) establishes an implication from the leaf obligations generated by a complete meaningful claim to the obligation of the claim. It is always true, regardless of the provability of any individual leaf obligation. While changing the visibility of assumptions in an obligation does not change its provability, a back-end prover may fail to prove it if important assumptions are hidden. As already mentioned in Section 3, the PM removes these hidden assumptions before sending a leaf obligation to a back-end prover. Therefore, in order to establish the Structural Soundness Theorem (1), we must prove a property about the result of this removal.

Definition 23 (Filtration). *The* filtered *form of any obligation ϕ, written $(\phi)_{\mathrm{f}}$, is obtained by deleting all assumptions of the form $[\phi_0]$ and replacing all assumptions of the form $[o \triangleq \delta]$ with $\quad o$ anywhere inside ϕ.*

For example, $(\quad x, [y \triangleq x] \Vdash x = y)_{\mathrm{f}} = (\quad x, \quad y \Vdash x = y)$. We thus see that filtration can render a true obligation false; however, if the filtered form of an obligation is true, then so is the obligation.

Lemma 24 (Verification Lemma). *If $(\phi)_{\mathrm{f}}$ is provable, then ϕ is provable.*

Proof Sketch. By induction on the structure of the obligation ϕ, with each case a straightforward consequence of facts 8 and 9. □

Definition 25 (Verifiability). *The obligation ϕ is said to be* verifiable *if $(\phi)_{\mathrm{f}}$ is provable.*

We now prove the Structural Soundness Theorem (1).

Theorem 1. *If $\pi : \phi$ is a complete meaningful claim and every leaf obligations it generates is verifiable, then ϕ is true.*

Proof.

⟨1⟩1. For every leaf obligation ϕ_0 generated by $\pi : \phi$, it must be that ϕ_0 is provable.
⟨2⟩1. *Take* ϕ_0 as a leaf obligation generated by $\pi : \phi$.
⟨2⟩2. $(\phi_0)_{\mathrm{f}}$ is provable — *By* assumption and Definition 25.
⟨2⟩3. *Qed* — *By* ⟨2⟩2, Verification Lemma 24.

⟨1⟩2. ϕ is provable — *By* ⟨1⟩1, Correctness Theorem 22.

⟨1⟩3. *Qed* — *By* ⟨1⟩2, Trust Axiom 7.

□

B A TLA^{+2} Proof of Cantor's Theorem

The following is the complete TLA^{+2} proof of Cantor's theorem referenced in Section 2.3.

$$\forall S : \forall f \in [S \to \quad S] : \exists A \in \quad S : \forall x \in S : f[x] \neq A$$

$\langle 1 \rangle 1.$ $\quad S,$
$\quad f \in [S \to \quad S]$
$\quad \exists A \in \quad S : \forall x \in S : f[x] \neq A$
$\quad \langle 2 \rangle 1.$ $\quad T \triangleq \{z \in S : z \notin f[z]\}$
$\quad \langle 2 \rangle 2.$ $\forall x \in S : f[x] \neq T$
$\quad\quad \langle 3 \rangle 1.$ $\quad x \in S \quad f[x] \neq T$
$\quad\quad\quad \langle 4 \rangle 1.$ $\quad x \in T$
$\quad\quad\quad \langle 4 \rangle 2.$ $\quad x \notin T$
$\quad\quad\quad \langle 4 \rangle 3.$ $\quad \langle 4 \rangle 1, \langle 4 \rangle 2$
$\quad\quad \langle 3 \rangle 2.$ $\quad \langle 3 \rangle 1$
$\quad \langle 2 \rangle 3.$ $\quad \langle 2 \rangle 2$
$\langle 1 \rangle 2.$ $\quad \langle 1 \rangle 1$

As an example, the leaf obligation generated (see Appendix A.3) for the proof of $\langle 4 \rangle 1$ is:

$$\Big(\langle 1 \rangle 1 \triangleq (\quad S, \quad f, f \in [S \to \quad S] \Vdash \exists A \in \quad S : \forall x \in S : f[x] \neq A),$$
$$S,$$
$$f, f \in [S \to \quad S],$$
$$T \triangleq \{z \in S : z \notin f[z]\},$$
$$[\neg(\exists A \in \quad S : \forall x \in S : f[x] \neq A)],$$
$$\langle 2 \rangle 2 \triangleq \forall x \in S : f[x] \neq T,$$
$$[\neg(\forall x \in S : f[x] \neq T)],$$
$$\langle 3 \rangle 1 \triangleq (\quad x, x \in S \Vdash f[x] \neq T),$$
$$x, x \in S,$$
$$[\neg(f[x] \neq T)],$$
$$\langle 4 \rangle 1 \triangleq (x \in T \Vdash f[x] \neq T),$$
$$x \in T$$
$$\Vdash f[x] \neq T \Big).$$

Filtering its obligation (see Definition 23) and expanding all definitions gives:

$$\Big(\quad S,$$
$$f, f \in [S \to \quad S],$$
$$x, x \in S,$$
$$x \in \{z \in S : z \notin f[z]\} \Vdash f[x] \neq \{z \in S : z \notin f[z]\} \Big).$$

In Isabelle/TLA$^+$, this is the following lemma:

$\texttt{lemma}\ \bigwedge S.$
$\quad \bigwedge f.\ f \in [S \to \quad S] \Longrightarrow$
$\quad\quad \big(\bigwedge x.\ [\![\, x \in S;$
$\quad\quad\quad x \in \{z \in S : z \notin f[z]\}\,]\!] \Longrightarrow f[x] \neq \{z \in S : z \notin f[z]\}\big)$