Redirecting Proofs by Contradiction

Jasmin Christian Blanchette

Technische Universität München, Germany

Abstract

This paper presents an algorithm that redirects proofs by contradiction. The input is a refutation graph, as produced by an automatic theorem prover (e.g., E, SPASS, Vampire, Z3); the output is a direct proof expressed in natural deduction extended with case analyses and nested subproofs. The algorithm is implemented in Isabelle's Sledgehammer, where it enhances the legibility of machine-generated proofs.

1 Introduction

The proofs returned by automatic theorem provers (ATPs) are notoriously difficult to read. This is an issue if an ATP solves an open mathematical problem (such as the Robbins conjecture [11]), because users then certainly want to study the proof closely. But even in the context of program verification, where users are normally satisfied with a "proved" or "disproved," they might still want to analyze proofs—for example, if they suspect errors in their axiom set.

Our interest in intelligible ATP proofs has a different origin. The tool Sledgehammer [19] integrates ATPs with the proof assistant Isabelle/HOL [15]. Given an Isabelle conjecture, Sledgehammer heuristically selects relevant lemmas from Isabelle's libraries, translates them along with the conjecture to first-order logic, and sends the resulting problem to state-of-the-art provers such as E [24], SPASS [28], Vampire [22], and Z3 [5].

To guard against bugs in the ATPs and in Sledgehammer's translation module, ATP proofs are reconstructed in Isabelle. This is accomplished through either a single invocation of the built-in resolution prover *metis* [8] or a structured Isar proof [20]. The latter option is useful for larger proofs, which *metis* fails to re-find within a reasonable time. But most users find the proofs unattractive and are disinclined to insert them in their theory text. As an illustration, consider the conjecture $\mathsf{length}\ (\mathsf{tl}\ xs) \le \mathsf{length}\ xs$, which states that the tail of a list (the list from which we remove its first element, or the empty list if the list is empty) is at most as long as the full list. The proof found by Vampire, translated to Isar, is as follows:

```
proof neg_clausify
  assume length (tl xs) ≰ length xs
  hence drop (length xs) (tl xs) ≠ [] by (metis drop_eq_Nil)
  hence tl (drop (length xs) xs) ≠ [] by (metis drop_tl)
  hence ∀u. xs @ u ≠ xs ∨ tl u ≠ [] by (metis append_eq_conv_conj)
  hence tl [] ≠ [] by (metis append_Nil2)
  thus False by (metis tl.simps(1))
qed
```

(The function drop n removes the first n elements from a list.) The *neg_clausify* proof method puts the Isabelle conjecture into negated clause form to ensure that it has the same shape as the corresponding ATP conjecture. The negation of the clause is introduced in the `assume` line, and a sequence of intermediate facts each introduced by `hence` leads to a contradiction.

There is a considerable body of research about making resolution proofs intelligible. Early work focused on translating detailed resolution proofs into natural deduction calculi [13, 21]. Although they are arguably more readable, these calculi operate at the logical level, whereas humans reason mostly at the "assertion level," invoking definitions and lemmas without providing the full logical details. A line of research focused on transforming natural deduction proofs into assertion-level proofs [1, 7], culminating with the systems TRAMP [12] and Otterfier [31]. More related work includes the identification of obvious inferences [4,23], the successful transformation of EQP's proof of the Robbins conjecture using ILF [3], and more recently the use of TPTP-based tools to present Mizar articles [27].

It would have been interesting to try out TRAMP and Otterfier, but these are large pieces of unmaintained software that are hardly installable on modern machines and that only support older ATP systems. Regardless, the problem looks somewhat different in the context of Sledgehammer. Because the provers are given hundreds of lemmas as axioms, they tend to find short proofs with few lemmas. Moreover, Sledgehammer can coalesce consecutive inferences if short proofs are desired. Replaying an inference is a minor issue, thanks to *metis*.

The first obstacle to readability is that the Isar proof, like the underlying ATP proof, is by contradiction. This paper presents an algorithm for transforming proofs by contradiction into direct proofs—or *redirecting* proofs—to improve intelligibility. Knuth, Larrabee, and Roberts call the unnecessary use of proof by contradiction a sin against mathematical exposition [10, § 3]—but since redirection is always possible, what would a necessary use look like?

The redirection algorithm is not be tied to any one calculus or logic, as long as it admits contraposition. In particular, it works on the Isar proofs generated by Sledgehammer or directly on first-order TSTP proofs [26]. The direct proofs are expressed in a simple Isar-like syntax, which can be regarded as natural deduction extended with case analyses and nested subproofs (Section 2). The algorithm is first demonstrated on a few examples (Section 3) before it is presented in more detail, both in prose and as Standard ML pseudocode (Section 4).

For examples with a simple linear structure, such as the Isar proof above, the proof can be turned around by applying contraposition repeatedly:

```
proof –
  have tl [] = [] by (metis tl.simps(1))
  hence ∃u. xs @ u = xs ∧ tl u = [] by (metis append_Nil2)
  hence tl (drop (length xs) xs) = [] by (metis append_eq_conv_conj)
  hence drop (length xs) (tl xs) = [] by (metis drop_tl)
  thus length (tl xs) ≤ length xs by (metis drop_eq_Nil)
qed
```

The direct proof is easier to understand than the indirect one, even though it does not quite look like a human-written proof—humans would most likely avoid the detour through drop.

The approach works on arbitrary proofs by contradiction. A prototype demonstrated at earlier workshops [18, 19] sometimes exhibited exponential behavior. This has been resolved: Excluding a linear number of additional inferences that justify case analyses, each inference in the proof by contradiction now gives rise to exactly one inference in the direct proof. The algorithm can easily process proofs with hundreds or thousands of inferences.

