Reconstructing Propositional Proofs in Type Theory

Jonathan Prieto-Cubides

Logic and Computation Research Group,
Universidad EAFIT, Medellín, Colombia
`jprieto9@eafit.edu.co`

Abstract. We describe a syntactical proof-reconstruction approach to verify derivations generated by `Metis` prover to theorems in classical propositional logic. To verify such derivations, we formalise in type theory each inference rule of the `Metis` reasoning. We developed a tool jointly with two `Agda` libraries to translate `Metis` derivations to `Agda` proof-terms. These developments allowed us to type-check with `Agda`, `Metis` derivations step-by-step.

Keywords: proof-reconstruction, type theory, automatic theorem prover, proof-assistant

1 Introduction

An automatic theorem prover (henceforth ATP) is a program that intends to prove conjectures from axioms and inference rules of some logical system. In the last decades, ATPs are fast becoming a key instrument in different disciplines and real applications (e. g., verifying a railway interlocking system, an operating-system kernel, or a pseudo-random number generator for cryptography). Since some programming errors have been found in these programs (see, for example, [27, 7, 17]), researchers and users from academy and industry have shown an increased interest to *formally* prove the validity of ATPs' results.

In order to give confidence to the ATP users many of these systems have started to include in their outputs the full derivations associated to the proved theorems. However, existing research recognizes that in many cases these derivations encode non-trivial reasoning hard to reconstruct and therefore hard to verify [33, 27].

Proof-reconstruction addresses this problem. Since many ATPs are poor-documented, this problem becomes in mostly cases a reverse engineering task to verify the prover reasoning. The usual is the reconstruction is made by another and not by the developers of the ATP. Therefore, the presentation of the derivations generated by the prover plays an important role in proof-reconstruction.

To verify such automatically generated derivation by the prover, it is convenient to have them in a consistent format, that is, a full script describing the derivation step-by-step with exhaustive details and without ambiguities. For example, for classical propositional logic (henceforth CPL) from a list of at least forty ATPs—available from the Web service `SystemOnTPTP` of the TPTP World[1]—just few of them show their proofs.

One approach to address the proof-reconstruction problem is proving each deduction of the prover, the *source* system, with a formalization of the prover reasoning in a proof-assistant, the *target* system. The target system is the proof *checker* in charge to verify the source system reasoning for each derivation. These proof-assistants allow us to formalize the logical system used in the proofs, i. e., logical constants, axioms, inference rules, hypotheses, and theorems. A proof-reconstruction tool

[1] `http://www.cs.miami.edu/~tptp/`.

provides such an integration, translating the derivation generated by the prover into the formalism of the proof-assistant.

Previous studies have reported proof-reconstruction using proof-assistants based on higher-order logic where the development is at a mature stage [32, 23, 24]. Another approaches has been proposed for proof-reconstructing based on type theory in [5, 26, 27].

We describe a formal reconstruction of proofs generated by the `Metis` prover [21]—our source system—in Martin-Löf type theory [30]. We formalize the subset of the `Metis` inference rules for the propositional logic fragment using a syntactical treatment. The `Metis` reasoning was formalized in `Agda` [43]—our target system—in two libraries [35, 37] and we implemented a proof-reconstruction tool named `Athena` [36] written in `Haskell` that is able to generate `Agda` proof-terms for `Metis` derivations. During writing this document, our formalization helps to report two programming errors in `Metis`. A bug[2] in the printing of the derivation and a soundness bug[3] in the stripping of the goal.

This paper has been organized in the following way. In Section 2, some limitations of type theory are discussed from our proof-reconstruction point of view. In Section 3, we introduce the `Metis` prover. In Section 4, we show our approach to reconstruct `Metis` derivations. Related work is described in Section 5. Conclusions and suggestions for future work are presented in Section 6.

The source code accompanying this paper (programs, libraries, and examples) is available at `GitHub`:

- The `Athena` program that translates proofs generated by `Metis` to `Agda` code: http://github.com/jonaprieto/athena.
- The `agda-prop` library as a formalization in `Agda` for classical propositional logic: http://github.com/jonaprieto/agda-prop.
- The `agda-metis` library as a formalization in `Agda` to justify `Metis` derivations of classical propositional logic: http://github.com/jonaprieto/agda-prop.

The proof-reconstruction tool `Athena` was tested with `GHC` 8.2.1. Both libraries, `agda-prop` and `agda-metis` were tested with `Agda` 2.5.3 and `Agda` standard library 0.14. `Athena` jointly with `agda-prop` and `agda-metis` are able to reconstruct propositional proofs of `Metis` 2.3 (release 20171021). We successfully reconstruct around eighty theorems in classical propositional logic from the `TPTP` collection [34] to test the developments and the formalization presented in this research.

2 Type Theory

Type theory is a formalism for the foundation of mathematics, it has became in a key instrument to study logic and proof theory that follows the same basis of constructive mathematics where the witness of a statement is *everything*. In that direction, we could say the main actors in type theory are the statements and their proofs.

By following the Curry-Howard correspondence (see, for example, [45]), a formula corresponds to a *type*, and one proof of that formula is a *term* of the correspondent type. Therefore, inhabitants types are such formulas with proofs, they are theorems.

[2] Issue No. 2 at https://github.com/gilith/metis/issues/2

[3] Issue No. 4 at https://github.com/gilith/metis/issues/4

Since type theory is a formal system, we have a syntax and a set of derivation rules. These rules enables us to derive a kind of conclusions called *judgments*. A judgment is another way to say that a term has a certain type. Each term has associated a derivation, we refer to this derivation as proof-terms.

Notation. We write the type judgments as $a : \mathsf{A}$ to denote that the term a is of type A.

We produce derivation trees inside the judgments using the derivation rules. Since type theory can be seen as typed λ-calculus with dependent function types, evaluation of λ-terms also called *normalization* of the proof-terms, is a process of reductions with the system inference rules.

Therefore, the proof verification task becomes in type theory as verifying that the proof-term has the correspondent type of the theorem. We know this process as *type-checking*. This feature of type theory allow us to verify a proof generated by an ATP by reconstructing its proof-term to type-check the proof. For such a purpose, we use a proof-assistant based on type theory like `Agda` to delegate this task. discussed in [5]. Some limitations from the type theory point of view for proof-reconstruction is described in the following sections.

2.1 Terminating functions

To reconstruct `Metis` inference rules in type theory, we observed that some rules or their inner functions are implemented by *general recursive* functions.

Functions defined by a general recursion can not be directly translated in type theory since it is not a guarantee they terminate. For that reason, we follow the technique described in [4] to avoid termination problems by modifying the recursive functions to be *structurally recursive*.

A recursive function is structurally recursive if it calls itself with only structurally smaller arguments [1]. General recursive functions can be rewrote into structurally recursive functions by using for instance, the *bounded recursion* technique. For a yet another methods, we refer the reader to [12, 1, 10].

The bounded technique defines a new structural recursive function based on the general recursive function by adding an argument. The new argument is the *bound*, a natural number given by the function complexity. In other words, the added argument will represent the number of times the function needs to call itself to get the expected outcome.

Notation. We use Prop for the type of propositions. A proposition is an expression of indivisible propositional variables (e. g., symbols $\varphi_0, \varphi_1, \cdots$), the logic constants: $\bot$, $\top$, the binary connectives $(\wedge, \vee, \supset)$, and the negation $(\neg)$. We write the syntactical equality using the symbol $(\equiv)$ as $\varphi \equiv \psi$, for $\varphi,\ \psi : \mathsf{Prop}$. For this equality, we assume the reflexivity, symmetry and transitivity properties. The type of natural numbers is called Nat, and it is defined as usual, where zero and succ are its data constructors. We use names and symbols for the arithmetic operations as usual (e. g., $+$, $-$, $*$). We use syntax sugar for zero, succ, $\mathsf{succ\ zero}$, $\ldots$, with the decimal representation 0, 1, 2, $\ldots$ as well.

Hence, one approach to define a structural recursive function based on a general recursive function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ is to formulate a new function $\mathsf{f}^* : \mathsf{A} \to \mathsf{Nat} \to \mathsf{B}$ where all of its recursive calls are done on smaller arguments.

Notation. We define some functions by pattern-matching on Prop or Nat.

Example 1. Let us consider the following example to show the bounded recursion technique for defining the `uh` function. This function is used for reconstructing a `Metis` inference rule in Section 4.2.1.

$$
\begin{aligned}
&\mathsf{uh}_0 : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{uh}_0\ (\varphi_1 \supset (\varphi_2 \supset \varphi_3)) = \mathsf{uh}_0\ ((\varphi_1 \wedge \varphi_2) \supset \varphi_3)\\
&\mathsf{uh}_0\ (\varphi_1 \supset (\varphi_2 \wedge \varphi_3))\ = \mathsf{uh}_0\ (\varphi_1 \supset \varphi_2) \wedge \mathsf{uh}_0\ (\varphi_1 \supset \varphi_3)\\
&\mathsf{uh}_0\ \varphi \qquad\qquad\qquad\quad\ = \varphi.
\end{aligned}
\tag{1}
$$

In (1), the first two equations call on arguments no structurally smaller. For instance, in the first equation,

$$\mathsf{uh}_0(\varphi_1 \supset (\varphi_2 \supset \varphi_3)) = \mathsf{uh}_0\ ((\varphi_1 \wedge \varphi_2) \supset \varphi_3),$$

the function uh_0 calls on the $(\varphi_1 \wedge \varphi_2) \supset \varphi_3$ argument in the right-hand side but this formula is not a subformula of $\varphi_1 \supset (\varphi_2 \supset \varphi_3)$. Therefore, one way to translate uh_0 to type theory is to define a new function using a bounded recursion.

$$
\begin{aligned}
&\mathsf{uh}_1 : \mathsf{Prop} \to \mathsf{Nat} \to \mathsf{Prop}\\
&\mathsf{uh}_1\ (\varphi_1 \supset (\varphi_2 \supset \varphi_3))\ (\mathsf{succ}\ n) = \mathsf{uh}_1\ ((\varphi_1 \wedge \varphi_2) \supset \varphi_3)\ n\\
&\mathsf{uh}_1\ (\varphi_1 \supset (\varphi_2 \wedge \varphi_3))\ \ (\mathsf{succ}\ n) = \mathsf{uh}_1\ (\varphi_1 \supset \varphi_2)\ n \wedge \mathsf{uh}_1\ (\varphi_1 \supset \varphi_3)\ n\\
&\mathsf{uh}_1\ \varphi \qquad\qquad\qquad\qquad n \qquad\quad = \varphi.
\end{aligned}
\tag{2}
$$

We bounded the recursion calls of the uh_0 function using its *complexity measure*.

The complexity measure of a function is the number of steps the function takes to finish. Since the uh_0 function is recursive, we can define its complexity measure by defining a recursive function instead of a closed formula for such a number. However, this function must to be structurally recursive as well and can be defined by following the pattern-matching cases of its definition.

Therefore, we define in (3) the complexity measure function of uh_0 by assigning the number of steps to finish for each pattern-matching case. We have used the complexity measure definition for a formula defined in [2] to define other complexity measures in this paper.

$$
\begin{aligned}
&\mathsf{uh}_{cm} : \mathsf{Prop} \to \mathsf{Nat}\\
&\mathsf{uh}_{cm}\ (\varphi_1 \supset (\varphi_2 \supset \varphi_3)) = \mathsf{uh}_{cm}\ \varphi_3 + 2\\
&\mathsf{uh}_{cm}\ (\varphi_1 \supset (\varphi_2 \wedge \varphi_3))\ = \mathsf{max}\ (\mathsf{uh}_{cm}\ \varphi_2)\ (\mathsf{uh}_{cm}\ \varphi_3) + 1\\
&\mathsf{uh}_{cm}\ \varphi \qquad\qquad\qquad\quad\ = 0.
\end{aligned}
\tag{3}
$$

Following the technique mentioned above, we define the function `uh`, the structural recursive definition of the function uh_0.

$$
\begin{aligned}
&\mathsf{uh} : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{uh}\ \varphi\ \ = \mathsf{uh}_1\ \varphi\ (\mathsf{uh}_{cm}\ \varphi).
\end{aligned}
$$

2.2 Intuitionistic logic

Type theory and intuitionistic logic are accompanying theories since they are based on the same philosophical basis of constructive mathematics. Proving propositions in these theories demands a witness construction for the proof. Nonetheless, classical logic does not always have constructive proofs since some of the proofs are stated by refutation.

To reconstruct proofs generated by `Metis`, we have formalized in type theory the classical propositional logic in [35]. A briefly description of it is presented in [13]. In this formalization, we have to assume the principle of excluded middle (henceforth PEM) as an axiom since `Metis` is a prover for classical logic. Assuming PEM, we can justify refutation proofs by deriving from it the *reductio ad absurdum* rule (henceforth RAA). The RAA rule is the formulation of the principle of proof by contradiction, that is, a derivation of a contradiction, $\bot$, from the hypothesis $\neg\,\varphi$, is a derivation of φ.

Notation. The LIST type is the usual inductive and parametric type for lists.

We formalise the syntactical consequence relation of CPL by an inductive family $\_\vdash\_$ with two indexes, a set of propositions (the premises) and a proposition (the conclusion), that is, $\Gamma \vdash \varphi$ represents that there is derivation with conclusion $\varphi : \mathsf{Prop}$ from the set of premises $\Gamma : \mathsf{List\ Prop}$. We implemented in [35] the syntactical consequence relation in a similar way as it was presented in [11]. For that reason, we have included structural rules like *weaken*, formation and elimination rules for connectives and the PEM axiom as the valid inference rules in Fig. 1.

$$
\frac{}{\Gamma, \varphi \vdash \varphi}\ \text{assume}
\qquad
\frac{\Gamma \vdash \varphi}{\Gamma, \psi \vdash \varphi}\ \text{weaken}
\qquad
\frac{}{\Gamma \vdash \top}\ \top\text{-intro}
\qquad
\frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\ \bot\text{-elim}
$$

$$
\frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\,\varphi}\ \neg\text{-intro}
\qquad
\frac{\Gamma \vdash \neg\,\varphi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \bot}\ \neg\text{-elim}
\qquad
\frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ \wedge\text{-intro}
$$

$$
\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ \wedge\text{-proj}_1
\qquad
\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ \wedge\text{-proj}_2
\qquad
\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ \vee\text{-intro}_1
$$

$$
\frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ \vee\text{-intro}_2
\qquad
\frac{\Gamma, \varphi \vdash \gamma \qquad \Gamma, \psi \vdash \gamma}{\Gamma, \varphi \vee \psi \vdash \gamma}\ \vee\text{-elim}
\qquad
\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \supset \psi}\ \supset\text{-intro}
$$

$$
\frac{\Gamma \vdash \varphi \supset \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ \supset\text{-elim}
\qquad
\frac{}{\Gamma \vdash \varphi \vee \neg\,\varphi}\ \text{PEM}
$$

Fig. 1. Inference rules for propositional logic.

3 Metis: Language and Proofs

`Metis` is an automatic theorem prover for first-order logic with equality developed by Hurd [21]. This prover is suitable for proof-reconstruction since it provides well-documented proofs to justify its deduction steps from the basis of only six inference rules for first-order logic (see, for example, [33, 16]). For the propositional fragment, `Metis` has three inference rules, see Fig. 2.

3.1 Input language

The `TPTP` language is the input language to encode problems used by `Metis`. It includes the first-order form (denoted by `fof`) and clause normal form (denoted by `cnf`) formats [40]. The `TPTP` syntax[4] describes a well-defined grammar to handle annotated formulas with the following form:

[4] See the complete syntax grammar at http://www.cs.miami.edu/~tptp/TPTP/SyntaxBNF.html

$$\frac{}{\Gamma \vdash \varphi}\ \text{axiom} \qquad \frac{}{\Gamma \vdash \varphi \vee \neg\, \varphi}\ \text{assume } \varphi$$

$$\frac{\Gamma \vdash \ell \vee \varphi \qquad \Gamma \vdash \neg\, \ell \vee \psi}{\Gamma \vdash \varphi \vee \psi}\ \text{resolve } \ell$$

Fig. 2. Propositional logic inference rules of the `Metis` prover.

```
language(name, role, formula).,
```

where the `language` can be `fof` or `cnf`. The `name` serves to identify the formula within the problem. Each formula assumes a `role`, this could be an `axiom`, `conjecture`, `definition`, `plain` or an `hypothesis`.

The formulas include the constants `$true` and `$false`, the negation unary operator (`~`), and the binary connectives (`&`, `|`, `=>`) to represent ($\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\supset$) respectively.

Example 2. For instance, let us express the problem $p \vdash \neg\, (p \wedge \neg\, p) \vee (q \wedge \neg\, q)$ in `TPTP` syntax. We begin by declaring the p axiom using the `axiom` keyword. Next, we include the expected conclusion using the `conjecture` keyword.

```
fof(h, axiom, p).
fof(goal, conjecture, ~ ((p & ~ p) | (q & ~ q))).
```

3.2 Output language

The `TSTP` language is an output language for derivations of ATPs [42]. A `TSTP` derivation is a directed acyclic graph, a proof tree, where each leaf is a formula from the `TPTP` input. A node is a formula inferred from the parent formulas. The root is the final derived formula, such a derivation is a list of annotated formulas with the following form:

```
language(name, role, formula, source [,useful info]).
```

The `source` field is an inference record with the following pattern:

```
inference(rule, useful info, parents).
```

The `rule` in the line above stands for the inference name; the other fields are supporting arguments or useful information to apply the reasoning step, and list the parents nodes.

Example 3. In the script below, `strip` is the name of the inference. It has no arguments and derives from one parent node named `goal`. The result of this inference when it applies to the `goal` formula is `p`.

```
fof(subgoal_0, plain, p, inference(strip, [], [goal])).
```

Notation. We adopt a customized `TSTP` syntax to keep as short as possible the `Metis` derivations for increasing the readability of this paper.

Example 4. Let us consider the following TSTP derivation using the customized TSTP syntax (see the original TSTP derivation and customizations in Appendix A).

fof(premise, axiom, p).
fof(goal, conjecture, p).
fof($\mathtt{s}_0$, p, inf(strip, goal)).
fof($\mathtt{neg}_0$, $\neg$ p, inf(negate, $\mathtt{s}_0$)).
fof($\mathtt{n}_0$, $\neg$ p, inf(canonicalize, $\mathtt{neg}_0$)).
fof($\mathtt{n}_1$, p, inf(canonicalize, premise)).
fof($\mathtt{n}_2$, $\bot$, inf(simplify, [$\mathtt{n}_0$, $\mathtt{n}_1$]))
cnf($\mathtt{r}_0$, $\bot$, inf(canonicalize, $\mathtt{n}_2$)).

3.3 Derivations

A derivation generated by Metis encodes a natural deduction proof, Fig. 3 is an example of such kind of proof. With the inference rules in Fig. 1 as the only valid deduction steps, Metis attempts to prove conjectures by refutation (i. e., *falsium* in the root of the TSTP derivation).

These derivations are directed acyclic graphs, trees of refutations. Each node stands for an application of an inference rule and the leaves in the tree represent formulas in the given problem. Each node is labeled with a name of the inference rule (e. g., canonicalize). Each edge links a premise with one conclusion. The proof graphs have at their root the conclusion $\bot$, since Metis derivations are refutations.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg p}\,\texttt{negate}}{\neg p}\,\texttt{strip} \qquad \dfrac{\dfrac{}{p}\,\texttt{axiom}}{p}\,\texttt{canonicalize}}{\bot}\,\texttt{simplify}}{\bot}\,\texttt{canonicalize}$$

Fig. 3. Metis derivation tree of Example 4.

3.4 Inference rules

We present the list of inference rules used by Metis in the TSTP derivations for propositional logic in Table 1. We reconstruct these rules in Section 4. The reader may notice that the inference rules presented in Fig 2 diverge from the rules in aforementioned table. The former rules are implemented by the latter rules in the TSTP derivations. For instance, as far as we know, in TSTP derivations, the *axiom* rule is implemented by the rules canonicalize, clausify, conjunct, and simplify.

We first present the strip inference rule since it is the rule that appears first after each conjecture. The other rules are sorted mainly follow their level of complexity of their definitions and the formalization presented in Section 4.2. Some inference rules depend on the formalizations of other rules. For instance, the simplify rule and the clausify rule need theorems developed for the canonicalize rule. The canonicalize rule needs theorems developed in the resolve section. But the resolve rule depends on the conjunct rule.

Table 1. `Metis` inference rules.

`Metis` rule	Purpose	Theorem number
`strip`	Strip a goal into subgoals	10
`conjunct`	Takes a formula from a conjunction	13
`resolve`	A general form of the resolution theorem	27
`canonicalize`	Normalization of the formula	39
`clausify`	Performs clausification	41
`simplify`	Simplify definitions and theorems	44

4 Proof-Reconstruction

The proof-reconstruction approach proposed consists of a series of steps similar to the workflow presented [39]. This process is a translation from a source system to a target system. In our case, the system of origin, the automatic theorem prover, is `Metis`; the target system is a proof-assistant, `Agda`. We choose `Agda`, but another proof-assistant with the same support of type theory and inductive types could be used.

4.1 Workflow

The overview of the proof-reconstruction is presented in Fig. 4. The process begins with a `TPTP` file that encodes a problem in CPL. We use this file as the input of the `Metis` prover. If the problem is a theorem, `Metis` generates a derivation of the proof in `TSTP` format.

With the `TSTP` derivation from Step 2, we process the derivation with the `Athena` translator tool. `Athena` parses the `TSTP` format, analyzes the derivation and generates a representation of the natural deduction proof using a tree data structure (see the properties of this tree in Section 3.3). In the `Athena` analysis, some unnecessary steps that introduce redundancies and some unused input are removed from the proof-tree. As result, we get from `Athena` an `Agda` file of the proof with names of functions and theorems from the `Agda` libraries that accompany this article: `agda-prop` and `agda-metis`.

Finally, we type-check the `Agda` proof-term. If the type-checking success, the `TSTP` derivation generated by `Metis` is correct module `Agda` and the proposed formalizations for the propositional logic and for the `Metis` reasoning. In that case `Agda` outputs an interface file. Otherwise, when type-checking fails, the failure must be investigated by the user looking at the error in the `TSTP` derivation by `Metis`, in the translation by `Athena`, in the `Agda` formalizations mentioned above or in the type-checker `Agda`.

In the remainder part of this section, a formal description using type theory is presented to build definitions and theorems of functions necessary to reconstruct `Metis` inference rules.

4.2 Reconstructing `Metis` inference rules

In this section, we reconstruct each `Metis` inference rule in Table 1 via a function and its respective theorem. We present a pattern of the formal description for these rules in Example 5.

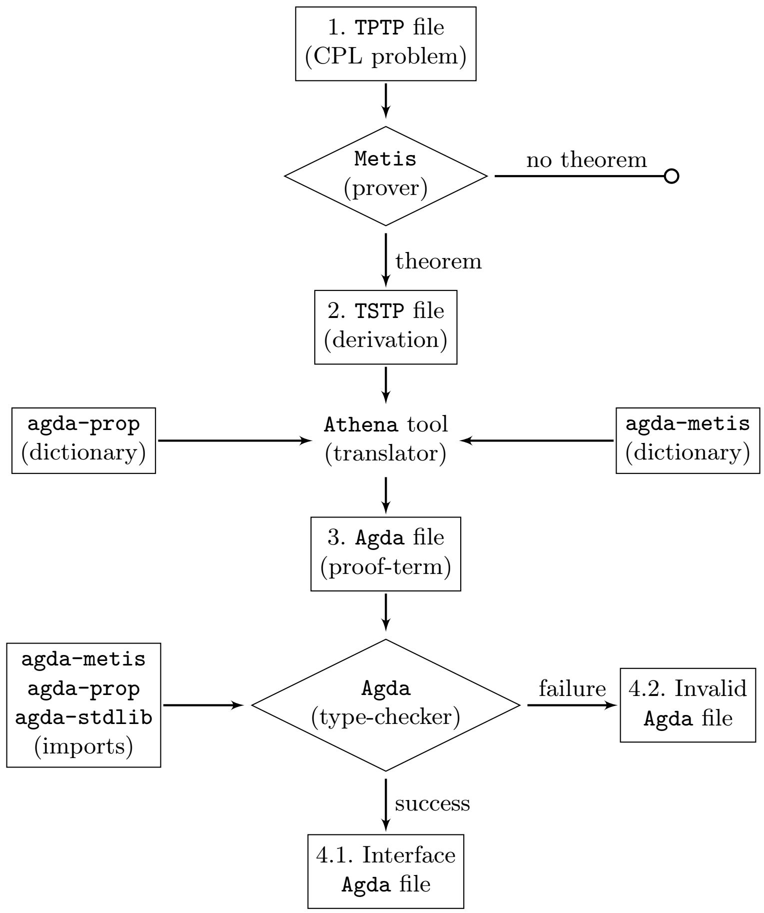

Fig. 4. Proof-reconstruction overview. The rectangles nodes represent text files. The direct edges in the diagram represent calls to programs where the input file is pointed by the edge entering and the out edge points to the output file. The rhombus nodes represent a process with two possible outcomes.

Notation. The function name written in `typewriter` font refers to a `Metis` inference rule. The same function name written using sans serif font refers to our formalized version to reconstruct the rule and implemented in [37]. We use $\mathsf{Premise}$ and $\mathsf{Conclusion}$ as synonyms of the Prop type to describe in the function types the role of the arguments.

Example 5. Let `metisRule` be a `Metis` inference rule. To reconstruct this rule, we define the function $\mathsf{metisRule}$ in type theory that follows the pattern:

$$\mathsf{metisRule} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}$$

$$\mathsf{metisRule}\ \varphi\ \psi = \begin{cases} \psi, & \text{if the conclusion } \psi \text{ can be derived by applying certain inference} \\ & \text{rules to the premise } \varphi; \\ \varphi, & \text{otherwise;} \end{cases}$$

To justify all transformations done by the metisRule rule, we prove its soundness with a theorem like the following:

$$\text{If } \Gamma \vdash \varphi \text{ then } \Gamma \vdash \mathsf{metisRule}\ \varphi\ \psi, \text{ where } \psi : \mathsf{Conclusion}.$$

The remainder of this section will be devoted to present a formal description in type theory of each rule presented in Table 1 using the pattern present in the example above. We follow the same order to present the rules as the table shows.

4.2.1 Strip.

To prove a goal, `Metis` splits the goal into disjoint cases. This process produces a list of new subgoals, the conjunction of these subgoals implies the goal. Then, a proof of the goal becomes, in smaller proofs, one refutation for each subgoal.

Example 6. The subgoals associated to a goal are introduced in the `TSTP` derivation with the `strip` inference rule.

```
fof(goal, conjecture, (p ∧ r) ∧ q)).
fof(s₁, p, inf(strip, goal)).
fof(s₂, p ⊃ r, inf(strip, goal)).
fof(s₃, (p ∧ r) ⊃ q, inf(strip, goal)).
```

In this example, the conjecture $(p \wedge r) \wedge q$ is stripped into tree subgoals: p, $p \supset r$ and $(p \wedge r) \supset q$.

$$(p \wedge (p \supset r) \wedge ((p \wedge r) \supset q)) \supset ((p \wedge r) \wedge q). \tag{4}$$

`Metis` proves each subgoal in the same order above from left to right in (4). So far, very little attention has been paid to the role of the `strip` rule in `TSTP` derivations since `Metis` does not make explicit the way it uses the subgoals to prove the conjecture.

We prove the correctness of the `strip` inference rule in Theorem 10. To show that theorem, we need to prove Lemma 7 and Lemma 8.

Lemma 7. Let n : Nat be the complexity measure of the uh_0 function in (1). If $\Gamma \vdash \mathsf{uh}_1\ \varphi\ n$ then $\Gamma \vdash \varphi$ where uh_1 is the function defined in (2).

Proof. The proof is by induction on the cases defined by the outcome of the uh_1 function.

- If $n = 0$, by definition in (2) we conclude $\Gamma \vdash \varphi$.
- For $n \geq 1$, we use induction on the structure of the first argument.
  - Case $\varphi \equiv \varphi_1 \supset (\varphi_2 \supset \varphi_3)$.

$$\cfrac{\cfrac{\cfrac{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset (\varphi_2 \supset \varphi_3))\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}_1\ ((\varphi_1 \wedge \varphi_2) \supset \varphi_3)\ n}\ \text{by (2)}}{\Gamma \vdash (\varphi_1 \wedge \varphi_2) \supset \varphi_3}\ \text{by ind. hyp.}}{\Gamma \vdash \varphi_1 \supset (\varphi_2 \supset \varphi_3)}\ \wedge\!\supset\!\text{-to-}\!\supset\supset$$

  using the following theorem proved in [35],

$$\wedge\!\supset\!\text{-to-}\!\supset\supset\ :\ \Gamma \vdash (\varphi_1 \wedge \varphi_2) \supset \varphi_3 \to \Gamma \vdash \varphi_1 \supset (\varphi_2 \supset \varphi_3).$$

- Case $\varphi \equiv \varphi_1 \supset (\varphi_2 \wedge \varphi_3)$.

$$(\mathcal{D}_1)\quad \dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset (\varphi_2 \wedge \varphi_3))\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset \varphi_2)\ n \wedge \mathsf{uh}_1\ (\varphi_1 \supset \varphi_3)\ n}\ \text{by (2)}}{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset \varphi_2)\ n}\ \wedge\text{-proj}_1}{\Gamma \vdash \varphi_1 \supset \varphi_2}\ \text{by ind. hyp.}$$

$$(\mathcal{D}_2)\quad \dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset (\varphi_2 \wedge \varphi_3))\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset \varphi_2)\ n \wedge \mathsf{uh}_1\ (\varphi_1 \supset \varphi_3)\ n}\ \text{by (2)}}{\Gamma \vdash \mathsf{uh}_1\ (\varphi_1 \supset \varphi_3)\ n}\ \wedge\text{-proj}_2}{\Gamma \vdash \varphi_1 \supset \varphi_3}\ \text{by ind. hyp.}$$

  Now, using $\supset\wedge\supset$-to-$\supset\wedge$ theorem from [35],

$$\supset\wedge\supset\text{-}\mathsf{to}\text{-}\supset\wedge\ :\ \Gamma \vdash (\varphi_1 \supset \varphi_2) \wedge (\varphi_1 \supset \varphi_3) \to \Gamma \vdash \varphi_1 \supset (\varphi_2 \wedge \varphi_3),$$

$$\dfrac{\dfrac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma \vdash (\varphi_1 \supset \varphi_2) \wedge (\varphi_1 \supset \varphi_3)}\ \wedge\text{-intro}}{\Gamma \vdash \varphi_1 \supset (\varphi_2 \wedge \varphi_3)}\ \supset\wedge\supset\text{-}\mathsf{to}\text{-}\supset\wedge$$

- Other cases are proved in a similar way.

■

The strip_0 function defined in (25) yields the conjunction of subgoals that implies the goal of the problem in the `Metis` `TSTP` derivations. Nonetheless, this function is not a structurally recursive function (see more details in Appendix B). Therefore, we present the strip_1 function in (5) as the structurally recursive version of the strip_0 function by applying the bounded technique described in Section 2.1. We define the strip_1 function based on the reading of the `Metis` source code.

$$\begin{aligned}
&\mathsf{strip}_1 : \mathsf{Prop} \to \mathsf{Nat} \to \mathsf{Prop}\\
&\mathsf{strip}_1\ (\varphi_1 \wedge \varphi_2)\quad (\mathsf{succ}\ n) = \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n) \wedge \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n)\\
&\mathsf{strip}_1\ (\varphi_1 \vee \varphi_2)\quad (\mathsf{succ}\ n) = \mathsf{uh}\ ((\neg\,\varphi_1) \supset \mathsf{strip}_1\ \varphi_2\ n)\\
&\mathsf{strip}_1\ (\varphi_1 \supset \varphi_2)\quad (\mathsf{succ}\ n) = \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n)\\
&\mathsf{strip}_1\ (\neg\,(\varphi_1 \wedge \varphi_2))\ (\mathsf{succ}\ n) = \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n)\\
&\mathsf{strip}_1\ (\neg\,(\varphi_1 \vee \varphi_2))\ (\mathsf{succ}\ n) = \mathsf{uh}\ (\mathsf{strip}_1\ (\neg\,\varphi_1)\ n) \wedge \mathsf{uh}\ ((\neg\,\varphi_1) \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n)\\
&\mathsf{strip}_1\ (\neg\,(\varphi_1 \supset \varphi_2))\ (\mathsf{succ}\ n) = \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n) \wedge \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n)\\
&\mathsf{strip}_1\ (\neg\,(\neg\,\varphi_1))\quad (\mathsf{succ}\ n) = \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n)\\
&\mathsf{strip}_1\ (\neg\,\bot)\quad (\mathsf{succ}\ n) = \top\\
&\mathsf{strip}_1\ (\neg\,\top)\quad (\mathsf{succ}\ n) = \bot\\
&\mathsf{strip}_1\ \varphi\quad n = \varphi.
\end{aligned} \tag{5}$$

In a similar way as we define uh_{cm} in (3), we define the strip_{cm} function in Appendix B as the complexity measure for the strip_0 function. Then we define the strip function as follows in 6.

$$\begin{aligned}
&\mathsf{strip} : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{strip}\ \varphi\ = \mathsf{strip}_1\ \varphi\ (\mathsf{strip}_{cm}\ \varphi).
\end{aligned} \tag{6}$$

Lemma 8. Let n : Nat be the complexity measure of the strip function defined in (5). If $\Gamma \vdash \mathsf{strip}_1\ \varphi\ n$ then $\Gamma \vdash \varphi$.

Proof. The proof is by induction on the structure of the formula φ by following the equations in (5). We present a straightforward case with double negation, the case for conjunctions, and last, the case for a negated disjunction. We refer the reader to [37] for the complete proof in `Agda`.