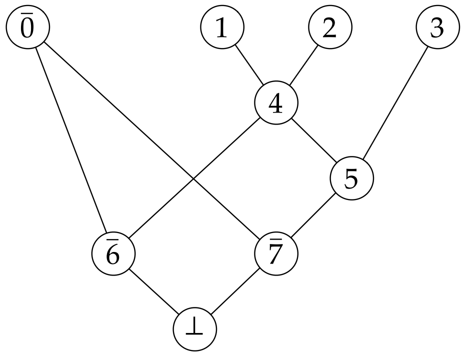
The algorithm is implemented in Sledgehammer. It is also described in Section 6.8 of my Ph.D. thesis [2], whose text largely forms the basis of this paper.

2 Proof Notations

Proof Graph. ATP proofs identify formulas by numbers. There may be several conjectures, in which case they are interpreted disjunctively. The negated conjectures and user-provided axioms are numbered 0, 1, 2, ..., $n-1$, and the derivations performed during proof search (whether or not they participate in the final proof) are numbered sequentially from n. We abstract the ATP proofs by ignoring the formulas and keeping only the numbers. We call formulas *atoms* since we are not interested in their structure. The letters a, b denote atoms.

An atom is *tainted* if it is one of the negated conjectures or has been derived, directly or indirectly, from a negated conjecture. For convenience, we relabel the ATP proof's atoms so that tainted atoms are decorated with a bar, denoting negation. Thus, if atom 3, corresponding to the formula ϕ, is tainted, it is relabeled to $\overline{3}$, but it still stands for ϕ and is called an atom despite the negative bar. After the relabeling, removing the bar negates the formula: $3 \equiv \neg\,\phi$.

A proof graph is a directed acyclic graph in which an edge $a \rightarrow a'$ indicates that atom a is used to derive atom a'. Proof graphs are required to have exactly one sink node, whose formula is $\bot$, and only one connected component. We adopt the convention that derived nodes appear lower than their parent nodes in the graph and omit the arrowheads:



It is natural to write $\bot$ rather than a numeric label for the sink node in examples.

In Sledgehammer, unary inferences are collapsed, and the first-order formulas are translated back to HOL before the proof is redirected. This is outside the scope of this paper and is explained in more detail in a companion paper [25].

Isar Proofs. Proof graphs cannot represent proofs by case analysis and only serve for the redirection algorithm's input. We need more powerful notations for the output. Isar proofs [14, §4; 29] are a linear representation of natural deduction proofs in the style of Jaśkowski [9]. Unlike Gentzen-style trees [6], they allow the sharing of common derivations. The proof on the left-hand side is by contradiction; that on the right-hand side is the corresponding direct proof:

```
proof neg_clausify                proof –
  assume $\overline{0}$                        have 4 by (metis 1 2)
  have 4 by (metis 1 2)             have 5 by (metis 3 4)
  have 5 by (metis 3 4)             have $6 \lor 7$ by metis
  have $\overline{6}$ by (metis $\overline{0}$ 4)             moreover
  have $\overline{7}$ by (metis $\overline{0}$ 5)             { assume 6
  show $\bot$ by (metis $\overline{6}$ $\overline{7}$)              have 0 by (metis 4 6) }
qed                                 moreover
                                    { assume 7
                                      have 0 by (metis 5 7) }
                                    ultimately show 0 by metis
                                  qed
```

Notice that the direct proof involves a 2-way case analysis on a disjunction. Generalized disjunctions of the form $a_1 \lor \cdots \lor a_m$ are called *clauses* and are denoted by the letters c, d, e. Clauses are considered equal modulo associativity, commutativity, and idempotence. Sets of clauses are denoted by Γ.

Proof redirection requires that inferences can be redirected using the contrapositive but otherwise makes no assumptions about the proof calculus. Inferences that introduce new symbols can also be redirected; for example, skolemization becomes "un-herbrandization" [25, §4].

Shorthand Proofs. The last proof format is an ad hoc shorthand notation for a subset of Isar. In their simplest form, these shorthand proofs are a list of derivations $c_1, \ldots, c_m \rhd c$ whose intuitive meaning is: "From the hypotheses c_1 and ... and c_m, infer c." The clauses on the left-hand side are interpreted as a set Γ.

If a hypothesis c_i is the previous derivation's conclusion, we can omit it and write ▶ instead of $\rhd$. This notation mimics Isar, with $\rhd$ for `have` or `show` and ▶ for `hence` or `thus`. Depending on whether we use the abbreviated format, our running example becomes

$$\begin{array}{rcl} 1,2 & \rhd & 4 \\ 3,4 & \rhd & 5 \\ \overline{0},4 & \rhd & \overline{6} \\ \overline{0},5 & \rhd & \overline{7} \\ \overline{6},\overline{7} & \rhd & \bot \end{array} \qquad\qquad \begin{array}{rcl} 1,2 & \rhd & 4 \\ 3 & \blacktriangleright & 5 \\ \overline{0},4 & \rhd & \overline{6} \\ \overline{0},5 & \rhd & \overline{7} \\ \overline{6} & \blacktriangleright & \bot \end{array}$$

Each derivation $\Gamma \rhd c$ is essentially a sequent with Γ as the antecedent and c as the succedent. For proofs by contradiction, the clauses in the antecedent are either the negated conjecture ($\overline{0}$), atoms that correspond to background facts (1, 2, and 3), or atoms that were proved in preceding sequents (4, 5, $\overline{6}$, and $\overline{7}$); the succedent of the last sequent is always $\bot$.

Direct proofs can be presented in the same way, but the negated conjecture $\overline{0}$ may not appear in any of the sequents' antecedents, and the last sequent must have the conjecture 0 as its succedent. In some of the direct proofs, it is useful to introduce case analyses. For example:

$$\begin{array}{r}
1,2 \rhd 4 \\
3 \blacktriangleright 5 \\
\rhd 6 \vee 7
\end{array}$$

$$\left[\begin{array}{r|r}
[6] & [7] \\
4 \blacktriangleright 0 & 5 \blacktriangleright 0
\end{array}\right]$$

In general, case analysis blocks have the form

$$\left[\begin{array}{r|c|r}
[c_1] & \cdots & [c_m] \\
\Gamma_{11} \rhd d_{11} & \cdots & \Gamma_{m1} \rhd d_{m1} \\
\vdots & & \vdots \\
\Gamma_{1k_1} \rhd d_{1k_1} & \cdots & \Gamma_{mk_m} \rhd d_{mk_m}
\end{array}\right]$$

with the requirement that a sequent with the succedent $c_1 \vee \cdots \vee c_m$ has been proved immediately above the case analysis. Each of the branches must also be a valid proof. The assumptions $[c_i]$ may be used to discharge hypotheses in the same branch, as if they had been sequents $\rhd c_i$. The case analysis will sometimes be regarded as a sequent

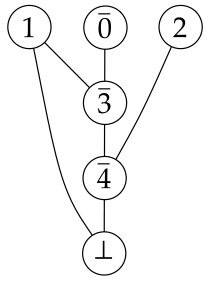
$$c_1 \vee \cdots \vee c_m,\ \bigcup_{i,j} \left(\Gamma_{ij} - c_i - \bigcup_{j'<j} d_{ij'}\right) \ \rhd\ d_{1k_1} \vee \cdots \vee d_{mk_m}$$

by ignoring its internal structure.

3 Examples of Proof Redirection

Before reviewing the redirection algorithm, we consider four examples of proofs by contradiction and redirect them to produce a direct proof. The first example has a simple linear structure, the second and third examples involve a "lasso," and the last example has no apparent structure.

A Linear Proof. We start with a simple proof by contradiction expressed as a proof graph and in our shorthand notation:



$$\begin{array}{r}
\overline{0},1 \rhd \overline{3} \\
2,\overline{3} \rhd \overline{4} \\
1,\overline{4} \rhd \bot
\end{array}$$

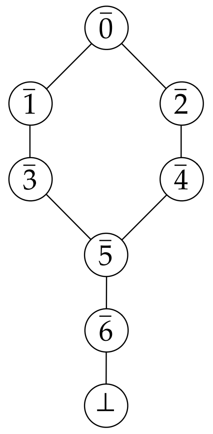
We redirect the sequents using sequent-level contraposition to eliminate all taints (represented as bars after the relabeling). This gives

$$\begin{aligned} 1,3 &\rhd 0 \\ 2,4 &\rhd 3 \\ 1 &\rhd 4 \end{aligned}$$

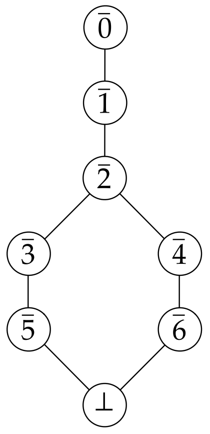
We then obtain the direct proof by reversing the order of the sequents, and introduce ▶ wherever possible:

```
proof –
  have 4 by (metis 1)          1 ▷ 4
  hence 3 by (metis 2)         2 ▶ 3
  thus 0 by (metis 1)          1 ▶ 0
qed
```

Lasso-Shaped Proofs. The next two examples look superficially like lassos but are of course acyclic, as required of all proof graphs:



$$\begin{aligned} \bar{0} &\rhd \bar{1} \\ \bar{0} &\rhd \bar{2} \\ \bar{1} &\rhd \bar{3} \\ \bar{2} &\rhd \bar{4} \\ \bar{3},\bar{4} &\rhd \bar{5} \\ \bar{5} &\rhd \bar{6} \\ \bar{6} &\rhd \bot \end{aligned}$$



$$\begin{aligned} \bar{0} &\rhd \bar{1} \\ \bar{1} &\rhd \bar{2} \\ \bar{2} &\rhd \bar{3} \\ \bar{2} &\rhd \bar{4} \\ \bar{3} &\rhd \bar{5} \\ \bar{4} &\rhd \bar{6} \\ \bar{5},\bar{6} &\rhd \bot \end{aligned}$$

We start with the example on the left-hand side. Starting from $\bot$, it is easy to redirect the stem:

$$\begin{aligned} &\rhd 6 \\ 6 &\rhd 5 \\ 5 &\rhd 3 \vee 4 \end{aligned}$$

When applying the contrapositive to eliminate the negations in $\bar{3},\bar{4} \rhd \bar{5}$, we obtain a disjunction in the succedent: $5 \rhd 3 \vee 4$. To continue from there, we introduce a case analysis. In each branch, we can finish the proof:

$$\left[\begin{array}{r|r} [3] & [4] \\ 3 \rhd 1 & 4 \rhd 2 \\ 1 \rhd 0 & 2 \rhd 0 \end{array}\right]$$

In the second lasso example, the cycle occurs near the end of the contradiction proof. A disjunction already arises when we redirect the last derivation. Naively finishing each branch independently leads to a fair amount of duplication:

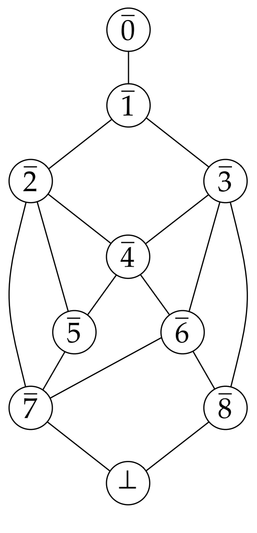
$$\begin{array}{c} \rhd\ 5 \lor 6 \\ \left[\begin{array}{r|r} {[5]} & {[6]} \\ 5 \rhd 3 & 6 \rhd 4 \\ 3 \rhd 2 & 4 \rhd 2 \\ 2 \rhd 1 & 2 \rhd 1 \\ 1 \rhd 0 & 1 \rhd 0 \end{array}\right] \end{array}$$

The key observation is that the two branches can share the last two inferences. This yields the following proof (without and with ▶):

$$\begin{array}{c} \rhd\ 5 \lor 6 \\ \left[\begin{array}{r|r} {[5]} & {[6]} \\ 5 \rhd 3 & 6 \rhd 4 \\ 3 \rhd 2 & 4 \rhd 2 \end{array}\right] \\ 2 \rhd 1 \\ 1 \rhd 0 \end{array} \qquad\qquad \begin{array}{c} \rhd\ 5 \lor 6 \\ \left[\begin{array}{r|r} {[5]} & {[6]} \\ \blacktriangleright 3 & \blacktriangleright 4 \\ \blacktriangleright 2 & \blacktriangleright 2 \end{array}\right] \\ \blacktriangleright 1 \\ \blacktriangleright 0 \end{array}$$

Here we were fortunate that the branches were joinable on the atom 2. To avoid duplication, we must in general join on a disjunction $a_1 \lor \cdots \lor a_m$, as in the next example.