- Case $\varphi \equiv \neg\,(\neg\,\varphi_1)$.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{strip}_1\ (\neg\,(\neg\,\varphi_1))\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n)}\ \text{by (5)}}{\Gamma \vdash \mathsf{strip}_1\ \varphi_1\ n}\ \text{Lemma 7}}{\Gamma \vdash \varphi_1}\ \text{by ind. hyp.}$$

- Case $\varphi \equiv \varphi_1 \wedge \varphi_2$. We prove $\Gamma \vdash \varphi_1$ and $\Gamma \vdash \varphi_2$. From the conjunction of φ_1 and φ_2, the expected result follows.

$$(\mathcal{D})\quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{strip}_1\ (\varphi_1 \wedge \varphi_2)\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n) \wedge \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n)}\ \text{by (5)}}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n)}\ \wedge\text{-proj}_1}{\Gamma \vdash \mathsf{strip}_1\ \varphi_1\ n}\ \text{Lemma 7}}{\Gamma \vdash \varphi_1}\ \text{by ind. hyp.}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash \varphi_1} \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{strip}_1\ (\varphi_1 \wedge \varphi_2)\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ \varphi_1\ n) \wedge \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n)}\ \text{by (5)}}{\Gamma \vdash \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n)}\ \wedge\text{-proj}_2}{\Gamma \vdash \varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n}\ \text{Lemma 7}}{\Gamma \vdash \mathsf{strip}_1\ \varphi_2\ n}\ \supset\text{-elim}}{\Gamma \vdash \varphi_2}\ \text{by ind. hyp.}$$

- Case $\varphi \equiv \neg\,(\varphi_1 \vee \varphi_2)$. We prove $\Gamma \vdash \neg\,\varphi_1$ and $\Gamma \vdash \neg\,\varphi_2$. From the conjunction of $\neg\,\varphi_1$ and $\neg\,\varphi_2$ by applying De Morgan Law and the result follows.

$$(\mathcal{D})\quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{strip}_1\ (\neg\,(\varphi_1 \vee \varphi_2))\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ (\neg\,\varphi_1)\ n) \wedge \mathsf{uh}\ ((\neg\,\varphi_1) \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n)}\ \text{by (5)}}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ (\neg\,\varphi_1)\ n)}\ \wedge\text{-proj}_1}{\Gamma \vdash \mathsf{strip}_1\ (\neg\,\varphi_1)\ n}\ \text{Lemma 7}}{\Gamma \vdash \neg\,\varphi_1}\ \text{by ind. hyp.}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash \neg\,\varphi_1} \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{strip}_1\ (\neg\,(\varphi_1 \vee \varphi_2))\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}\ (\mathsf{strip}_1\ (\neg\,\varphi_1)\ n) \wedge \mathsf{uh}\ ((\neg\,\varphi_1) \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n)}\ \text{by (5)}}{\Gamma \vdash \mathsf{uh}\ ((\neg\,\varphi_1) \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n)}\ \wedge\text{-proj}_2}{\Gamma \vdash (\neg\,\varphi_1) \supset \mathsf{strip}_1\ (\neg\,\varphi_2)\ n}\ \text{Lemma 7}}{\Gamma \vdash \mathsf{strip}_1\ (\neg\,\varphi_2)\ n}\ \supset\text{-elim}}{\Gamma \vdash \neg\,\varphi_2}\ \text{by ind. hyp.}$$

- Other cases are proved in a similar way, see Appendix C.

■

The following theorem is convenient to substitute equals by equals in a theorem. Recall the equality ($\equiv$) is symmetric and transitive as well. We use these properties without any mention.

Lemma 9. Substitution theorem.

$$\dfrac{\Gamma \vdash \varphi \qquad \psi \equiv \varphi}{\Gamma \vdash \psi}\ \mathsf{subst}$$

We can now formulate the result that justifies the stripping strategy of `Metis` to prove goals. For the sake of brevity, we state the following theorem for the `strip` function when the goal has two subgoals. In other cases, we extend the theorem in the natural way.

Theorem 10. Let n : Nat be the complexity measure of the strip function defined in (5). Let s_2 and s_3 be the subgoals of the goal φ, that is,

$$\mathsf{strip}_1\ \varphi\ n \ \equiv\ s_2 \wedge\ s_3.$$

If $\Gamma \vdash s_2$ and $\Gamma \vdash s_3$ then $\Gamma \vdash \varphi$.

Proof.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash s_1 \qquad \Gamma \vdash s_2}{\Gamma \vdash s_1 \wedge s_2}\ \wedge\text{-intro} \qquad \mathsf{strip}_1\ \varphi\ n \equiv s_1 \wedge s_2}{\Gamma \vdash \mathsf{strip}_1\ \varphi\ n}\ \mathsf{subst}}{\Gamma \vdash \varphi}\ \text{Lemma 8}$$

■

Since `Metis` proves a conjecture by refutation, to prove each subgoal, `Metis` assumes the negation of it by using the `negate` rule after the `strip` inference application that introduce such a subgoal.

Example 11. In the following `TSTP` derivation, note that the three subgoals $\mathtt{s}_1$, $\mathtt{s}_2$ and $\mathtt{s}_3$ are assumed by the `negate` rule in $\mathtt{neg}_1$, $\mathtt{neg}_2$ and $\mathtt{neg}_3$ respectively.

```
fof(goal, conjecture, p ∧ r ∧ q).
fof(s1, p, inf(strip, goal)).
fof(s2, p ⊃ r, inf(strip, goal)).
fof(s3, (p ∧ r) ⊃ q, inf(strip, goal)).
fof(neg1, ¬ p, inf(negate, s1)).
...
fof(neg2, ¬ (p ⊃ r), inf(negate, s2)).
...
fof(neg3, ¬ ((p ∧ r) ⊃ q), inf(negate, s3)).
```

4.2.2 Conjunct.

The `conjunct` rule extracts from a conjunction one of its conjuncts. This rule is a generalization of the projection rules for the conjunction connective as the following `TSTP` excerpt shows.

Example 12.

```
fof(p₁, p ∧ q ∧ (r ∨ ¬ p), ...
fof(p₂, q, inf(conjunct, p₁)).
fof(p₃, r ∨ ¬ p, inf(conjunct, p₁)).
```

In the first formula, $p \wedge q \wedge (r \vee \neg\, p)$, we find a left-associative conjunction named $\mathtt{p}_1$. The `conjunct` rule extracts q from the $\mathtt{p}_1$ using a left projection ($\wedge$-proj$_1$) follow by a right projection ($\wedge$-proj$_2$). After, the `conjunct` rule extracts $r \vee \neg\, p$ by using a right projection on $\mathtt{p}_1$.

Theorem 13. Let ψ : Conclusion. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{conjunct}\ \varphi\ \psi$, where

$$\mathsf{conjunct} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}$$

$$\mathsf{conjunct}\ \varphi\ \psi = \begin{cases} \psi, & \text{if } \varphi \equiv \psi; \\ \psi, & \text{if } \varphi \equiv \varphi_1 \wedge \varphi_2 \text{ and } \psi \equiv \mathsf{conjunct}\ \varphi_1\ \psi; \\ \psi, & \text{if } \varphi \equiv \varphi_1 \wedge \varphi_2 \text{ and } \psi \equiv \mathsf{conjunct}\ \varphi_2\ \psi; \\ \varphi, & \text{otherwise.} \end{cases}$$

Proof.

- Case $\varphi \equiv \psi$. $\Gamma \vdash \mathsf{conjunct}\ \varphi\ \psi$ normalizes to $\Gamma \vdash \psi$. Then, we get the desire conclusion by applying the subst lemma.
- Case $\varphi \equiv \varphi_1 \wedge \varphi_2$. If we can get $\psi \equiv \mathsf{conjunct}\ \varphi_i\ \psi$ for some $i = 1,\ 2$, then,

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_i}\ \wedge\text{-proj}_i}{\Gamma \vdash \mathsf{conjunct}\ \varphi_i\ \psi}\ \text{by ind. hyp.} \qquad \psi \equiv \mathsf{conjunct}\ \varphi_i\ \psi}{\Gamma \vdash \psi}\ \mathsf{subst}$$

- Otherwise, the last case follows from the hypothesis.

∎

4.2.3 Resolve.

Logic equivalence between propositions is a major issue to justify prover reasoning steps. Since we left out semantics to treat only the syntax aspects of propositional logic, our approach shows logic equivalence by converting propositions to their conjunctive normal form, and reordering those and the inner disjunctions to match them. Below, we provide Lemma 19, Lemma 20, Lemma 23 to perform such reordering tasks, the omitted proofs can be found in [37].

First, we define the $\mathsf{assoc}_{\Box}$ function in (7) to convert a disjunction or a conjunction into its right-associative form. The square symbol ($\Box$) can be the conjunction symbol or the disjunction symbol. We use $\mathsf{assoc}_{\wedge}$ to convert conjunctions and $\mathsf{assoc}_{\vee}$ for disjunctions.

$$\begin{aligned} &\mathsf{assoc}_{\Box} : \mathsf{Prop} \to \mathsf{Prop} \\ &\mathsf{assoc}_{\Box}\ ((\varphi_1 \Box \varphi_2) \Box \varphi_3) = \mathsf{assoc}_{\Box}\ (\varphi_1 \Box (\varphi_2 \Box \varphi_3)) \\ &\mathsf{assoc}_{\Box}\ (\varphi_1 \Box \varphi_2) \qquad\quad\ = \varphi_1 \Box \mathsf{assoc}_{\Box}\ \varphi_2 \\ &\mathsf{assoc}_{\Box}\ \varphi \qquad\qquad\qquad\ \ = \varphi. \end{aligned} \tag{7}$$

Lemma 14. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{assoc}_{\square}\ \varphi$.

Remark. In `TPTP` syntax, the formulas are in left-associative form by default. Despite of that convention, `Metis` assumes the formulas to be in right-associative form by default. This is a matter to take into account for the proof-reconstruction.

The $\mathsf{build}_{\vee}$ function defined in (8) intends to construct a disjunction from another disjunction, specifically, this function will try to rearrange the disjuncts in the source formula to match with the target disjunction formula.

Lemma 15. If $\Gamma \vdash \varphi$ and $\psi : \mathsf{Conclusion}$ then $\Gamma \vdash \mathsf{build}_{\vee}\ \varphi\ \psi$, where

$$
\begin{aligned}
&\mathsf{build}_{\vee} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}\\
&\mathsf{build}_{\vee}\ \varphi\ \psi\ \ = \begin{cases} \psi, & \text{if } \varphi \equiv \psi;\\ \psi, & \text{if } \psi \equiv \psi_1 \vee \psi_2 \text{ and } \psi_i \equiv \mathsf{build}_{\vee}\ \varphi\ \psi_i \text{ for some } i = 1,2;\\ \varphi, & \text{otherwise.}\end{cases}
\end{aligned} \tag{8}
$$

From now on, we assume all propositions to be right-associative unless otherwise stated.

The factor function in (9) simplifies a special case of disjunction, the repeated disjuncts (e. g., $\mathsf{factor}\ (\varphi \vee \varphi)\ =\ \varphi$). Notice that other cases like $\varphi\ \vee\ (\psi\ \vee\ \varphi)$ do not reduce to $\psi\ \vee\ \varphi$. We use this function in Lemma 17.

Lemma 16. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{factor}\ \varphi$, where

$$
\begin{aligned}
&\mathsf{factor} : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{factor}\ \varphi\ \ = \begin{cases} \varphi_1, & \text{if } \varphi \equiv \varphi_1 \vee \varphi_2 \text{ and } \varphi_1 \equiv \mathsf{factor}\ \varphi_2;\\ \varphi, & \text{otherwise.}\end{cases}
\end{aligned} \tag{9}
$$

To construct a disjunction ψ from a disjunction φ, we have used ideas from the description in [8] to prove equality between nested disjunctions. We define the $\mathsf{sbuild}_{\vee}$ function in (10) that uses every disjunct from the source formula, φ, to build up the target disjunction ψ.

Lemma 17. If $\Gamma \vdash \varphi$ and $\psi : \mathsf{Conclusion}$ then $\Gamma \vdash \mathsf{sbuild}_{\vee}\ \varphi\ \psi$, where

$$
\begin{aligned}
&\mathsf{sbuild}_{\vee} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}\\
&\mathsf{sbuild}_{\vee}\ (\varphi_1 \vee \varphi_2)\ \psi = \mathsf{factor}\ (\mathsf{build}_{\vee}\ \varphi_1\ \psi \vee \mathsf{build}_{\vee}\ \varphi_2\ \psi)\\
&\mathsf{sbuild}_{\vee}\ \varphi\ \psi \qquad\qquad\ = \mathsf{build}_{\vee}\ \varphi\ \psi.
\end{aligned} \tag{10}
$$

Example 18. We build the disjunction $(p \vee q) \vee r$ from the disjunction $r \vee (q \vee p)$ using Lemma 17.

$$
(\mathcal{D})\quad \dfrac{\dfrac{\dfrac{\Gamma \vdash q}{\Gamma \vdash p \vee q}\ \vee\text{-intro}_2}{\Gamma \vdash (p \vee q) \vee r}\ \vee\text{-intro}_1 \qquad \dfrac{\dfrac{\Gamma \vdash p}{\Gamma \vdash p \vee q}\ \vee\text{-intro}_1}{\Gamma \vdash (p \vee q) \vee r}\ \vee\text{-intro}_1}{\Gamma, q \vee p \vdash (p \vee q) \vee r}\ \vee\text{-elim}
$$

$$
\dfrac{\dfrac{\dfrac{\Gamma \vdash r}{\Gamma \vdash (p \vee q) \vee r}\ \vee\text{-intro}_2 \qquad \dfrac{\mathcal{D}}{\Gamma, q \vee p \vdash (p \vee q) \vee r}}{\Gamma, r \vee (q \vee p) \vdash (p \vee q) \vee r}\ \vee\text{-elim}}{\Gamma \vdash r \vee (q \vee p) \supset (p \vee q) \vee r}\ \supset\text{-intro}
$$

Remark. Notice that using $\mathsf{sbuild}_\vee$ we can build not only a disjunction with the same disjuncts of the source formula but also a complete different disjunction by adding new disjuncts to the source formula via introduction rules for disjunctions.

The following lemma aims to reorder nested disjunctions by forcing the formula to be in right-associative form in order to apply Lemma 17.

Lemma 19. If $\Gamma \vdash \varphi$ and $\psi : \mathsf{Conclusion}$ then $\Gamma \vdash \mathsf{reorder}_\vee\ \varphi\ \psi$, where

$$\begin{aligned} &\mathsf{reorder}_\vee : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop} \\ &\mathsf{reorder}_\vee\ \varphi\ \psi\ \ = \mathsf{sbuild}_\vee\ (\mathsf{assoc}_\vee\ \varphi)\ \psi. \end{aligned} \tag{11}$$

Proof. Use Lemma 14 and Lemma 17. ∎

Now, we define the $\mathsf{reorder}_\wedge$ function in (12) to reorder nested conjunctions. This will help us in the end of this section to reorder conjunctive normal forms.

Lemma 20. If $\Gamma \vdash \varphi$ and $\psi : \mathsf{Conclusion}$ then $\Gamma \vdash \mathsf{reorder}_\wedge\ \varphi\ \psi$, where

$$\begin{aligned} &\mathsf{reorder}_\wedge : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop} \\ &\mathsf{reorder}_\wedge\ \varphi\ \psi = \begin{cases} \varphi, & \text{if } \varphi \equiv \psi; \\ \psi_1 \wedge \psi_2, & \text{if } \psi \equiv \psi_1 \wedge \psi_2,\ \psi_1 \equiv \mathsf{reorder}_\wedge\ \varphi\ \psi_1; \\ & \text{and } \psi_2 \equiv \mathsf{reorder}_\wedge\ \varphi\ \psi_2; \\ \varphi, & \text{if } \psi \equiv \psi_1 \wedge \psi_2; \\ \mathsf{conjunct}\ \varphi\ \psi, & \text{otherwise.} \end{cases} \end{aligned} \tag{12}$$

Example 21.

$$\begin{aligned} &\mathsf{reorder}_\wedge\ (p \wedge q \wedge r) \quad (r \wedge q \wedge p) &&= (r \wedge q \wedge p), \\ &\mathsf{reorder}_\wedge\ (p \wedge q \wedge r) \quad (r \wedge r \wedge p) &&= (r \wedge q \wedge p), \\ &\mathsf{reorder}_\wedge\ (p \wedge q \wedge r) \quad (k \wedge q \wedge p) &&= (p \wedge q \wedge r), \\ &\mathsf{reorder}_\wedge\ ((p \vee q) \wedge r)\ ((r \wedge (q \vee p)) &&= ((p \vee q) \wedge r). \end{aligned} \tag{13}$$

In the last example in (13), we could not build the conjunction $r \wedge (q \vee p)$ since $p \vee q$ is not syntactical equal to $q \vee p$. We solve this issue in Lemma 23 by using the disj function defined in (14). The purpose of this function consists of extracting a disjunction from a conjunction, but without matter the order of the inner disjunctions.