A Spaghetti Proof. The final example is diabolical (and slightly unrealistic, perhaps):



$$\begin{array}{r@{\ }l} \overline{0} & \rhd\ \overline{1} \\ \overline{1} & \rhd\ \overline{2} \\ \overline{1} & \rhd\ \overline{3} \\ \overline{2},\overline{3} & \rhd\ \overline{4} \\ \overline{2},\overline{4} & \rhd\ \overline{5} \\ \overline{3},\overline{4} & \rhd\ \overline{6} \\ \overline{2},\overline{5},\overline{6} & \rhd\ \overline{7} \\ \overline{3},\overline{6} & \rhd\ \overline{8} \\ \overline{7},\overline{8} & \rhd\ \bot \end{array}$$

We start with the contrapositive of the last sequent:

$$\rhd\ 7 \lor 8$$

We perform a case analysis on $7 \vee 8$. Since we want to avoid duplication in the two branches, we first determine which nodes are reachable in the refutation graph by navigating upward from either $\overline{7}$ or $\overline{8}$ but not from both. The only such nodes here are $\overline{5}$, $\overline{7}$, and $\overline{8}$. In each branch, we can perform derivations of the form $\Gamma \rhd b$ where $\Gamma \cap \{5,7,8\} \neq \emptyset$ without fearing duplication. Following this rule, we can only perform one inference in the right branch before we must stop:

$$\begin{array}{c}[8]\\ 8 \;\rhd\; 3 \vee 6\end{array}$$

Any further inferences would need to be repeated in the left branch, so it is indeed a good idea to stop. The left branch starts as follows:

$$\begin{array}{c}[7]\\ 7 \;\rhd\; 2 \vee 5 \vee 6\end{array}$$

We would now like to perform the inference $5 \rhd 2 \vee 4$. This would certainly not lead to any duplication, because $\overline{5}$ is not reachable from $\overline{8}$ by navigating upward in the refutation graph. However, we cannot discharge the hypothesis 5, having established only the disjunction $2 \vee 5 \vee 6$. We need a case analysis on the disjunction to proceed:

$$\left[\begin{array}{c|c|c} & [5] & \\ [2] & 5 \;\rhd\; 2 \vee 4 & [6]\end{array}\right]$$

The 2 and 6 subbranches are left alone, because there is no node that is reachable only from $\overline{2}$ or $\overline{6}$ but not from the other two nodes in $\{\overline{2},\overline{5},\overline{6}\}$ by navigating upward in the refutation graph. Since only one branch is nontrivial, it is arguably more aesthetically pleasing to abbreviate the entire case analysis to

$$2 \vee 5 \vee 6 \;\rhd\; 2 \vee 4 \vee 6$$

Putting this all together, the outer case analysis becomes

$$\left[\begin{array}{l|l} \quad[7] & \\ \blacktriangleright\; 2 \vee 5 \vee 6 & \quad[8] \\ \blacktriangleright\; 2 \vee 4 \vee 6 & \blacktriangleright\; 3 \vee 6\end{array}\right]$$

The left branch proves $2 \vee 4 \vee 6$, the right branch proves $3 \vee 6$; hence, both branches together prove $2 \vee 3 \vee 4 \vee 6$. Next, we perform the inference $6 \rhd 3 \vee 4$. This requires a case analysis on $2 \vee 3 \vee 4 \vee 6$:

$$\left[\begin{array}{c|c|c|c} & & & [6] \\ [2] & [3] & [4] & 6 \;\rhd\; 3 \vee 4\end{array}\right]$$

This proves $2 \vee 3 \vee 4$. Since only one branch is nontrivial, we prefer to abbreviate the case analysis to

$$2 \vee 3 \vee 4 \vee 6 \;\rhd\; 2 \vee 3 \vee 4$$

It may help to think of such abbreviated inferences as instances of rewriting modulo associativity, commutativity, and idempotence. Here, 6 is rewritten to $3 \lor 4$ in $2 \lor 3 \lor 4 \lor 6$, resulting in $2 \lor 3 \lor 4$. Similarly, the sequent $4 \rhd 2 \lor 3$ gives rise to the case analysis

$$\left[\begin{array}{c|c|c} & & {[4]} \\ {[2]} & {[3]} & 4 \rhd 2 \lor 3 \end{array}\right]$$

which can be abbreviated as well. We are left with $2 \lor 3$. The rest is analogous to the second lasso-shaped proof:

$$\begin{array}{c} \left[\begin{array}{r|r} {[2]} & {[3]} \\ 2 \rhd 1 & 3 \rhd 1 \end{array}\right] \\ 1 \rhd 0 \end{array}$$