Lemma 22. If $\Gamma \vdash \varphi$ and $\psi : \mathsf{Conclusion}$ then $\Gamma \vdash \mathsf{disj}\ \varphi\ \psi$, where

$$\begin{aligned} &\mathsf{disj} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop} \\ &\mathsf{disj}\ \varphi\ \psi = \begin{cases} \psi, & \text{if } \varphi \equiv \psi; \\ \psi, & \text{if } \psi \equiv \mathsf{reorder}_\vee\ \varphi\ \psi; \\ \psi, & \text{if } \psi \equiv \psi_1 \wedge \psi_2,\ \ \psi_1 \equiv \mathsf{disj}\ \varphi\ \psi_1, \\ & \text{and } \psi_2 \equiv \mathsf{reorder}_\vee\ \varphi\ \psi_2; \\ \psi, & \text{if } \varphi \equiv \varphi_1 \wedge \varphi_2,\ \ \psi \equiv \mathsf{disj}\ \varphi_1\ \psi; \\ \psi, & \text{if } \varphi \equiv \varphi_1 \wedge \varphi_2,\ \ \psi \equiv \mathsf{disj}\ \varphi_2\ \psi; \\ \varphi, & \text{otherwise.} \end{cases} \end{aligned} \tag{14}$$

We are able now to reorder conjunctive normal forms using the $\mathsf{reorder}_{\wedge\vee}$ function defined in (15) by using the previous lemma.

Lemma 23. If $\Gamma \vdash \varphi$ and $\psi : \mathsf{Conclusion}$ then $\Gamma \vdash \mathsf{reorder}_{\wedge\vee}\ \varphi\ \psi$, where

$$
\begin{aligned}
&\mathsf{reorder}_{\wedge\vee} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}\\
&\mathsf{reorder}_{\wedge\vee}\ \varphi\ \psi = \begin{cases} \psi, & \text{if } \varphi \equiv \psi;\\ \psi, & \text{if } \psi \equiv \psi_1 \wedge \psi_2,\ \psi_1 \equiv \mathsf{reorder}_{\wedge\vee}\ \varphi\ \psi_1,\\ & \text{and } \psi_2 \equiv \mathsf{reorder}_{\wedge\vee}\ \varphi\ \psi_2;\\ \varphi, & \text{if } \psi \equiv \psi_1 \wedge \psi_2;\\ \mathsf{disj}\ \varphi\ \psi, & \text{otherwise.} \end{cases}
\end{aligned}
\tag{15}
$$

Now, we are ready to reconstruct the `resolve` rule using Lemma 23. As we see in the following, the `resolve` rule is the `Metis` version of the resolution theorem. This rule takes into account, two propositions that contain a positive literal ℓ and its negation $\neg\,\ell$ respectively. Then, it produces the *resolvent*, a disjunction of two propositions: the first proposition after removing the literal ℓ and the second proposition after removing its negation $\neg\,\ell$.

Definition 24. A *literal* is an propositional variable (positive literal) or a negation of an propositional variable (negative literal).

Notation. We use Lit as synonym of Prop type to refer literals.

The positive literal ℓ must occur in the formula from the first derivation and the negative literal $\neg\,\ell$ must occur in the formula from the second derivation, see the pattern of the *resolve* rule in Fig. 2.

Example 25.

```
cnf(r₄, ¬ r ∨ p ∨ q, ...
cnf(r₅, p ∨ q ∨ r, ...
cnf(r₆, p ∨ q, inf(resolve, r, [r₅, r₄])).
```

In the excerpt above, we apply resolution to the first two formulas, $\neg\, r \vee p \vee q$ and $p \vee q \vee r$. The last line tells us the literal used for resolution is r. Syntactically speaking, we can not derive neither the conclusion $p \vee q$ in $\mathtt{r}_6$ nor apply the resolution theorem with $\mathtt{r}_4$ and $\mathtt{r}_5$ since the formulas do not fit the pattern required.

If the scenario would have other like replacing $\mathtt{r}_5$ by

```
cnf(r₅, r ∨ p ∨ q, ...
```

The `resolve` rule have could derive $(p \vee q) \vee (p \vee q)$, but again, that is not the expected result.

Therefore, we perform a sequence of rearrangements inside the involved formulas to match with the expected pattern by the *resolve* inference rule in Fig. 2.

Using reordering after applying a customized version of the resolution theorem defined in (16) we get the expected result.

Lemma 26. If $\mathit{\Gamma} \vdash \varphi$ then $\mathit{\Gamma} \vdash \mathsf{rsol}\ \varphi$, where

$$\begin{aligned}
&\mathsf{rsol} : \mathsf{Prop} \to \mathsf{Prop} \\
&\mathsf{rsol}\ \varphi = \begin{cases} \varphi_2, & \text{if } \varphi \equiv (\varphi_1 \vee \varphi_2) \wedge (\neg\, \varphi_1 \vee \varphi_2); \\ \varphi_2 \vee \varphi_4, & \text{if } \varphi \equiv (\varphi_1 \vee \varphi_2) \wedge (\neg\, \varphi_1 \vee \varphi_4); \\ \varphi, & \text{otherwise.} \end{cases}
\end{aligned} \tag{16}$$

Theorem 27. Let ℓ be a literal, ℓ : Lit, and ψ : $\mathsf{Conclusion}$. If $\mathit{\Gamma} \vdash \varphi_1$ and $\mathit{\Gamma} \vdash \varphi_2$ then $\mathit{\Gamma} \vdash \mathsf{resolve}\ \varphi_1\ \varphi_2\ \ell\ \psi$, where

$$\begin{aligned}
&\mathsf{resolve} : \mathsf{Premise} \to \mathsf{Premise} \to \mathsf{Lit} \to \mathsf{Conclusion} \to \mathsf{Prop} \\
&\mathsf{resolve}\ \varphi_1\ \varphi_2\ \ell\ \psi = \mathsf{rsol}\ (\mathsf{reorder}_\vee\ \varphi_1\ (\ell \vee \psi) \wedge \mathsf{reorder}_\vee\ \varphi_2\ (\neg\, \ell \vee \psi)).
\end{aligned} \tag{17}$$

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{\Gamma} \vdash \varphi_1}{\mathit{\Gamma} \vdash \mathsf{reorder}_\vee\ \varphi_1\ (\ell \vee \psi)}\ \text{Lemma 23} \qquad \dfrac{\mathit{\Gamma} \vdash \varphi_2}{\mathit{\Gamma} \vdash \mathsf{reorder}_\vee\ \varphi_2\ (\neg\, \ell \vee \psi)}\ \text{Lemma 23}}{\mathit{\Gamma} \vdash \mathsf{reorder}_\vee\ \varphi_1\ (\ell \vee \psi) \wedge \mathsf{reorder}_\vee\ \varphi_2\ (\neg\, \ell \vee \psi)}\ \wedge\text{-intro}}{\mathit{\Gamma} \vdash \mathsf{rsol}\ (\mathsf{reorder}_\vee\ \varphi_1\ (\ell \vee \psi) \wedge \mathsf{reorder}_\vee\ \varphi_2\ (\neg\, \ell \vee \psi))}\ \text{Lemma 26}}{\mathit{\Gamma} \vdash \mathsf{resolve}\ \varphi_1\ \varphi_2\ \ell\ \psi}\ \text{by (17)}$$

■

Example 28. Continuing with the problem presented in Example 25, we can use Theorem 27 to derive $\mathit{\Gamma} \vdash p \vee q$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{\Gamma} \vdash p \vee q \vee r \qquad \mathit{\Gamma} \vdash \neg\, r \vee p \vee q}{\mathit{\Gamma} \vdash \mathsf{resolve}\ (p \vee q \vee r)\ (\neg\, r \vee p \vee q)\ r\ (p \vee q)}\ \text{Theorem 27}}{\mathit{\Gamma} \vdash \mathsf{rsol}\ (\mathsf{reorder}_\vee\ (p \vee q \vee r)\ (r \vee (p \vee q))\ \wedge\ \mathsf{reorder}_\vee\ (\neg\, r \vee p \vee q)\ (\neg\, r \vee (p \vee q))}\ \text{by (17)}}{\mathit{\Gamma} \vdash \mathsf{rsol}\ ((r \vee (p \vee q)) \wedge (\neg\, r \vee (p \vee q)))}\ \text{by (11)}}{\mathit{\Gamma} \vdash p \vee q}\ \text{by (16)}$$

4.2.4 Canonicalize.

The `canonicalize` rule is an inference that transforms a formula to a its negative normal form or its conjunctive normal form depending on the role that the formula plays in the problem as we will explain in a moment. However, in both cases, this rule removes inside of the formula any redundancy as long as possible (i. e., tautologies or definitions).

Definition 29. The *negative normal form* of a formula is the logical equivalent version of it in which negations appear only in the literals and the formula does not contain any implications.

Definition 30. The *conjunctive normal form* of a formula also called clausal normal form is the logical equivalent version expressed as a conjunction of clauses where a *clause* is the disjunction of zero or more literals.

The `canonicalize` rule is used by `Metis` to introduce the subgoals in their refutation proofs but also helps to simplify formulas at intermediate steps in the derivations. As far as we know, the `canonicalize` rule implements a conjunctive normal form conversion with simplifications of tautologies and definitions. Otherwise, when an axiom, definition or hypothesis is needed to prove some goal, this rule gets the negative normal form of the formula. The `canonicalize` rule jointly with the `clausify` rule perform the so-called *clausification* process mainly described in [41].

To reconstruct the `canonicalize` rule, we adapted some ideas from the `Metis` source code. The presentation of this reconstruction is as follows. We firstly describe functions to remove redundancies inside of the formula. After, we present the negative normal form conversion in Lemma 34 and the conjunctive normal form in Lemma 36. At the end of this section, we state Theorem 39 to reconstruct the `canonicalize` rule.

Now, we say that there are redundancies in a formula when some of the theorems in Fig. 5 can be applied inside of it.

$$\frac{\Gamma \vdash \varphi \vee \bot}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \vee \top}{\Gamma \vdash \top} \qquad \frac{\Gamma \vdash \varphi \vee \neg\, \varphi}{\Gamma \vdash \top} \qquad \frac{\Gamma \vdash \varphi \vee \varphi}{\Gamma \vdash \varphi}$$

$$\frac{\Gamma \vdash \varphi \wedge \bot}{\Gamma \vdash \bot} \qquad \frac{\Gamma \vdash \varphi \wedge \top}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \wedge \varphi}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \wedge \neg\, \varphi}{\Gamma \vdash \bot}$$

Fig. 5. Theorems to remove redundancies inside of a formula.

Notation. In a disjunction, $\varphi \equiv \varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$, we say $\psi \in_{\vee} \varphi$, if there is some $i = 1, \cdots, n$ such that $\psi \equiv \varphi_i$. Note that $\psi \in_{\vee} \varphi$ is another representation for the equality $\psi \equiv \mathsf{reorder}_{\vee}\ \varphi\ \psi$.

In right-associative disjunctions we remove the redundancies in Fig. 6 using Lemma 31. We assume the formulas to be right-associative unless otherwise stated.

$$\frac{\Gamma \vdash \varphi \vee \bot}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \vee \top}{\Gamma \vdash \top} \qquad \frac{\Gamma \vdash \varphi \vee \varphi}{\Gamma \vdash \varphi} \quad \frac{\Gamma \vdash \varphi \vee \neg\ \varphi}{\Gamma \vdash \top}$$

Fig. 6. Theorems to remove redundancies inside of a disjunction.

Lemma 31. Let $\varphi : \mathsf{Prop}$ be a right-associative formula. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{simplify}_{\vee}\ \varphi$, where

$$
\begin{aligned}
&\mathsf{simplify}_\vee : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{simplify}_\vee\ (\bot \vee \varphi) \quad = \mathsf{simplify}_\vee\ \varphi\\
&\mathsf{simplify}_\vee\ (\varphi \vee \bot) \quad = \mathsf{simplify}_\vee\ \varphi\\
&\mathsf{simplify}_\vee\ (\top \vee \varphi) \quad = \top\\
&\mathsf{simplify}_\vee\ (\varphi \vee \top) \quad = \top\\
&\mathsf{simplify}_\vee\ (\varphi_1 \vee \varphi_2) = \begin{cases} \top, & \text{if } \varphi_1 \equiv \neg\, \psi \text{ for some } \psi : \mathsf{Prop} \text{ and } \psi \in_\vee \varphi_2;\\ \top, & \text{if } (\neg\, \varphi_1) \in_\vee \varphi_2;\\ \mathsf{simplify}_\vee\ \varphi_2, & \text{if } \varphi_1 \in_\vee \varphi_2;\\ \top, & \text{if } \mathsf{simplify}_\vee\ \varphi_2 \ \equiv \top;\\ \varphi_1, & \text{if } \mathsf{simplify}_\vee\ \varphi_2 \ \equiv \bot;\\ \varphi_1 \vee \mathsf{simplify}_\vee\ \varphi_2, & \text{otherwise.} \end{cases}\\
&\mathsf{simplify}_\vee\ \varphi \qquad\quad = \varphi.
\end{aligned} \tag{18}
$$

Example 32. The formula $\varphi \vee (\psi \vee (\varphi \vee \varphi)) \vee \varphi$ in (19) has redundancies. To remove such redundancies we first use Lemma 14 to get the right-associative version of the formula. Then, we can use the $\mathsf{simplify}_\vee$ function to get the logical equivalent formula $\psi \vee \varphi$.

$$
\cfrac{\cfrac{\cfrac{\Gamma \vdash \varphi \vee (\psi \vee (\varphi \vee \varphi)) \vee \varphi}{\Gamma \vdash \mathsf{assoc}_\vee\ (\varphi \vee (\psi \vee (\varphi \vee \varphi)) \vee \varphi)}\ \text{Lemma 14}}{\Gamma \vdash \varphi \vee (\psi \vee (\varphi \vee (\varphi \vee \varphi)))}\ \text{by (7)}}{\cfrac{\Gamma \vdash \mathsf{simplify}_\vee\ (\varphi \vee (\psi \vee (\varphi \vee (\varphi \vee \varphi))))}{\Gamma \vdash \psi \vee \varphi}\ (18)}\ \text{Lemma 31} \tag{19}
$$

Now, we have removed redundancies in disjunctions by applying the $\mathsf{simplify}_\vee$ function. In a similar way, we define the $\mathsf{simplify}_\wedge$ function to work with conjunctions.

Notation. In a conjunction, $\varphi \equiv \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$, we say $\psi \in_\wedge \varphi$, if there is some $i = 1, \cdots, n$ such that $\psi \equiv \varphi_i$. Note that $\psi \in_\wedge \varphi$ is another representation of the equality $\psi \equiv \mathsf{conjunct}\ \varphi\ \psi$.

In right-associative conjunctions we remove the redundancies in Fig. 7 using Lemma 33.

$$
\frac{\Gamma \vdash \varphi \wedge \top}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \wedge \bot}{\Gamma \vdash \bot} \qquad \frac{\Gamma \vdash \varphi \wedge \varphi}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \wedge \neg\ \varphi}{\Gamma \vdash \bot}
$$

Fig. 7. Theorems to remove redundancies inside of a conjunction.