Putting all of this together, we obtain the following proof, expressed in Isar and in shorthand. The result is quite respectable, considering the spaghetti-like graph we started with:

```
proof –
  have 7∨8 by metis
  moreover
  { assume 7
    hence 2∨5∨6 by metis
    hence 2∨4∨6 by metis }
  moreover
  { assume 8
    hence 3∨6 by metis }
  ultimately have 2∨3∨4∨6 by metis
  hence 2∨3∨4 by metis
  hence 2∨3 by metis
  moreover
  { assume 2
    hence 1 by metis }
  moreover
  { assume 3
    hence 1 by metis }
  ultimately have 1 by metis
  thus 0 by metis
qed
```

$$\begin{array}{c} \rhd\ 7 \lor 8 \\ \left[\begin{array}{l|l} \quad {[7]} & \\ \blacktriangleright\ 2 \lor 5 \lor 6 & \quad {[8]} \\ \blacktriangleright\ 2 \lor 4 \lor 6 & \blacktriangleright\ 3 \lor 6 \end{array}\right] \\ \blacktriangleright\ 2 \lor 3 \lor 4 \\ \blacktriangleright\ 2 \lor 3 \\ \left[\begin{array}{r|r} {[2]} & {[3]} \\ \blacktriangleright\ 1 & \blacktriangleright\ 1 \end{array}\right] \\ \blacktriangleright\ 0 \end{array}$$

4 The Redirection Algorithm

The process we applied in the examples above can be generalized into an algorithm. The algorithm takes an arbitrary proof by contradiction expressed as a set of sequents as input, and produces a proof in our Isar-like shorthand notation, with sequents and case analysis blocks. The proof is constructed one inference at a time starting from $\top$ (the negation of $\bot$) until the conjecture—in general, the disjunction of the conjectures—is proved.

Basic Concepts. A fundamental operation is sequent-level contraposition. Let $a_1, \ldots, a_m$ be the untainted atoms and $\overline{b_1}, \ldots, \overline{b_n}$ the tainted atoms of a proof by contradiction. The proof then consists of the following three kinds of sequent (with $n > 0$):

$$a_1, \ldots, a_m, \overline{b_1}, \ldots, \overline{b_n} \rhd \bot \qquad a_1, \ldots, a_m, \overline{b_1}, \ldots, \overline{b_n} \rhd \overline{b} \qquad a_1, \ldots, a_m \rhd a$$

Their *contrapositives* are, respectively,

$$a_1, \ldots, a_m \rhd b_1 \vee \cdots \vee b_n \qquad a_1, \ldots, a_m, b \rhd b_1 \vee \cdots \vee b_n \qquad a_1, \ldots, a_m \rhd a$$

We call the contrapositives of the sequents in the proof by contradiction the *redirected sequents*.

Based on the set of redirected sequents, we define the *atomic inference graph* (AIG) with, for each redirected sequent $\Gamma \rhd c$, an edge from each atom in Γ to each atom in c, and no additional edges. The AIG encodes the order in which the atoms can be inferred in a direct proof. Navigating forward (downward) in this graph along the unnegated tainted atoms b_j corresponds to navigating backward (upward) in the refutation graph along the $\overline{b_j}$'s.

Like the underlying refutation graph, the AIG is acyclic and connected. Potential cycles would involve either only untainted atoms a_i, only tainted atoms b_j's, or a mixture of both kinds. A cycle $a_{i_1} \to \cdots \to a_{i_k} \to a_{i_1}$ is impossible, because the contrapositive leaves these inferences unchanged and hence the cycle would need to occur in the (acyclic) refutation graph. A cycle $b_{j_1} \to \cdots \to b_{j_k} \to b_{j_1}$ is impossible, because the contrapositive turns all the edges around and hence the reverse cycle would need to occur in the refutation graph. Finally, mixed cycles necessarily involve an edge $b \to a$, which is impossible because redirected sequents with untained atoms a can only have untainted atoms as predecessors (cf. $a_1, \ldots, a_m \rhd a$).

Given a set of (tainted or untainted) atoms A, the *zone* of an atom $a \in A$ with respect to A is the set of possibly trivial descendants of a in the AIG that are not descendants of any of the other atoms in A. As a trivial descendant of itself, a will either belong to its own zone or to no zone all at (depending on whether it is a descendant of a node $a' \in A - \{a\}$). Zones identify inferences that can safely be performed inside a branch in a case analysis.

The Algorithm. The algorithm keeps track of the *last-proved clause* (initially $\top$), the set of *already proved atoms* (initially the set of facts taken as axioms), and the set of *remaining sequents* to use (initially all the redirected sequents provided as input). It performs these steps:

1. If there are no remaining sequents, stop.
2. If the last-proved clause is $\top$ or a single atom:
    - 2.1. Pick a sequent $\Gamma \rhd c$ among the remaining sequents that can be proved using only already proved atoms, preferring sequents with a single atom in their succedent.
    - 2.2. Append $\Gamma \rhd c$ to the proof.
    - 2.3. Make c the last-proved clause, add c to the already proved atoms if it is an atom, and remove $\Gamma \rhd c$ from the remaining sequents.
    - 2.4. Go to step 1.

3. Otherwise, the last-proved succedent is of the form $a_1 \vee \cdots \vee a_m$. An m-way case analysis is called for:[1]
   3.1. Compute the zone of each atom a_i with respect to $\{a_1, \ldots, a_m\}$.
   3.2. For each a_i, compute the set $\mathscr{S}_i$ of sequents $\Gamma \rhd c$ such that Γ consists only of already proved atoms or atoms within a_i's zone.
   3.3. Recursively invoke the algorithm m times, once for each a_i, each time with a_i as the last-proved clause, a_i added to the already proved atoms, and $\mathscr{S}_i$ as the set of remaining sequents. This step yields m (possibly empty) subproofs $\pi_1, \ldots, \pi_m$.
   3.4. Append the following case analysis block to the proof:
   $$\left[\begin{array}{c|c|c} [a_1] & \cdots & [a_m] \\ \pi_1 & \cdots & \pi_m \end{array}\right]$$
   3.5. Make the succedent $b_1 \vee \cdots \vee b_n$ of the case analysis block (regarded as a sequent) the last-proved clause, add b_1 to the already proved atoms if $k = 1$, and remove all sequents belonging to any of the sets $\mathscr{S}_i$ from the remaining sequents.
   3.6. Go to step 1.