Lemma 33. Let $\varphi : \mathsf{Prop}$ be a right-associative formula. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{simplify}_\wedge\ \varphi$, where

$$
\begin{aligned}
&\mathsf{simplify}_\wedge : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{simplify}_\wedge\ (\bot \wedge \varphi) \quad = \bot\\
&\mathsf{simplify}_\wedge\ (\varphi \wedge \bot) \quad = \bot\\
&\mathsf{simplify}_\wedge\ (\top \wedge \varphi) \quad = \mathsf{simplify}_\wedge\ \varphi\\
&\mathsf{simplify}_\wedge\ (\varphi \wedge \top) \quad = \mathsf{simplify}_\wedge\ \varphi\\
&\mathsf{simplify}_\wedge\ (\varphi_1 \wedge \varphi_2) = \begin{cases} \bot, & \text{if } \varphi_1 \equiv \neg\,\psi \text{ for some } \psi : \mathsf{Prop} \text{ and } \psi \in_\wedge \varphi_2;\\ \bot, & \text{if } (\neg\,\varphi_1) \in_\wedge \varphi_2;\\ \mathsf{simplify}_\wedge\ \varphi_2, & \text{if } \varphi_1 \in_\wedge \varphi_2;\\ \varphi_1, & \text{if } \mathsf{simplify}_\wedge\ \varphi_2 \ \equiv \top;\\ \bot, & \text{if } \mathsf{simplify}_\wedge\ \varphi_2 \ \equiv \bot;\\ \varphi_1 \wedge \mathsf{simplify}_\wedge\ \varphi_2, & \text{otherwise.} \end{cases}\\
&\mathsf{simplify}_\wedge\ \varphi \qquad\quad = \varphi.
\end{aligned}
\tag{20}
$$

Now, we are ready to define the negative normal form of a formula with simplifications by applying to it the nnf function defined in Lemma 34 (see more details in Appendix D). This definition is mainly based on the `Metis` source code to normalize formulas. To define such a function in type theory we used bounded recursion as we describe in Section 2.1.

$$
\begin{aligned}
&\mathsf{nnf}_1 : \mathsf{Prop} \to \mathsf{Nat} \to \mathsf{Prop}\\
&\mathsf{nnf}_1\ (\varphi_1 \wedge \varphi_2) \quad\ (\mathsf{succ}\ n) = \mathsf{simplify}_\wedge\ (\mathsf{assoc}_\wedge\ (\mathsf{nnf}_1\ \varphi_1\ n \wedge \mathsf{nnf}_1\ \varphi_2\ n))\\
&\mathsf{nnf}_1\ (\varphi_1 \vee \varphi_2) \quad\ (\mathsf{succ}\ n) = \mathsf{simplify}_\vee\ (\mathsf{assoc}_\vee\ (\mathsf{nnf}_1\ \varphi_1\ n \vee \mathsf{nnf}_1\ \varphi_2\ n))\\
&\mathsf{nnf}_1\ (\varphi_1 \supset \varphi_2) \quad\ (\mathsf{succ}\ n) = \mathsf{simplify}_\vee\ (\mathsf{assoc}_\vee\ (\mathsf{nnf}_1\ ((\neg\,\varphi_1) \vee \varphi_2)\ n))\\
&\mathsf{nnf}_1\ (\neg\,(\varphi_1 \wedge \varphi_2))\ (\mathsf{succ}\ n) = \mathsf{simplify}_\vee\ (\mathsf{assoc}_\vee\ (\mathsf{nnf}_1\ ((\neg\,\varphi_1) \vee (\neg\,\varphi_2))\ n))\\
&\mathsf{nnf}_1\ (\neg\,(\varphi_1 \vee \varphi_2))\ (\mathsf{succ}\ n) = \mathsf{simplify}_\wedge\ (\mathsf{assoc}_\wedge\ (\mathsf{nnf}_1\ ((\neg\,\varphi_1) \wedge (\neg\,\varphi_2))\ n))\\
&\mathsf{nnf}_1\ (\neg\,(\varphi_1 \supset \varphi_2))\ (\mathsf{succ}\ n) = \mathsf{simplify}_\wedge\ (\mathsf{assoc}_\wedge\ (\mathsf{nnf}_1\ ((\neg\,\varphi_2) \wedge \varphi_1)\ n))\\
&\mathsf{nnf}_1\ (\neg\,(\neg\,\varphi)) \qquad (\mathsf{succ}\ n) = \mathsf{nnf}_1\ \varphi_1\ n\\
&\mathsf{nnf}_1\ (\neg\,\top) \qquad\quad\ \ (\mathsf{succ}\ n) = \bot\\
&\mathsf{nnf}_1\ (\neg\,\bot) \qquad\quad\ \ (\mathsf{succ}\ n) = \top\\
&\mathsf{nnf}_1\ \varphi \qquad\qquad\quad\ \mathsf{zero} \qquad\ = \varphi
\end{aligned}
\tag{21}
$$

Lemma 34. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{nnf}\ \varphi$, where

$$
\begin{aligned}
&\mathsf{nnf} : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{nnf}\ \varphi = \mathsf{nnf}_1\ \varphi\ (\mathsf{nnf}_{cm}\ \varphi).
\end{aligned}
$$

The nnf_{cm} complexity measure function is defined in Appendix D.

To get the conjunctive normal form, we make sure the formula is a conjunction of disjunctions. For such a purpose, we use distributive laws in Lemma 35 to get that form after applying the nnf function.

Lemma 35. $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{dist}\ \varphi$, where

$$\begin{aligned}
&\mathsf{dist} : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{dist}\ (\varphi_1 \wedge \varphi_2) = \mathsf{dist}\ \varphi_1 \wedge \mathsf{dist}\ \varphi_2\\
&\mathsf{dist}\ (\varphi_1 \vee \varphi_2) = \mathsf{dist}_\vee\ (\mathsf{dist}\ \varphi_1 \vee \mathsf{dist}\ \varphi_2)\\
&\mathsf{dist}\ \varphi \qquad\qquad\ = \varphi
\end{aligned}$$

and

$$\begin{aligned}
&\mathsf{dist}_\vee : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{dist}_\vee\ ((\varphi_1 \wedge \varphi_2) \vee \varphi_3) = \mathsf{dist}_\vee\ (\varphi_1 \vee \varphi_2) \wedge \mathsf{dist}_\vee\ (\varphi_2 \vee \varphi_3)\\
&\mathsf{dist}_\vee\ (\varphi_1 \vee (\varphi_2 \wedge \varphi_3)) = \mathsf{dist}_\vee\ (\varphi_1 \vee \varphi_2) \wedge \mathsf{dist}_\vee\ (\varphi_1 \vee \varphi_3)\\
&\mathsf{dist}_\vee\ \varphi \qquad\qquad\qquad\quad = \varphi.
\end{aligned}$$

We get the conjunctive normal form by applying the nnf function follow by the dist function.

Lemma 36. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{cnf}\ \varphi$, where

$$\begin{aligned}
&\mathsf{cnf} : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{cnf}\ \varphi = \mathsf{dist}\ (\mathsf{nnf}\ \varphi).
\end{aligned}$$

Proof. Composition of Lemma 35 and Lemma 34. ∎

Since all the transformations in Lemma 34 and Lemma 35 came from logical equivalences in propositional logic, we state the following lemmas used in the reconstruction of the `simplify` rule in Lemma 42 and in Theorem 39 for the `canonicalize` rule.

Lemma 37. If $\Gamma \vdash \mathsf{nnf}\ \varphi$ then $\Gamma \vdash \varphi$.

Lemma 38. If $\Gamma \vdash \mathsf{cnf}\ \varphi$ then $\Gamma \vdash \varphi$.

Now, we are ready to reconstruct the canonicalize rule. This inference rule defined in (22) performs normalization for a proposition. That is, depending on the role of the formula in the problem, it converts that formula to its negative normal form or its conjunctive normal form. In both cases, `canonicalize` simplifies the formula by removing redundancies inside of it as we widely described above for theorems in Fig. 5. When the formula plays the axiom or definition role, the `canonicalize` rule transforms the source formula to its negative normal form. Otherwise, this rule converts the formula to its conjunctive normal form.

Since this rule mostly consists of dealing with clauses, to reconstruct this rule, our strategy mainly consists of checking the equality of negative normal form between the source and the target formula. If it fails, we try to reorder the conjunctive normal form of the source formula to match with the conjunctive normal form of the target formula. It definition is as follows.

Theorem 39. Let $\psi : \mathsf{Conclusion}$. If $\Gamma \vdash \varphi$ then $\Gamma\ \vdash\ \mathsf{canonicalize}\ \varphi\ \psi$, where

$$\mathsf{canonicalize} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}$$

$$\mathsf{canonicalize}\ \varphi\ \psi = \begin{cases} \psi, & \text{if } \psi \equiv \varphi;\\ \psi, & \text{if } \psi \equiv \mathsf{nnf}\ \varphi;\\ \psi, & \text{if } \mathsf{cnf}\ \psi \equiv \mathsf{reorder}_{\wedge\vee}\ (\mathsf{cnf}\ \varphi)\ (\mathsf{cnf}\ \psi);\\ \varphi, & \text{otherwise.}\end{cases} \tag{22}$$

Proof.

- Case $\varphi \equiv \psi$. By substitution theorem we conclude $\Gamma \vdash \psi$.
- Case $\psi \equiv \mathsf{nnf}\ \varphi$.

$$\dfrac{\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \ \mathsf{nnf}\ \varphi}\ \text{Lemma 34} \qquad \psi \equiv \mathsf{nnf}\ \varphi}{\Gamma \vdash \psi}\ \mathsf{subst}$$

- Case $\mathsf{cnf}\ \psi \equiv \mathsf{reorder}_{\wedge\vee}\ (\mathsf{cnf}\ \varphi)\ (\mathsf{cnf}\ \psi)$.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \mathsf{cnf}\ \varphi}\ \text{Lemma 36}}{\Gamma \vdash \mathsf{reorder}_{\wedge\vee}\ (\mathsf{cnf}\ \varphi)\ (\mathsf{cnf}\ \psi)}\ \text{Lemma 23} \qquad \mathsf{cnf}\ \psi\ \equiv\ \mathsf{reorder}_{\wedge\vee}\ (\mathsf{cnf}\ \varphi)\ (\mathsf{cnf}\ \psi)}{\dfrac{\Gamma \vdash \mathsf{cnf}\ \psi}{\Gamma \vdash \psi}\ \text{Lemma 38}}\ \mathsf{subst}$$

■

4.2.5 Clausify. The `clausify` rule is an alternative rule to transform a formula into its clausal normal form but without performing simplifications with tautologies or definitions. Recall, this kind of conversion was addressed by the `canonicalize` rule. It is important to notice that this kind of conversions between one formula to its clausal normal form are not unique, and `Metis` has customized approaches to perform that transformations. Therefore, we perform a reordering of the conjunctive normal form given by the cnf function defined in Lemma 36 with the $\mathsf{reorder}_{\wedge\vee}$ function from Lemma 23 to the input formula of the rule.

Example 40. In the following `TSTP` derivation by `Metis`, we see how `clausify` transforms the $\mathtt{n}_0$ formula to get $\mathtt{n}_1$ formula.

```
fof(n₀, ¬ p ∨ (q ∧ r) ...
fof(n₁, (¬ p ∨ q) ∧ (¬ p ∨ r), inf(clausify, n₀)).
```

Theorem 41. Let $\psi : \mathsf{Conclusion}$. If $\Gamma \vdash \varphi$ then $\Gamma \vdash \mathsf{clausify}\ \varphi\ \psi$, where

$$\mathsf{clausify} : \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}$$
$$\mathsf{clausify}\ \varphi\ \psi = \begin{cases} \psi, & \text{if } \varphi \equiv \psi; \\ \mathsf{reorder}_{\wedge\vee}\ (\mathsf{cnf}\ \varphi)\ \psi, & \text{otherwise.} \end{cases}$$

Proof. If $\varphi \equiv \psi$, $\Gamma \vdash \mathsf{clausify}\ \varphi\ \psi$ normalizes to $\Gamma \vdash \psi$. The conclusion follows by applying the subst lemma. Otherwise, we use Lemma 23 and Lemma 36. ■

4.2.6 Simplify. The `simplify` rule is an inference that performs simplification of definitions and tautologies. This rule transverses a list of previous derivations by applying Lemma 31, Lemma 33, among others. This rule works to find a contradiction in the first place, or a new formula (often smaller than its input formulas) to use later in the derivation.

We observe based on the analysis of different cases in the `TSTP` derivations that `simplify` can be modeled by a function with three arguments: two source formulas and the target formula.

Since the main purpose of the `simplify` rule is simplification of formulas, we have defined the reduce_ℓ function to help removing the negation of a given literal ℓ from a input formula.

$$
\begin{aligned}
&\mathsf{reduce}_\ell : \mathsf{Prop} \to \mathsf{Lit} \to \mathsf{Prop}\\
&\mathsf{reduce}_\ell\ (\varphi_1\ \wedge\ \varphi_2)\ \ \ell\ = \mathsf{simplify}_\wedge\ (\mathsf{reduce}_\ell\ \varphi_1\ \ell\ \wedge\ \mathsf{reduce}_\ell\ \varphi_2\ \ell)\\
&\mathsf{reduce}_\ell\ (\varphi_1\ \vee\ \varphi_2)\ \ \ell\ = \mathsf{simplify}_\vee\ (\mathsf{reduce}_\ell\ \varphi_1\ \ell\ \vee\ \mathsf{reduce}_\ell\ \varphi_2\ \ell)\\
&\mathsf{reduce}_\ell\ \varphi \qquad\qquad\ \ \ell\ = \begin{cases}\bot, & \text{if } \varphi \text{ is a literal and } \ell \equiv \mathsf{nnf}(\neg\,\varphi);\\ \varphi, & \text{otherwise.}\end{cases}
\end{aligned}
\tag{23}
$$

Lemma 42. Let ℓ be a literal and $\varphi : \mathsf{Prop}$. If $\Gamma \vdash \varphi$ and $\Gamma \vdash \ell$ then $\Gamma\ \vdash\ \mathsf{reduce}_\ell\ \varphi\ \ell$.

Proof. This proof is by induction on the structure of φ.

- Case $\varphi \equiv \varphi_1 \wedge \varphi_2$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\vdash\varphi_1\wedge\varphi_2}{\Gamma\vdash\varphi_1}\wedge\text{-proj}_1 \qquad \Gamma\vdash\ell}{\Gamma\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell}\text{by ind. hyp.} \qquad \dfrac{\dfrac{\Gamma\vdash\varphi_1\wedge\varphi_2}{\Gamma\vdash\varphi_2}\wedge\text{-proj}_2 \qquad \Gamma\vdash\ell}{\Gamma\vdash\mathsf{reduce}_\ell\ \varphi_2\ \ell}\text{by ind. hyp.}}{\Gamma\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell\ \wedge\ \mathsf{reduce}_\ell\ \varphi_2\ \ell}\wedge\text{-intro}}{\Gamma\vdash\mathsf{simplify}_\wedge\ (\mathsf{reduce}_\ell\ \varphi_1\ \ell\ \wedge\ \mathsf{reduce}_\ell\ \varphi_2\ \ell)}\text{Lemma 33}
$$

- Case $\varphi \equiv \varphi_1 \vee \varphi_2$.

$$
(\mathcal{D})\quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma,\varphi_1\vdash\varphi_1 \qquad \dfrac{\Gamma\vdash\ell}{\Gamma,\varphi_1\vdash\ell}\text{weaken}}{\Gamma,\varphi_1\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell}\text{by ind. hyp.}}{\Gamma,\varphi_1\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell\vee\mathsf{reduce}_\ell\ \varphi_2\ \ell}\vee\text{-intro}_1 \qquad \dfrac{\dfrac{\Gamma,\varphi_2\vdash\varphi_2 \qquad \dfrac{\Gamma\vdash\ell}{\Gamma,\varphi_2\vdash\ell}\text{weaken}}{\Gamma,\varphi_2\vdash\mathsf{reduce}_\ell\ \varphi_2\ \ell}\text{by ind. hyp.}}{\Gamma,\varphi_2\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell\vee\mathsf{reduce}_\ell\ \varphi_2\ \ell}\vee\text{-intro}_2}{\Gamma,\varphi_1\vee\varphi_2\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell\vee\mathsf{reduce}_\ell\ \varphi_2\ \ell}\vee\text{-elim}}{\Gamma\vdash\varphi_1\vee\varphi_2\supset(\mathsf{reduce}_\ell\ \varphi_1\ \ell\vee\mathsf{reduce}_\ell\ \varphi_2\ \ell)}\supset\text{-intro}
$$

$$
\dfrac{\dfrac{\mathcal{D} \qquad \Gamma\vdash\varphi_1\vee\varphi_2}{\Gamma\vdash\mathsf{reduce}_\ell\ \varphi_1\ \ell\vee\mathsf{reduce}_\ell\ \varphi_2\ \ell}\supset\text{-elim}}{\Gamma\vdash\mathsf{simplify}_\vee\ (\mathsf{reduce}_\ell\ \varphi_1\ \ell\vee\mathsf{reduce}_\ell\ \varphi_2\ \ell)}\text{Lemma 31}
$$

- Case φ is a literal and $\ell \equiv \mathsf{nnf}\ (\neg\,\varphi)$.

$$
\dfrac{\dfrac{\dfrac{\Gamma\vdash\ell \qquad \ell\equiv\mathsf{nnf}\ (\neg\,\varphi)}{\Gamma\vdash\mathsf{nnf}\ (\neg\,\varphi)}\text{subst}}{\Gamma\vdash\neg\,\varphi}\text{Lemma 37} \qquad \Gamma\vdash\varphi}{\Gamma\vdash\bot}\neg\text{-elim}
$$

- Otherwise use the same hypothesis $\Gamma \vdash \varphi$. ■

The $\mathsf{simplify}$ function is defined in (24). If some input formula is equal to the target formula, we derive that formula. If some formula is $\bot$, we derive the target formula by using the $\bot - elim$ inference rule. Otherwise, the simplification functions take place if the second input formula is a conjunction or a disjunction. Otherwise, if the second input formula in the sources is a literal, we use the reduce_ℓ function and Lemma 42.

$$\mathsf{simplify} : \mathsf{Premise} \to \mathsf{Premise} \to \mathsf{Conclusion} \to \mathsf{Prop}$$
$$\mathsf{simplify}\ \varphi_1\ \varphi_2\ \psi =$$
$$\begin{cases} \psi, & \text{if } \varphi_i \equiv \bot \text{ for some } i = 1,2; \\ \psi, & \text{if } \varphi_i \equiv \psi \text{ for some } i = 1,2; \\ \mathsf{simplify}_\wedge\ (\mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi)\ \varphi_{22}\ \psi, & \text{if } \varphi_2 \equiv \varphi_{21} \wedge \varphi_{22}; \\ \mathsf{simplify}_\vee\ (\mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi\ \vee\ \mathsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi) & \text{if } \varphi_2 \equiv \varphi_{21} \vee \varphi_{22}; \\ \mathsf{reduce}_\ell\ \varphi_1\ \varphi_2, & \text{if } \varphi_2 \text{ is a literal;} \\ \varphi_1, & \text{otherwise.} \end{cases} \tag{24}$$

Lemma 43. Let $\psi : \mathsf{Conclusion}$. If $\Gamma \vdash \varphi_1$ and $\Gamma \vdash \varphi_2$ then $\Gamma \vdash \mathsf{simplify}\ \varphi_1\ \varphi_2$.