Whenever a redirected sequent is generated, it is removed from the set of remaining sequents. In step 3, the recursive calls operate on pairwise disjoint subsets $\mathscr{S}_i$ of the remaining sequents. Consequently, each redirected sequent appears at most once in the generated proof, and the resulting direct proof contains the same number of inferences as the initial proof by contradiction. In Isar, each case analysis is additionally justified by one *metis* inference.

In the degenerate case where no atoms are tainted (i.e., the proof exploits an inconsistency in the axiom set), the generated proof is simply a linearization of the refutation graph, and the last inference proves $\bot$ (which is, unusually, untainted). To produce a syntactically valid Isar proof, a trivial inference must be added to derive the conjecture from $\bot$.

Pseudocode. To make the above description more concrete, the algorithm is presented in Standard ML pseudocode below.[2] The pseudocode is fairly faithful to the description above. Atoms are represented by integers and literals by sets (lists) of integers. Go-to statements are implemented by recursion, and the state is threaded through recursive calls as three arguments (*last*, *earlier*, and *seqs*). One notable difference, justified by a desire to avoid code duplication, is that the set of already proved atoms, called *earlier*, excludes the last-proved clause *last*. Hence, we take *last* $\cup$ *earlier* to obtain the already proved atoms, where *last* is either the empty list (representing $\top$) or a singleton list (representing a single atom).

Shorthand proofs are represented as lists of *inference*s:

`datatype` *inference* $=$ Have `of` *int list* $\times$ *int list* | Cases `of` (*int* $\times$ *inference list*) *list*

[1] A generalization would be to perform a m'-way case analysis, with $m' < m$, by keeping some disjunctions. For example, we could perform a 3-way case analysis with $a_1 \vee a_2$, a_3, and a_4 as the assumptions instead of breaking all the disjunctions in a 4-way analysis. This could lead to nicer proofs if the disjuncts are carefully chosen.

[2] The actual ML code is distributed with Isabelle. A recent repository version is available at `http://isabelle.in.tum.de/repos/isabelle/file/e5303bd748f2/src/HOL/Tools/ATP/atp_proof_redirect.ML`.

The main function implementing the algorithm follows:

```
fun redirect $last\ earlier\ seqs$ =
  if null $seqs$ then
    []
  else if length $last \leq 1$ then
    let val $provable$ = filter (fn $(\Gamma, \_) \Rightarrow \Gamma \subseteq last \cup earlier$) $seqs$
        val $horn\_provable$ = filter (fn $(\_, [\_]) \Rightarrow$ true | _ $\Rightarrow$ false) $provable$
        val $(\Gamma, c)$ = hd ($horn\_provable$ @ $provable$)
    in Have $(\Gamma, c)$ :: redirect $c$ $(last \cup earlier)$ $(seqs - \{(\Gamma, c)\})$ end
  else
    let val $zs$ = zones_of (length $last$) (map (descendants $seqs$) $last$)
        val $\mathscr{S}$ = map (fn $z \Rightarrow$ filter (fn $(\Gamma, \_) \Rightarrow \Gamma \subseteq earlier \cup z$) $seqs$) $zs$
        val $cases$ = map (fn $(a, ss) \Rightarrow (a,$ redirect $[a]$ $earlier\ ss$)) (zip $last\ \mathscr{S}$)
    in Cases $cases$ :: redirect (succedent_of_cases $cases$) $earlier$ $(seqs - \bigcup \mathscr{S})$ end
```

The code uses familiar ML functions, such as map, filter, and zip. It also relies on a descendants function that returns the descendants of the specified node in the AIG associated with *seqs*; its definition is omitted. Finally, the code depends on the following straightforward functions:

```
fun zones_of 0 _ = []
  | zones_of $n$ $(B :: Bs) = (B - \bigcup Bs)$ :: zones_of $(n-1)$ $(Bs$ @ $[B])$

fun succedent_of_inf (Have $(\_, c)) = c$
  | succedent_of_inf (Cases $cases$) = succedent_of_cases $cases$
and succedent_of_case $(a, []) = [a]$
  | succedent_of_case $(\_, infs)$ = succedent_of_inf (last $infs$)
and succedent_of_cases $cases$ = $\bigcup$(map succedent_of_case $cases$)
```

Correctness. It is not hard to convince ourselves that the proof output by redirect is correct by inspecting the code. A Have (Γ, c) sequent is appended only if all the atoms in Γ have been proved (or assumed) already, and a case analysis on $a_1 \vee \cdots \vee a_m$ always follows a sequent with the succedent $a_1 \vee \cdots \vee a_m$. Whenever a sequent is output, it is removed from *seqs*. The function returns only if *seqs* is empty, at which point the conjecture must have been proved (except in the degenerate case where the negated conjecture does not participate in the refutation).

Termination is not quite as obvious. The recursion is well-founded, because the pair (length *seqs*, length *last*) becomes strictly smaller with respect to the lexicographic extension of $<$ on natural numbers for each of the three syntactic recursive calls.

- For the first recursive call, the list $seqs - \{(\Gamma, c)\}$ is strictly shorter than *seqs* since $(\Gamma, c) \in seqs$.
- The second call is performed for each branch of a case analysis; the *ss* argument is a (not necessarily strict) subset of the caller's *seqs*, and the list $[a]$ is strictly shorter than *last*, which has length 2 or more.

- For the third call, the key property is that at least one of the zones is nonempty, from which we obtain $seqs - \bigcup \mathscr{S} \subset seqs$. If all the zones were empty, each atom a_i would be the descendant of at least one atom $a_{i'}$ in the AIG (with $i' \neq i$), which is impossible because the AIG is acyclic.