Proof. This proof is by induction on the structure of φ.

- Case $\varphi_i \equiv \psi$ for some $i = 1,2$. If $\varphi_i \equiv \psi$ then by subst lemma since $\Gamma \vdash \varphi_i$ we derive $\Gamma \vdash \psi$.

- Case $\varphi_i \equiv \bot$ for some $i = 1,2$.

$$\dfrac{\dfrac{\Gamma \vdash \varphi_i \qquad \varphi_i \equiv \bot}{\Gamma \vdash \bot}\ \text{subst}}{\Gamma \vdash \psi}\ \bot\text{-elim}$$

- Case $\varphi_2 \equiv \varphi_{21} \wedge \varphi_{22}$.

$$\dfrac{\dfrac{\Gamma \vdash \varphi_1 \qquad \dfrac{\Gamma \vdash \varphi_{21} \wedge \varphi_{22}}{\Gamma \vdash \varphi_{21}}\ \wedge\text{-proj}_1}{\Gamma \vdash \mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi}\ \text{by ind. hyp.} \qquad \dfrac{\Gamma \vdash \varphi_{21} \wedge \varphi_{22}}{\Gamma \vdash \varphi_{22}}\ \wedge\text{-proj}_2}{\Gamma \vdash \mathsf{simplify}\ (\mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi)\ \varphi_{22}\ \psi}\ \text{by ind. hyp.}$$

- Case $\varphi_2 \equiv \varphi_{21} \vee \varphi_{22}$.

$$(\mathcal{D}_1) \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \varphi_1}{\Gamma, \varphi_{21} \vdash \varphi_1}\ \text{weaken} \qquad \Gamma, \varphi_{21} \vdash \varphi_{21}}{\Gamma, \varphi_{21} \vdash \mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi}\ \text{by ind. hyp.}}{\Gamma, \varphi_{21} \vdash \mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi \vee \mathsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi}\ \vee\text{-intro}_1$$

$$(\mathcal{D}_2) \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \varphi_1}{\Gamma, \varphi_{22} \vdash \varphi_1}\ \text{weaken} \qquad \Gamma, \varphi_{22} \vdash \varphi_{22}}{\Gamma, \varphi_{22} \vdash \mathsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi}\ \text{by ind. hyp.}}{\Gamma, \varphi_{22} \vdash \mathsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi \vee \mathsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi}\ \vee\text{-intro}_2$$

$$(\mathcal{D}_3) \qquad \dfrac{\dfrac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma, \varphi_{21} \vee \varphi_{22} \vdash \textsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi \vee \textsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi}\ \vee\text{-elim}}{\Gamma \vdash \varphi_{21} \vee \varphi_{22} \supset (\textsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi \vee \textsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi)}\ \supset\text{-intro}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_3}{\Gamma \vdash \varphi_{21} \vee \varphi_{22} \supset (\textsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi \vee \textsf{simplify}\ \varphi_1\ \varphi_{22}\ \psi)} \qquad \Gamma \vdash \varphi_{21} \vee \varphi_{22}}{\Gamma \vdash \textsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi \vee \textsf{simplify}\ \varphi_2\ \varphi_{22}\ \psi}\ \supset\text{-elim}}{\Gamma \vdash \textsf{simplify}_{\vee}\ (\textsf{simplify}\ \varphi_1\ \varphi_{21}\ \psi \vee \textsf{simplify}\ \varphi_2\ \varphi_{22}\ \psi)}\ \text{Lemma 31}$$

- Case φ_2 is a literal. Use Lemma 42 with $\varphi = \varphi_1$ and $\ell = \varphi_2$. ∎

Theorem 44. Let ψ : Conclusion. Let φ_i : Premise such that $\Gamma \vdash \varphi_i$ for $i = 1, \cdots, n$ and $n \geq 2$. Then $\Gamma \vdash \textsf{simplify}\ \gamma_{n-1}\ \varphi_n\ \psi$ where $\gamma_1 = \varphi_1$, and $\gamma_i \equiv \textsf{simplify}\ \gamma_{i-1}\ \varphi_i\ \psi$.

Proof. We prove this theorem by induction on n.
• Case $n = 2$. Use Lemma 43.
• Case $n > 2$. Suppose this theorem is valid for n, that is, for $i = 1, \cdots, n$, $\Gamma \vdash \textsf{simplify}\ \gamma_{n-1}\ \varphi_n\ \psi$. Let us prove it for $n + 1$.

$$\dfrac{\dfrac{\Gamma \vdash \textsf{simplify}\ \gamma_{n-1}\ \varphi_n\ \psi \qquad \gamma_n \equiv \textsf{simplify}\ \gamma_{n-1}\ \varphi_n\ \psi}{\Gamma \vdash \gamma_n}\ \text{subst} \qquad \Gamma \vdash \varphi_{n+1}}{\Gamma \vdash \textsf{simplify}\ \gamma_n\ \varphi_{n+1}\ \psi}\ \text{by ind. hyp.}$$

∎

Remark. Besides the fact that List Prop $\rightarrow$ Prop is the type that most fit with the `simplify` rule, we choose a different option. In the translation from `TSTP` to `Agda`, we take the list of derivations and we apply the rule by using a left folding (the `foldl` function in functional programming) with the simplify function over the list of $\varphi_1, \varphi_2, \cdots, \varphi_n$ that avoids us to define a new theorem type to support List Prop type in the conclusion side.

Example 45. Let us review the following `TSTP` excerpt where `simplify` was used twice.

```
fof(n₀, (¬ p ∨ q) ∧ ¬ r ∧ ¬ q ∧ (p ∨ (¬ s ∨ r)), ...
fof(n₁, p ∨ (¬ s ∨ r), inf(conjunct, n₀)).
fof(n₂, ¬ p ∨ q, inf(conjunct, n₀)).
fof(n₃, ¬ q, inf(conjunct, n₀)).
fof(n₄, ¬ p, inf(simplify, [n₂, n₃])).
fof(n₅, ¬ r, inf(conjunct, n₀)).
fof(n₆, ⊥, inf(simplify, [n₁, n₄, n₅])).
```

1. The `simplify` rule derives $\neg\, p$ in $\texttt{n}_4$ from $\texttt{n}_2$ and $\texttt{n}_3$ derivations.

$$\textsf{simplify}\ (\neg\, p \vee q)\ (\neg\, q)\ (\neg\, p) = \neg\, p.$$

2. To derive $\bot$ in $\mathtt{n}_6$ we use Theorem 44 and we get the following proof.

$$\dfrac{\dfrac{\Gamma \vdash p \vee (\neg\, s \wedge r) \qquad \Gamma \vdash \neg\, p}{\Gamma \vdash \neg\, s \wedge r}\ \text{Theorem 44} \qquad \Gamma \vdash \neg\, r}{\Gamma \vdash \bot}\ \text{Theorem 44}$$

We have finished the formalization of every inference rule in a `Metis` derivation, we are able to justify step-by-step any proof for a problem in propositional logic. For instance, we tested successfully the translation by `Athena` jointly with the `Agda` formalizations of the rules mentioned above with more than eighty representative theorems in propositional logic. An interested reader can test the problems [34] in the `Athena` tool repository [36].

5 Related Work

Many approaches have been proposed for proof-reconstruction and some tools have been implemented in the last decades. We first mention some tools in type theory and later we listed some proof-reconstruction tools for classical logic.

Kanso in [25, 26] presents a proof-reconstruction in `Agda` for propositional logic. Its tool support proof-reconstruction for `EProver` and `Z3` ATPs following a similar work-flow as we presented in Section 4.1. Nonetheless, its approach employs semantics for logic equivalences. We have avoided the use of propositions meanings towards a future work to support other logics where a syntactical approach plays an important role (for an example of such logics, we refer the reader to [2]). Foster and Struth [18] describe the proof-reconstruction in `Agda` for `Waldmeister` [20], a prover for pure equational logic. As far as we know, no other proof-reconstruction has been carried out neither in `Agda` nor with `Metis` prover.

Another important proof-assistant in type theory is `Coq` [44]. We found the `SMTCoq` [3, 15] tool which provides a certified checker for proof witness coming from the SMT solver `veriT` [9] and adds a new tactic named verit, that calls `veriT` on any `Coq` goal. Also for `Coq`, given a fixed but arbitrary first-order signature, Bezem, Hendriks, and Nivelle in [5] transform a proof produced by the first-order automatic theorem prover `Bliksem` [29] in a `Coq` proof-term.

There are some successful attempts using proof-assistants for classical logic instead of type theory. Let us mention some representative of such tools. This description is mainly based on Sicard-Ramírez and Ospina-Giraldo [38].

The `Isabelle` proof-assistant has the `Sledgehammer` tool. This program provides a full integration between automatic theorem provers [6, 17, 8] and `Isabelle/HOL` [28], the specialization of `Isabelle` for higher-order logic. A modular proof-reconstruction workflow is presented jointly with the full integration of `Leo-II` and `Satallax` provers with `Isabelle/HOL` in Eén and Sörensson [14].

Hurd [22] integrates the first-order resolution prover `Gandalf` with the high-order theorem prover `HOL` [31]. Its `GANDALF_TAC` tactic is able to reconstruct `Gandalf` proofs by using a LCF model. For `HOL Light`, a version of `HOL` but with a simpler logic core, the SMT solver `CVC4` was integrated. Kaliszyk and Urban [24] reconstruct proofs from different ATPs with the `PRocH` tool by replaying detailed inference steps from the ATPs with internal inference methods implemented in `HOL Light`.

6 Conclusions and Future Work

We presented a proof-reconstruction approach in type theory for the propositional fragment of the `Metis` prover. We provided for each `Metis` inference rule a formal description in type theory following a syntactical approach. This formalizations are mainly exposed in Section 4.2.

We built the `Athena` translator tool written in `Haskell` that generates `Agda` proof-terms of `Metis` derivations. `Agda` files generated by this translator imports `Agda` formalizations of the `Metis` reasoning [35, 37].

The reconstruction approach in this study was designed to use only syntactical aspects of the logic. This decision was in the beginning a drawback since it demands more detailed proofs, a description of every transformation or deduction step performed by the prover, which is rarely included in the output of these programs, see Section 4.2. Nevertheless, we chose that syntactical treatment instead of using semantics to extend this work towards the support of first-order logic or other non-classical logics. For first-order logic, recall satisfiability is undecidable and its syntactical aspect plays an important role to reconstruct proofs.

One of the main contribution of this study was increasing the trustworthiness of the automatic prover `Metis`. Justifying a proof by a theorem prover has a real significant impact for these automatic tools. The reverse engineering task to grasp the prover reasoning can reveal important issues or bugs in many parts of these systems (e. g., preprocessing, reasoning, or deduction modules). During this research, we had the opportunity to contribute to `Metis` by reporting some bugs—see Issues No. 2, No. 4, and commit `8a3f11e` in `Metis`' official repository.[5] Fortunately, all these problems were fixed quickly by Hurd in `Metis` 2.3 (release 20170822).

Future work

Further research directions include, but are not limited to:

- extend the proof-reconstruction presented in this paper to
  - support `Metis` inference rules with equality (e. g., equality).
  - support other ATPs for propositional logic like `EProver` or `Z3`. This development can be carried out by following the `EProver` description on Kanso's Ph.D. thesis [25].
  - support `Metis` first-order proofs.
- improve some functions in Section 4.2
  - by investigating the consequences of removing the `clausify` inference rule by the `canonicalize` rule.
  - by increasing the coverage of the `simplify` rule. Since is fairly complex its implementation in the `Metis` source code, some cases could be omitted in Section 4.2.6.

Acknowledgments. Research on this paper was fully supported by Universidad EAFIT in Medellín, Colombia. I would like to thank for this funding and its support for my master studies. This project was a product of the Logic and Computation Research Group at Universidad EAFIT. My special thanks to Andrés Sicard-Ramiréz to give me the opportunity to work on these exciting topics but also for its patience, discussions, and very valuable comments during all this research.

[5] https://github.com/gilith/metis.

I thank Joe Leslie-Hurd for his support and comments about `Metis`. I also acknowledge the work carried out by Alejandro Gómez-Londoño [19] used as the basis for our `TSTP` parsing module in [36].

Last but not least, I thankfully acknowledge Andreas Abel and Chalmers University of Technology at Goteborg, Sweden for inviting me to be part of the `Agda` Implementors' Meeting XXV where I presented part of this research in the meeting.

References

[1] Andreas Abel and Thorsten Altenkirch. A Predicative Analysis of Structural Recursion. Journal of Functional Programming 12.01 (2002), pp. 1–41. DOI: 10.1017/S0956796801004191 (cit. on p. 3).

[2] Juan C. Agudelo-Agudelo. Translating Non-classical Logics into Classical Logic by Using Hidden Variables. Logica Universalis 11.2 (2017), pp. 205–224. DOI: 10.1007/s11787-017-0168-1 (cit. on pp. 4, 27).

[3] Michael Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In: Certified Programs and Proofs (CPP 2011). Ed. by Jean-Pierre Jouannaud and Zhong Shao. Vol. 7080. Lecture Notes in Computer Science. Springer, 2011, pp. 135–150. DOI: 10.1007/978-3-642-25379-9_12 (cit. on p. 27).

[4] Yves Bertot and Pierre Castéran. Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. Springer, 2004. DOI: 10.1007/978-3-662-07964-5 (cit. on p. 3).

[5] Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automated Proof Construction in Type Theory Using Resolution. Journal of Automated Reasoning 29.3-4 (2002), pp. 253–275. DOI: 10.1023/A:1021939521172 (cit. on pp. 2, 3, 27, 36).

[6] Jasmin Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT Solvers. Journal of Automated Reasoning 51.1 (June 2013), pp. 109–128. DOI: 10.1007/s10817-013-9278-5 (cit. on p. 27).

[7] Sascha Böhme and Tjark Weber. Designing Proof Formats: A User's Perspective - Experience Report. In: First International Workshop on Proof eXchange for Theorem Proving - PxTP 2011. 2011. URL: http://hal.inria.fr/hal-00677244 (cit. on p. 1).

[8] Sascha Böhme and Tjark Weber. Fast LCF-Style Proof Reconstruction for Z3. In: Interactive Theorem Proving (ITP 2010). Ed. by Matt Kaufmann and Lawrence C. Paulson. Vol. 6172. Lecture Notes in Computer Science. Springer, 2010, pp. 179–194. DOI: 10.1007/978-3-642-14052-5_14 (cit. on pp. 15, 27).

[9] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: An Open, Trustable and Efficient SMT-Solver. In: Automated Deduction (CADE-22). Ed. by Renate A. Schmidt. Vol. 5663. Lecture Notes in Artifical Intellingence. Springer, 2009, pp. 151–156. DOI: 10.1007/978-3-642-02959-2_12 (cit. on p. 27).

[10] Ana Bove and Venanzio Capretta. Recursive Functions with Higher Order Domains. In: Typed Lambda Calculi and Applications (TLCA 2005). Ed. by Paweł Urzyczyn. Vol. 3461. Lecture Notes in Computer Science. Springer, 2005, pp. 116–130. DOI: 10.1007/11417170_10 (cit. on p. 3).

[11] Leran Cai, Ambrus Kaposi, and Thorsten Altenkirch. Formalising the Completeness Theorem of Classical Propositional Logic in Agda. Unpublished. 2015. URL: https://akaposi.github.io/proplogic.pdf (cit. on p. 5).

[12] T. Coquand. Pattern Matching With Dependent Types. 1992. DOI: 10.1.1.37.9541 (cit. on p. 3).

[13] Dirk van Dalen. Logic and Structure. Universitext. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994. DOI: 10.1007/978-3-662-02962-6 (cit. on p. 5).

[14] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In: Theory and Applications of Satisfiability Testing (SAT 2003). Ed. by Enrico Giunchiglia and Tacchella. Armando. Vol. 2919. Lecture Notes in Computer Science. Springer, 2004, pp. 116–130. DOI: 10.1007/978-3-540-24605-3_37 (cit. on p. 27).

[15] B. Ekici, A. Mebsout, C. Tinelli, C. Keller, and G. Katz. SMTCoq: A plug-in for integrating SMT solvers into Coq. stanford.edu (2017). URL: http://web.stanford.edu/~guyk/pub/CAV2017_C.pdf (cit. on p. 27).

[16] Michael Färber and Cezary Kaliszyk. Metis-based Paramodulation Tactic for HOL Light. In: GCAI 2015. Global Conference on Artificial Intelligence Metis-based. Vol. 36. 2015, pp. 127–136 (cit. on p. 5).

[17] Mathias Fleury and Jasmin Blanchette. Translation of Proofs Provided by External Provers. Tech. rep. Techniche Universität München, 2014. URL: http://perso.eleves.ens-rennes.fr/~mfleur01/documents/Fleury_internship2014.pdf (cit. on pp. 1, 27).