As for run-time exceptions, the only worrisome construct is the hd call in redirect's second branch. We must convince ourselves that there exists at least one sequent $(\Gamma, c) \in seqs$ such that $\Gamma \subseteq last \cup earlier$. Intuitively, this is unsurprising, because *seqs* is initialized from a well-formed refutation graph: The nonexistence of such a sequent would indicate a gap or a cycle in the refutation graph. More precisely, if there exist untainted atoms $\notin last \cup earlier$, these can always be processed first; indeed, the preference for sequents with a single atom in their succedent ensures that they are processed before the first case analysis. Otherwise:

- If *last* is [] (representing $\top$) or an untainted atom, the contrapositive $a_1, \ldots, a_m \rhd b_1 \vee \cdots \vee b_n$ of the very last inference is applicable since it only depends on untainted atoms, all of which have already been proved.
- Otherwise, *last* is a tainted atom b. The refutation graph must contain an inference $a_1, \ldots, a_m, \overline{b_1}, \ldots, \overline{b_n} \rhd \overline{b}$, whose redirected inference is $a_1, \ldots, a_m, b \rhd b_1 \vee \cdots \vee b_n$. Since it only depends on b and untainted atoms, it is applicable.

Inlining. As a postprocessing step, we can abbreviate case analyses in which only one branch is nontrivial, transforming

$$\left[\begin{array}{c|c|c|r|c|c|c} & & & [c_i] & & & \\ & & & d_{11}, \ldots, d_{1k_1} \rhd e_1 & & & \\ & & & \vdots & & & \\ [c_1] & \cdots & [c_{i-1}] & d_{n1}, \ldots, d_{nk_n} \rhd e_n & [c_{i+1}] & \cdots & [c_m] \end{array}\right] \quad \text{into} \quad \begin{array}{r} \widetilde{d_{11}}, \ldots, \widetilde{d_{1k_1}} \rhd \widetilde{e_1} \\ \vdots \\ \widetilde{d_{n1}}, \ldots, \widetilde{d_{nk_n}} \rhd \widetilde{e_n} \end{array}$$

where the function $\widetilde{\ }$ is the identity except for the assumption c_i and the conclusions $e_1, \ldots, e_n$:

$$\widetilde{c_i} = c_1 \vee \cdots \vee c_m \qquad\qquad \widetilde{e_j} = c_1 \vee \cdots \vee c_{i-1} \vee e_j \vee c_{i+1} \vee \cdots \vee c_m$$

It is debatable whether such inlining is a good idea. The resulting proof has a simpler structure, with fewer nested proof blocks. However, these nested blocks can help make complex proof more intelligible. Moreover, the n-fold repetition of the disjuncts $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_m$ clutters the proof and can slow it down.

The above procedure can be generalized to arbitrary case analysis blocks. I am grateful to Konstantin Korovin for insights that lead me to realize this. We can rewrite an m-way case analysis

$$\left[\begin{array}{r|c|r} [c_1] & \cdots & [c_m] \\ \Gamma_{11} \rhd e_{11} & \cdots & \Gamma_{m1} \rhd e_{m1} \\ \vdots & & \vdots \\ \Gamma_{1k_1} \rhd e_{1k_1} & \cdots & \Gamma_{mk_m} \rhd e_{mk_m} \end{array}\right]$$

into a sequence of m case analyses:

$$\left[\begin{array}{c|c|c|r|c|c|c} & & & [c_i] & & & \\ & & & \Gamma_{i1} \rhd e_{i1} & & & \\ & & & \vdots & & & \\ [e_{1k_1}] & \cdots & [e_{(i-1)k_{i-1}}] & \Gamma_{ik_i} \rhd e_{ik_i} & [c_{i+1}] & \cdots & [c_m] \end{array}\right]$$

Each of these has only one nontrivial branch and can be inlined, yielding a branch-free proof. For the spaghetti proof of the previous section, this process yields

$$\begin{array}{l} \rhd\ 7 \lor 8 \\ \blacktriangleright\ 2 \lor 5 \lor 6 \lor 8 \\ \blacktriangleright\ 2 \lor 4 \lor 6 \lor 8 \\ \blacktriangleright\ 2 \lor 3 \lor 4 \lor 6 \\ \blacktriangleright\ 2 \lor 3 \lor 4 \\ \blacktriangleright\ 2 \lor 3 \\ \blacktriangleright\ 1 \lor 3 \\ \blacktriangleright\ 1 \\ \blacktriangleright\ 0 \end{array}$$

The example shows clearly that we rapidly obtain large disjunctions. In practice, each of the disjuncts would be an arbitrarily complex formula. Local definitions could be used to avoid repeating the formulas, but the loss of modularity is deplorable. Indeed, similar concerns about Hoare-style proof outlines for separation logic have lead to the development of ribbon proofs [30], whose parallel ribbons evoke the branches of a case analysis.

If branch-free proofs are nonetheless desired, they can be generated more directly by iteratively "rewriting" the atoms, following a suggestion by Korovin. For example, starting from the sequent $\rhd 7 \lor 8$, rewriting 7 would involve resolving $\rhd 7 \lor 8$ with $7 \rhd 2 \lor 5 \lor 6$, resulting in $2 \lor 5 \lor 6 \lor 8$. In general, rewriting a tainted atom b_j within a sequent $\Gamma \rhd b_1 \lor \cdots \lor b_n$ involves resolving that sequent with the redirected sequent that has b_j in its assumptions. To guarantee that the procedure is linear, it suffices to rewrite atoms only if all their ancestors in the AIG have already been rewritten, thereby preventing multiple rewrites of the same atom.

5 Conclusion

This paper presented an algorithm that transforms proofs by contradiction as returned by automatic theorem provers into direct proofs. It sometimes introduces case splits but avoids duplicating inferences in the different branches of the split by joining again as early as possible. The resulting proofs are direct Isar proofs that have some of the structure Isabelle users have come to expect. The described procedure is admittedly fairly straightforward; it would not be surprising if it were part of folklore or a special case of existing work.