[18] Simon Foster and Georg Struth. Integrating an Automated Theorem Prover in Agda. In: NASA Formal Methods (NFM 2011). Ed. by Mihael Bobaru et al. Vol. 6617. Lecture Notes in Computer Science. Springer, 2011, pp. 116–130. DOI: 10.1007/978-3-642-20398-5_10 (cit. on p. 27).

[19] Alejandro Gómez-Londoño. Proof Reconstruction: Parsing Proofs. Tech. rep. Universidad EAFIT, 2015. URL: http://repository.eafit.edu.co/handle/10784/5484 (cit. on p. 29).

[20] Thomas Hillenbrand, Arnim Buch, Roland Vogt, and Bernd Löchner. WALDMEISTER - High-Performance Equational Deduction. Journal of Automated Reasoning 18.2 (1997), pp. 265–270. DOI: 10.1023/A:1005872405899 (cit. on p. 27).

[21] Joe Hurd. First-order Proof Tactics In Higher-order Logic Theorem Provers. Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports (2003), pp. 56–68. URL: http://www.gilith.com/research/papers (cit. on pp. 2, 5).

[22] Joe Hurd. Integrating Gandalf and HOL. In: Theorem Proving in Higher Order Logics (TPHOLs 2001). Ed. by Yves Bertot, Gilles Dowek, Laurent Théry, and Christine Paulin. Vol. 1690. Lecture Notes in Computer Science. Springer, 2001, pp. 311–321. DOI: 10.1007/3-540-48256-3_21 (cit. on p. 27).

[23] Clément Hurlin, Amine Chaieb, Pascal Fontaine, Stephan Merz, and Tjark Weber. Practical Proof Reconstruction for First-Order Logic and Set-Theoretical Constructions. In: Proceedings of the Isabelle Workshop 2007. Ed. by Lucas Dixon and Moa Johansson. Bremen, Germany, 2007, pp. 2–13 (cit. on p. 2).

[24] Cezary Kaliszyk and Josef Urban. PRocH: Proof Reconstruction for HOL Light. In: Automated Deduction (CADE-24). Ed. by Maria Paola Bonacina. Vol. 7898. Lecture Notes in Artifical Intellingence. Springer, 2013, pp. 267–274. DOI: 10.1007/978-3-642-38574-2_18 (cit. on pp. 2, 27).

[25] Karim Kanso. Agda as a Platform for the Development of Verified Railway Interlocking Systems. PhD thesis. Department of Computer Science. Swansea University, 2012. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.310.1502 (cit. on pp. 27, 28).

[26] Karim Kanso and Anton Setzer. A Light-Weight Integration of Automated and Interactive Theorem Proving. Mathematical Structures in Computer Science 26.1 (2016), pp. 129–153. DOI: 10.1017/S0960129514000140 (cit. on pp. 2, 27).

[27] Chantal Keller. A Matter of Trust: Skeptical Communication Between Coq and External Provers. PhD thesis. École Polytechnique, 2013. URL: https://hal.archives-ouvertes.fr/pastel-00838322/ (cit. on pp. 1, 2).

[28] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. Isabelle/HOL: A Proof Assistant for Higher-order Logic. Vol. 2283. Springer Science & Business Media, 2002 (cit. on p. 27).

[29] Hans de Nivelle. Bliksem 1.10 User Manual. 2003. URL: http://www.ii.uni.wroc.pl/~nivelle/software/bliksem. (cit. on p. 27).

[30] Bengt Nordström, Kent Petersson, and Jan M. Smith. Programming in Martin-Löf's Type Theory. Oxford University Press, 1990 (cit. on p. 2).

[31] Michael Norrish and Konrad Slind. The HOL system description. 2017. URL: https://sourceforge.net/projects/hol/files/hol/kananaskis-11/kananaskis-11-reference.pdf/download (cit. on p. 27).

[32] Lawrence C. Paulson and Jasmin Blanchette. Three Years Of Experience with Sledgehammer, A Practical Link Between Automatic And Interactive Theorem Provers. In: PAAR@ IJCAR. 2010, pp. 1–10 (cit. on p. 2).

[33] Lawrence C. Paulson and Kong Woei Susanto. Source-level Proof Reconstruction For Interactive Theorem Proving. In: TPHOLs. Vol. 4732. Springer. 2007, pp. 232–245 (cit. on pp. 1, 5).

[34] Jonathan Prieto-Cubides. A Collection of Propositional Problems in TPTP Format. June 2017. DOI: 10.5281/ZENODO.817997 (cit. on pp. 2, 27, 37).

[35] Jonathan Prieto-Cubides. A Library for Classical Propositional Logic in Agda. 2017. DOI: 10.5281/zenodo.398852 (cit. on pp. 2, 5, 10, 11, 28).

[36] Jonathan Prieto-Cubides. A Translator Tool for Metis Derivations in Haskell. 2017. DOI: 10.5281/zenodo.437196 (cit. on pp. 2, 27, 29).

[37] Jonathan Prieto-Cubides. Metis Prover Reasoning for Propositional Logic in Agda. 2017. DOI: 10.5281/zenodo.398862 (cit. on pp. 2, 9, 12, 14, 28).

[38] Andrés Sicard-Ramírez and Juan-Fernando Ospina-Giraldo. First-Order Proof Reconstruction (Research Proposal). Universidad EAFIT. 2016 (cit. on p. 27).

[39] Nik Sultana, Christoph Benzmüller, and Lawrence C. Paulson. Proofs and Reconstructions. In: Frontiers of Combining Systems (FroCoS 2015). Ed. by Carsten Lutz and Silvio Ranise. Vol. 9322. Lecture Notes in Computer Science. Springer, 2015, pp. 256–271. DOI: 10.1007/978-3-319-24246-0_16 (cit. on p. 8).

[40] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure. Journal of Automated Reasoning 43.4 (July 2009), p. 337. DOI: 10.1007/s10817-009-9143-8 (cit. on p. 5).

[41] Geoff Sutcliffe and Stuart Melville. The Practice of Clausification in Automatic Theorem Proving. South African Computer Journal 18 (1996), pp. 57–68 (cit. on p. 19).

[42] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Allen Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In: International Joint Conference on Automated Reasoning (IJCAR 2006). Ed. by Ulrich Furbach and Natarajan Shankar.

Vol. 4130. Lecture Notes in Artifical Intellingence. Springer, 2006, pp. 67–81. DOI: 10.1007/11814771_7 (cit. on p. 6).

[43] The Agda Developement Team. Agda 2.4.2.3. 2015. URL: http://wiki.portal.chalmers.se/agda/pmwiki.php (cit. on p. 2).

[44] The Coq Developement Team. The Coq Proof Assistant. Reference Manual. 2015 (cit. on p. 27).

[45] Philip Wadler. Propositions as Types. Communications of the ACM 58.12 (2015), pp. 75–84 (cit. on p. 2).

Appendix A Customized TSTP syntax

We adopted a special TSTP syntax to improve the readability of the TSTP examples shown in this document. Some of the modifications to the original presentation of TSTP syntax in Section 3.2 are the following.

- The formulas names are sub indexed (e. g., instead of `axiom_0`, we write $\texttt{axiom}_0$).
- We use `inf` instead of `inference` field.
- We shorten names generated automatically by `Metis`, (e. g., $\texttt{s}_0$ instead of `subgoal_0` or $\texttt{n}_0$ instead of `normalize_0`).
- We remove the `plain` role.
- We remove empty fields in the inference information.
- The brackets in the argument of a unary inference are removed (e. g., instead of `inf(rule, [], [`$\texttt{n}_0$`]))`, we write `inf(rule, [], `$\texttt{n}_0$`)))`.
- If the inference rule does not need arguments except its parent nodes, we remove the field of useful information (e. g., `inf(canonicalize, premise)` instead of `inf(canonicalize, [], premise)`).
- We use the symbols ($\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\supset$) for formulas instead of (`$false`, `$true`, `~`, `&`, `|`, `=>`) TPTP symbols.
- When the purpose to show a TSTP derivation does not include some parts of the derivation we use the ellipsis (`...`) to avoid such unnecessary parts.

For example, let us consider the TSTP derivation generated by `Metis` in Fig. 8 and its customized version in Fig. 9

```
fof(premise, axiom, p).
fof(goal, conjecture, p).
fof(subgoal_0, plain, p, inference(strip, [], [goal])).
fof(negate_0_0, plain, ~ p, inference(negate, [], [subgoal_0])).
fof(normalize_0_0, plain, ~ p,
  inference(canonicalize, [], [negate_0_0])).
fof(normalize_0_1, plain, p,
  inference(canonicalize, [], [premise])).
fof(normalize_0_2, plain, $false,
  inference(simplify, [], [normalize_0_0, normalize_0_1]))
cnf(refute_0_0, plain, $false,
  inference(canonicalize, [], [normalize_0_2])).
```

Fig. 8. `Metis`' TSTP derivation for the problem $p \vdash p$.

```
fof(premise, axiom, p).
fof(goal, conjecture, p).
fof(s₀, p, inf(strip, goal)).
fof(neg₀, ¬ p, inf(negate, s₀)).
fof(n₀, ¬ p, inf(canonicalize, neg₀)).
fof(n₁, p, inf(canonicalize, premise)).
fof(n₂, ⊥, inf(simplify, [n₀, n₁]))
cnf(r₀, ⊥, inf(canonicalize, n₂)).
```

Fig. 9. Metis' TSTP derivation using a customized syntax

Appendix B Bounded Recursion of the `strip` Function

In Section 4.2.1 we describe the `strip` function to get the subgoals of a certain a goal. We define the first version of this function with the strip_0 function in (25) but the reader can note that this function is not a structurally recursive function. Therefore, we define a structurally recursive function of this function in (5).

$$
\begin{aligned}
&\mathsf{strip}_0 : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{strip}_0\ (\varphi_1 \wedge \varphi_2) &&= \mathsf{uh}\ (\mathsf{strip}_0\ \varphi_1) \wedge \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_0\ \varphi_2)\\
&\mathsf{strip}_0\ (\varphi_1 \vee \varphi_2) &&= \mathsf{uh}\ ((\neg\, \varphi_1) \supset \mathsf{strip}_0\ \varphi_2)\\
&\mathsf{strip}_0\ (\varphi_1 \supset \varphi_2) &&= \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_0\ \varphi_2)\\
&\mathsf{strip}_0\ (\neg\, (\varphi_1 \wedge \varphi_2)) &&= \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_0\ (\neg\, \varphi_2))\\
&\mathsf{strip}_0\ (\neg\, (\varphi_1 \vee \varphi_2)) &&= \mathsf{uh}\ (\mathsf{strip}_0\ (\neg\, \varphi_1)) \wedge \mathsf{uh}\ ((\neg\, \varphi_1) \supset \mathsf{strip}_0\ (\neg\, \varphi_2))\\
&\mathsf{strip}_0\ (\neg\, (\varphi_1 \supset \varphi_2)) &&= \mathsf{uh}\ (\mathsf{strip}_0\ \varphi_1) \wedge \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_0\ (\neg\, \varphi_2))\\
&\mathsf{strip}_0\ (\neg\, (\neg\, \varphi_1)) &&= \mathsf{uh}\ (\mathsf{strip}_0\ \varphi_1)\\
&\mathsf{strip}_0\ (\neg\, \bot) &&= \top\\
&\mathsf{strip}_0\ (\neg\, \top) &&= \bot\\
&\mathsf{strip}_0\ \varphi &&= \varphi.
\end{aligned}
\tag{25}
$$

The complexity measure of strip_0 is given by the strip_{cm} function defined in (B).

$$
\begin{aligned}
&\mathsf{strip}_{cm} : \mathsf{Prop} \to \mathsf{Nat} \to \mathsf{Prop}\\
&\mathsf{strip}_{cm}\ (\varphi_1 \wedge \varphi_2) &&= \max\ (\mathsf{strip}_{cm}\ \varphi_1)\ (\mathsf{strip}_{cm}\ \varphi_2) + 1\\
&\mathsf{strip}_{cm}\ (\varphi_1 \vee \varphi_2) &&= \mathsf{strip}_{cm}\ \varphi_2 + 1\\
&\mathsf{strip}_{cm}\ (\varphi_1 \supset \varphi_2) &&= \mathsf{strip}_{cm}\ \varphi_2 + 1\\
&\mathsf{strip}_{cm}\ (\neg\, \top) &&= \mathsf{strip}_{cm}\ (\neg\, \varphi_2) + 1\\
&\mathsf{strip}_{cm}\ (\neg\, \bot) &&= \max\ (\mathsf{strip}_{cm}\ (\neg\, \varphi_1))\ (\mathsf{strip}_{cm}\ (\neg\, \varphi_2)) + 1\\
&\mathsf{strip}_{cm}\ (\neg\, (\varphi_1 \wedge \varphi_2)) &&= \max\ (\mathsf{strip}_{cm}\ \varphi_1)\ (\mathsf{strip}_{cm}\ (\neg\, \varphi_2)) + 1\\
&\mathsf{strip}_{cm}\ (\neg\, (\varphi_1 \vee \varphi_2)) &&= \max\ (\mathsf{strip}_{cm}\ (\neg\, \varphi_1))\ (\mathsf{strip}_{cm}\ (\neg\, \varphi_2)) + 1\\
&\mathsf{strip}_{cm}\ (\neg\, (\varphi_1 \supset \varphi_2)) &&= \mathsf{strip}_{cm}\ \varphi_1 + 1\\
&\mathsf{strip}_{cm}\ (\neg\, (\neg\, \varphi)) &&= 1\\
&\mathsf{strip}_{cm}\ \varphi &&= 1
\end{aligned}
$$

Appendix C Another Case in the Proof of the `strip` Inference Rule

- Case $\varphi \equiv \varphi_1 \supset \varphi_2$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi_1 \vdash \varphi_1}\ \text{assume} \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \mathsf{strip}_1\ (\varphi_1 \supset \varphi_2)\ (\mathsf{succ}\ n)}{\Gamma \vdash \mathsf{uh}\ (\varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n)}\ \text{by (5)}}{\Gamma \vdash \varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n}\ \text{Lemma 7}}{\Gamma, \varphi_1 \vdash \varphi_1 \supset \mathsf{strip}_1\ \varphi_2\ n}\ \text{weaken}}{\Gamma, \varphi_1 \vdash \mathsf{strip}_1\ \varphi_2\ n}\ \supset\text{-elim}}{\Gamma, \varphi_1 \vdash \varphi_2}\ \text{by ind. hyp.}}{\Gamma \vdash \varphi_1 \supset \varphi_2}\ \supset\text{-intro.}
$$

Appendix D Bounded Recursion of the nnf Function

In Section 4.2.4 we discuss a custom negative normal form of a formula. To convert a formula to such a normal form, we define the function nnf_0 in (26).

$$
\begin{aligned}
&\mathsf{nnf}_0 : \mathsf{Prop} \to \mathsf{Prop}\\
&\mathsf{nnf}_0\ (\varphi_1 \wedge \varphi_2) && = \mathsf{simplify}_\wedge\ (\mathsf{assoc}_\wedge\ (\mathsf{nnf}_0\ \varphi_1 \wedge \mathsf{nnf}_0\ \varphi_2))\\
&\mathsf{nnf}_0\ (\varphi_1 \vee \varphi_2) && = \mathsf{simplify}_\vee\ (\mathsf{assoc}_\vee\ (\mathsf{nnf}_0\ \varphi_1 \vee \mathsf{nnf}_0\ \varphi_2))\\
&\mathsf{nnf}_0\ (\varphi_1 \supset \varphi_2) && = \mathsf{simplify}_\vee\ (\mathsf{assoc}_\vee\ (\mathsf{nnf}_0\ ((\neg\, \varphi_1) \vee \varphi_2)))\\
&\mathsf{nnf}_0\ (\neg\, (\varphi_1 \wedge \varphi_2)) && = \mathsf{simplify}_\vee\ (\mathsf{assoc}_\vee\ (\mathsf{nnf}_0\ ((\neg\, \varphi_1) \vee (\neg\, \varphi_2))))\\
&\mathsf{nnf}_0\ (\neg\, (\varphi_1 \vee \varphi_2)) && = \mathsf{simplify}_\wedge\ (\mathsf{assoc}_\wedge\ (\mathsf{nnf}_0\ ((\neg\, \varphi_1) \wedge (\neg\, \varphi_2))))\\
&\mathsf{nnf}_0\ (\neg\, (\varphi_1 \supset \varphi_2)) && = \mathsf{simplify}_\wedge\ (\mathsf{assoc}_\wedge\ (\mathsf{nnf}_0\ ((\neg\, \varphi_2) \wedge \varphi_1)))\\
&\mathsf{nnf}_0\ (\neg\, (\neg\, \varphi)) && = \mathsf{nnf}_0\ \varphi_1\\
&\mathsf{nnf}_0\ (\neg\, \top) && = \bot\\
&\mathsf{nnf}_0\ (\neg\, \bot) && = \top\\
&\mathsf{nnf}_0\ \varphi && = \varphi
\end{aligned}
\tag{26}
$$

However, then nnf_0 function is not a structurally recursive function. Therefore, we define a bounded recursion in (21) using as the second argument for the bounded recursion its complexity measure. The nnf_{cm} function in (27) computes that complexity measure.

$$
\begin{aligned}
&\mathsf{nnf}_{cm} : \mathsf{Prop} \to \mathsf{Nat} \to \mathsf{Prop}\\
&\mathsf{nnf}_{cm}\ (\varphi_1 \wedge \varphi_2) && = \mathsf{nnf}_{cm}\ \varphi_1 + \mathsf{nnf}_{cm}\ \varphi_2 + 1\\
&\mathsf{nnf}_{cm}\ (\varphi_1 \vee \varphi_2) && = \mathsf{nnf}_{cm}\ \varphi_1 + \mathsf{nnf}_{cm}\ \varphi_2 + 1\\
&\mathsf{nnf}_{cm}\ (\varphi_1 \supset \varphi_2) && = 2 \cdot \mathsf{nnf}_{cm}\ \varphi_1 + \mathsf{nnf}_{cm}\ \varphi_2 + 1\\
&\mathsf{nnf}_{cm}\ (\neg\, (\varphi_1 \wedge \varphi_2)) && = \mathsf{nnf}_{cm}\ (\neg\, \varphi_1) + \mathsf{nnf}_{cm}\ (\neg\, \varphi_2) + 1\\
&\mathsf{nnf}_{cm}\ (\neg\, (\varphi_1 \vee \varphi_2)) && = \mathsf{nnf}_{cm}\ (\neg\, \varphi_1) + \mathsf{nnf}_{cm}\ (\neg\, \varphi_2) + 1\\
&\mathsf{nnf}_{cm}\ (\neg\, (\varphi_1 \supset \varphi_2)) && = \mathsf{nnf}_{cm}\ (\neg\, \varphi_1) + 1\\
&\mathsf{nnf}_{cm}\ (\neg\, (\neg\, \varphi)) && = \mathsf{nnf}_{cm}\ \varphi_1 + \mathsf{nnf}_{cm}\ (\neg\, \varphi_2) + 3\\
&\mathsf{nnf}_{cm}\ (\neg\, \top) && = 1\\
&\mathsf{nnf}_{cm}\ (\neg\, \bot) && = 1\\
&\mathsf{nnf}_{cm}\ \varphi && = 1
\end{aligned}
\tag{27}
$$

Another approach to define the negative normal form in type theory without using a complexity measure for the bounded recursion would modify the definition of nnf defined in [5]. The authors avoid the termination problem by using the polarity of the formula as an additional argument of its negative normal form function. However, be aware the polarity function is not standard and `Metis` has its own definition.

 Compiled on 29/11/2017 at 3:55pm

Appendix E A Complete Example

E.1 Installing Athena

Athena is the proof-reconstruction tool that accompanying this paper. This tool is written in Haskell and it was tested with GHC 8.2.1. To install Athena, the package manager cabal is required as well. Athena was tested with cabal 1.24.0.

Let us download the Athena repository running the following command:

```
$ git clone https://github.com/jonaprieto/athena.git
$ cd athena
```

To install Athena run the following command:

```
$ make install
```

To install the Agda libraries, agda-prop, agda-metis, and the Agda standard library, run the following command:

```
$ make install-libraries
```

E.2 Installing the Metis Prover

To install the Metis prover v2.3 (release 20171021), we refer the reader to its official repository at https://github.com/gilith/metis.

As an alternative to install the prover from the Metis sources, we have provided a Haskell client to use this prover but also other provers with Online-ATPs tool[6]. To install this tool run the following command:

```
$ make online-atps
$ online-atps --version
Online-atps version 0.1.1
```

E.3 TPTP problem

Let us consider the following theorem[7]:

$$(p \Rightarrow q) \wedge (q \Rightarrow p) \vdash (p \vee q) \Rightarrow (p \wedge q) \tag{28}$$

This problem can be encode in TPTP syntax (file problem.tptp) as follows:

```
$ cat problem.tptp
fof(premise, axiom, (p => q) & (q => p)).
fof(goal, conjecture, (p | q) => (p & q)).
```

[6] https://github.com/jonaprieto/online-atps.

[7] Problem No. 13 in Disjunction Section in [34].

E.4 Metis derivation

To obtain the Metis derivation of the TPTP problem showed above, make sure your Metis version is supported by running the following command. Recall we support the version 2.3 (release 20171021).

```
$ metis --version
metis 2.3 (release 20171021)
```

To generate the TSTP derivation of problem.tptp run the following command:

```
$ metis --show proof problem.tptp > problem.tstp
$ cat problem.tstp
...
fof(premise, axiom, ((p => q) & (q => p))).
fof(goal, conjecture, ((p | q) => (p & q))).
fof(subgoal_0, plain, ((p | q) => p), inference(strip, [], [goal])).
fof(subgoal_1, plain, (((p | q) & p) => q), inference(strip, [], [goal])).
fof(negate_0_0, plain, (~ ((p | q) => p)),
    inference(negate, [], [subgoal_0])).
...
```

If we are using the Online-ATPs tool run the following command:

```
$ online-atps --atp=metis problem.tptp  > problem.tstp
```

Using our customized TSTP syntax, the above Metis derivation looks like:

```
fof(premise, axiom, (p ⊃ q) ∧ (q ⊃ p)).
fof(goal, conjecture, (p ∨ q) ⊃ (p ∧ q)).
fof(s0, (p ∨ q) ⊃ p, inf(strip, goal)).
fof(s1, ((p ∨ q) ∧ p) ⊃ q, inf(strip, goal)).
fof(neg0, ¬ ((p ∨ q) ⊃ p), inf(negate, s0)).
fof(n00, (¬ p ∨ q) ∧ (¬ q ∨ p), inf(canonicalize, premise)).
fof(n01, ¬ q ∨ p, inf(conjunct, n00)).
fof(n02, ¬ p ∧ (p ∨ q), inf(canonicalize, neg0)).
fof(n03, p ∨ q, inf(conjunct, n02)).
fof(n04, ¬ p, inf(conjunct, n02)).
fof(n05, q, inf(simplify, [n03, n04])).
cnf(r00, ¬ q ∨ p, inf(canonicalize, n01)).
cnf(r01, q, inf(canonicalize, n05)).
cnf(r02, p, inf(resolve, q, [r01, r00])).
cnf(r03, ¬ p, inf(canonicalize, n04)).
cnf(r04, ⊥, inf(resolve, p, [r02, r03])).
fof(neg1, ¬ ((p ∨ q) ∧ p) ⊃ q), inf(negate, s1)).
fof(n10, ¬ q ∧ p ∧ (p ∨ q), inf(canonicalize, neg1)).
fof(n11, (¬ p ∨ q) ∧ (¬ q ∨ p), inf(canonicalize, premise)).
fof(n12, ¬ p ∨ q, inf(conjunct, n11)).
fof(n13, ⊥, inf(simplify,[n10, n12])).
cnf(r10, ⊥, inf(canonicalize, n13)).
```

E.5 Generating the Agda proof-term

Table 2. Metis inference rules implemented in agda-metis.

Metis rule	Theorem number	Implementation
strip	10	strip-thm
conjunct	13	conjunct-thm
resolve	27	resolve-thm
canonicalize	39	canonicalize-thm
clausify	41	clausify-thm
simplify	44	simplify-thm

To obtain the Agda proof-term of the Metis derivation run the following command:

```
$ athena problem.tstp
```

The correspondent Agda file will be created in the same directory that contains problem.tstp using the same name but the extension of Agda, that is, .agda.

```
$ cat problem.agda
------------------------------------------------------------------------------
-- Athena version 0.1-f54e580.
-- TSTP file: problem.tstp.
------------------------------------------------------------------------------

module problem where

------------------------------------------------------------------------------

open import ATP.Metis 2 public
open import Data.PropFormula 2 public

------------------------------------------------------------------------------

-- Variables.

p : PropFormula
p = Var (# 0)

q : PropFormula
```

```
q = Var (# 1)

-- Axiom.

a₁ : PropFormula
a₁ = ((p ⊃ q) ∧ (q ⊃ p))

-- Premise.

Γ : Ctxt
Γ = [ a₁ ]

-- Conjecture.

goal : PropFormula
goal = ((p ∨ q) ⊃ (p ∧ q))

-- Subgoals.

subgoal₀ : PropFormula
subgoal₀ = ((p ∨ q) ⊃ p)

subgoal₁ : PropFormula
subgoal₁ = (((p ∨ q) ∧ p) ⊃ q)

------------------------------------------------------------------------------
-- Proof of subgoal₀.
------------------------------------------------------------------------------

proof₀ : Γ ⊢ subgoal₀
proof₀ =
  (RAA
    (resolve-thm ⊥ p
      (resolve-thm p q
        (simplify-thm q
          (conjunct-thm (p ∨ q)
            (canonicalize-thm ((¬ p) ∧ (p ∨ q))
              (assume {Γ = Γ} (¬ subgoal₀))))
          (conjunct-thm (¬ p)
            (canonicalize-thm ((¬ p) ∧ (p ∨ q))
              (assume {Γ = Γ} (¬ subgoal₀)))))
        (conjunct-thm ((¬ q) ∨ p)
          (canonicalize-thm (((¬ p) ∨ q) ∧ ((¬ q) ∨ p))
            (weaken (¬ subgoal₀)
              (assume {Γ = ∅} a₁)))))
      (conjunct-thm (¬ p)
```

```
        (canonicalize-thm ((¬ p) ∧ (p ∨ q))
          (assume {Γ = Γ} (¬ subgoal₀))))))

------------------------------------------------------------------------------
-- Proof of subgoal₁.
------------------------------------------------------------------------------

proof₁ : Γ ⊢ subgoal₁
proof₁ =
  (RAA
    (simplify-thm ⊥
      (canonicalize-thm ((¬ q) ∧ (p ∧ (p ∨ q)))
        (assume {Γ = Γ} (¬ subgoal₁)))
      (conjunct-thm ((¬ p) ∨ q)
        (canonicalize-thm (((¬ p) ∨ q) ∧ ((¬ q) ∨ p))
          (weaken (¬ subgoal₁)
            (assume {Γ = ∅} a₁))))))

------------------------------------------------------------------------------
-- Proof of the goal.
------------------------------------------------------------------------------

proof : Γ ⊢ goal
proof =
  ⊃-elim
    strip-thm
    (∧-intro proof₀ proof₁)
```

Now, we are ready to verify the `Metis` derivation by type-checking with `Agda` the reconstructed proof showed above. Make sure the `Agda` version is 2.5.3.

```
$ agda --version
Agda version 2.5.3
$ agda problem.agda
```

As we can see in the `Agda` code showed above, the term `proof`, the proof-term of the `Metis` derivation is referring to the proof-terms $\texttt{proof}_0$ and $\texttt{proof}_1$. Recall, `Metis` stripes the goal into subgoals to prove it. Therefore, these terms are the proof-terms for the refutations of the subgoals s_0 and s_1. We show in the following sections the respective natural deduction trees for these refutations.

E.6 Refutation tree for the subgoal s_0

For this subgoal, its respective `TSTP` derivation is the following:

```
fof(premise, axiom, (p ⊃ q) ∧ (q ⊃ p)).
fof(goal, conjecture, (p ∨ q) ⊃ (p ∧ q)).
```

```
fof(s_0, (p ∨ q) ⊃ p, inf(strip, goal)).
...
fof(neg_0, ¬ ((p ∨ q) ⊃ p), inf(negate, s_0)).
fof(n_00, (¬ p ∨ q) ∧ (¬ q ∨ p), inf(canonicalize, premise)).
fof(n_01, ¬ q ∨ p, inf(conjunct, n_00)).
fof(n_02, ¬ p ∧ (p ∨ q), inf(canonicalize, neg_0)).
fof(n_03, p ∨ q, inf(conjunct, n_02)).
fof(n_04, ¬ p, inf(conjunct, n_02)).
fof(n_05, q, inf(simplify,[n_03, n_04])).
cnf(r_00, ¬ q ∨ p, inf(canonicalize, n_01)).
cnf(r_01, q, inf(canonicalize, n_05)).
cnf(r_02, p, inf(resolve, q, [r_01, r_00])).
cnf(r_03, ¬ p, inf(canonicalize, n_04)).
cnf(r_04, ⊥, inf(resolve, p, [r_02, r_03])).
...
```

The refutation tree is the following:

$$(\mathcal{R}_1)\quad \dfrac{\dfrac{\mathcal{D}_1 \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, \neg s_0 \vdash \neg s_0}\,\text{assume } \neg s_0}{\Gamma, \neg s_0 \vdash \neg p \wedge (p \vee q)}\,\text{Theorem 39}}{\Gamma, \neg s_0 \vdash \neg p}\,\text{Theorem 13}}{\Gamma, \neg s_0 \vdash \bot}\,\text{Theorem 27 with } \ell = p}{\Gamma \vdash s_0}\,\text{RAA.}$$

$$(\mathcal{D}_1)\quad \dfrac{\dfrac{\mathcal{D}_2}{\Gamma, \neg s_0 \vdash \neg q \vee p} \qquad \dfrac{\dfrac{\mathcal{D}_3}{\Gamma, \neg s_0 \vdash p \vee q} \quad \dfrac{\mathcal{D}_4}{\Gamma, \neg s_0 \vdash \neg p}}{\Gamma, \neg s_0 \vdash q}\,\text{Theorem 44}}{\Gamma, \neg s_0 \vdash p}\,\text{Theorem 27 with } \ell = q$$

$$(\mathcal{D}_2)\quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash (p \supset q) \wedge (q \supset p)}\,\text{axiom premise}}{\Gamma, \neg s_0 \vdash (p \supset q) \wedge (q \supset p)}\,\text{weaken}}{\Gamma, \neg s_0 \vdash (\neg p \vee q) \wedge (\neg q \vee p)}\,\text{Theorem 39}}{\Gamma, \neg s_0 \vdash \neg q \vee p}\,\text{Theorem 13}$$

$$(\mathcal{D}_3)\quad \dfrac{\dfrac{\dfrac{}{\Gamma, \neg s_0 \vdash \neg s_0}\,\text{assume}}{\Gamma, \neg s_0 \vdash \neg p \wedge (p \vee q)}\,\text{Theorem 39}}{\Gamma, \neg s_0 \vdash p \vee q}\,\text{Theorem 13}$$

$$(\mathcal{D}_4)\quad \dfrac{\dfrac{\dfrac{}{\Gamma, \neg s_0 \vdash \neg s_0}\,\text{assume } \neg s_0}{\Gamma, \neg s_0 \vdash \neg p \wedge (p \vee q)}\,\text{Theorem 39}}{\Gamma, \neg s_0 \vdash \neg p}\,\text{Theorem 13}$$

E.7 Refutation tree for the subgoal s_1

For this subgoal, its respective TSTP derivation is the following:

```
fof(premise, axiom, (p ⊃ q) ∧ (q ⊃ p)).
...
fof(s₁, ((p ∨ q) ∧ p) ⊃ q, inf(strip, goal)).
...
fof(neg₁, ¬ (((p ∨ q) ∧ p) ⊃ q), inf(negate, s₁)).
fof(n₁₀, ¬ q ∧ p ∧ (p ∨ q), inf(canonicalize, neg₁)).
fof(n₁₁, (¬ p ∨ q) ∧ (¬ q ∨ p), inf(canonicalize, premise)).
fof(n₁₂, ¬ p ∨ q, inf(conjunct, n₁₁)).
fof(n₁₃, ⊥, inf(simplify,[n₁₀, n₁₂])).
cnf(r₁₀, ⊥, inf(canonicalize, n₁₃)).
```

The refutation tree is the following:

$$(\mathcal{R}_2)\quad \cfrac{\cfrac{\cfrac{\cfrac{}{\Gamma, \neg s_1 \vdash \neg s_1}\ \text{assume } (\neg s_1)}{\Gamma, \neg s_1 \vdash \neg q \wedge p \wedge (p \vee q)}\ \text{Theorem 39} \qquad \cfrac{\cfrac{\cfrac{\cfrac{}{\Gamma \vdash (p \supset q) \wedge (q \supset p)}\ \text{axiom premise}}{\Gamma, \neg s_1 \vdash (p \supset q) \wedge (q \supset p)}\ \text{weaken}}{\Gamma, \neg s_1 \vdash (\neg p \vee q) \wedge (\neg q \vee p)}\ \text{Theorem 39}}{\Gamma, \neg s_1 \vdash \neg p \vee q}\ \text{Theorem 13}}{\Gamma, \neg s_1 \vdash \bot}\ \text{Theorem 44}}{\Gamma \vdash s_1}\ \text{RAA.}$$

E.8 The proof of the goal

$$\cfrac{\cfrac{}{\Gamma \vdash (s_0 \wedge s_1) \supset \text{goal}}\ \text{Theorem 10} \qquad \cfrac{\cfrac{\mathcal{R}_1}{\Gamma \vdash s_0} \qquad \cfrac{\mathcal{R}_2}{\Gamma \vdash s_1}}{\Gamma \vdash s_0 \wedge s_1}\ \wedge\text{-intro}}{\Gamma \vdash \text{goal}}\ \supset\text{-elim}$$