While the output is designed for replaying proofs, it also has a pedagogical value: Unlike Isabelle's automatic tactics, which are black boxes, the proofs delivered by Sledgehammer can be inspected and understood. The direct proof also forms a good basis for manual tuning. Further steps toward robust, intelligible Isar proofs, are described in a companion paper [25]. In future work, I am interested in transformations that increase proof readability and in the automatic discovery of concepts and lemmas, such as those available for Mizar proofs [16,17].

Acknowledgment. Tobias Nipkow, Lawrence Paulson, Geoff Sutcliffe, and Josef Urban motivated me to pursue this research. Andrei Popescu helped simplify the approach considerably. Charles Francis, sponsored by Google Summer of Code, developed a prototype. Sascha Böhme, Konstantin Korovin, Tobias Nipkow, Hans de Nivelle, Josef Urban, Tjark Weber, and two (still) anonymous reviewers produced a wealth of comments that helped enhance this text. I thank them all. The research was financed by the Deutsche Forschungsgemeinschaft (DFG) projects Quis Custodiet (grant Ni 491/11-1) and Hardening the Hammer (grant Ni 491/14-1).

References

[1] S. Autexier, C. Benzmüller, A. Fiedler, H. Horacek, and Q. B. Vo. Assertion-level proof representation with under-specification. *Electr. Notes Theor. Comput. Sci.*, 93:5–23, 2004.

[2] J. C. Blanchette. *Automatic Proofs and Refutations for Higher-Order Logic*. Ph.D. thesis, Dept. of Informatics, Technische Universität München, 2012.

[3] B. I. Dahn. Robbins algebras are Boolean: A revision of McCune's computer-generated solution of Robbins problem. *J. Algebra*, 208(2):526–532, 1998.

[4] M. Davis. Obvious logical inferences. In P. J. Hayes, editor, *IJCAI '81*, pages 530–531. William Kaufmann, 1981.

[5] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[6] G. Gentzen. Untersuchungen über das logische Schließen. *Math. Z.*, 39:176–210, 1935.

[7] X. Huang. Translating machine-generated resolution proofs into ND-proofs at the assertion level. In N. Y. Foo and R. Goebel, editors, *PRICAI '96*, volume 1114 of *LNCS*, pages 399–410. Springer, 1996.

[8] J. Hurd. First-order proof tactics in higher-order logic theorem provers. In M. Archer, B. Di Vito, and C. Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics*, number CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.

[9] S. Jaśkowski. On the rules of suppositions in formal logic. *Studia Logica*, 1:5–32, 1934.

[10] D. E. Knuth, T. L. Larrabee, and P. M. Roberts. *Mathematical Writing*. Mathematical Association of America, 1989.

[11] W. McCune. Solution of the Robbins problem. *J. Autom. Reasoning*, 19(3):263–276, 1997.

[12] A. Meier. TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level (system description). In D. McAllester, editor, *CADE-17*, volume 1831 of *LNAI*, pages 460–464. Springer, 2000.

[13] D. Miller and A. Felty. An integration of resolution and natural deduction theorem proving. In *AAAI-86*, volume I: Science, pages 198–202. Morgan Kaufmann, 1986.

[14] T. Nipkow. Programming and proving in Isabelle. `http://isabelle.in.tum.de/dist/Isabelle2013/doc/prog-prove.pdf`, 2013.

[15] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[16] K. Pąk. The methods of improving and reorganizing natural deduction proofs. In *MathUI10*, 2010.

[17] K. Pąk. Methods of lemma extraction in natural deduction proofs. *J. Autom. Reasoning*, 50(2):217–228, 2013.

[18] L. C. Paulson. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In B. Konev, R. Schmidt, and S. Schulz, editors, *PAAR-2010*, pages 1–10, 2010.

[19] L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In G. Sutcliffe, E. Ternovska, and S. Schulz, editors, *IWIL-2010*, 2010.

[20] L. C. Paulson and K. W. Susanto. Source-level proof reconstruction for interactive theorem proving. In K. Schneider and J. Brandt, editors, *TPHOLs 2007*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.

[21] F. Pfenning. Analytic and non-analytic proofs. In R. E. Shostak, editor, *CADE-7*, volume 170 of *LNCS*, pages 393–413. Springer, 1984.

[22] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Comm.*, 15(2-3):91–110, 2002.

[23] P. Rudnicki. Obvious inferences. *J. Autom. Reasoning*, 3(4):383–393, 1987.

[24] S. Schulz. System description: E 0.81. In D. Basin and M. Rusinowitch, editors, *IJCAR 2004*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[25] S. J. Smolka and J. C. Blanchette. Robust, semi-intelligible Isabelle proofs from ATP proofs. In J. C. Blanchette and J. Urban, editors, *PxTP 2013*, 2013.

[26] G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP data-exchange formats for automated theorem proving tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, volume 112 of *Frontiers in Artificial Intelligence and Applications*, pages 201–215. IOS Press, 2004.

[27] J. Urban, G. Sutcliffe, S. Trac, and Y. Puzis. Combining Mizar and TPTP semantic presentation and verification tools. In A. Grabowski and A. Naumowicz, editors, *Computer Reconstruction of the Body of Mathematics*, volume 18(31) of *Studies in Logic, Grammar and Rhetoric*, pages 121–136. University of Białystok, 2009.

[28] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, pages 1965–2013. Elsevier, 2001.

[29] M. Wenzel. Isabelle/Isar—A generic framework for human-readable proof documents. In R. Matuszewski and A. Zalewska, editors, *From Insight to Proof—Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar and Rhetoric*. University of Białystok, 2007.

[30] J. Wickerson, M. Dodds, and M. J. Parkinson. Ribbon proofs for separation logic. In M. Felleisen and P. Gardner, editors, *ESOP 2013*, volume 7792 of *LNCS*, pages 189–208. Springer, 2013.

[31] J. Zimmer, A. Meier, G. Sutcliffe, and Y. Zhan. Integrated proof transformation services. In C. Benzmüller and W. Windsteiger, editors, *IJCAR WS 7*, 2004.