BEST-PATH THEOREM PROVING: COMPILING DERIVATIONS

Martin Frické, SIRLS

The University of Arizona

1. Introduction

Increasingly, computers answer our questions in mathematics. However, mere answers are not enough in the general case. We need the actual justification, and we need explanatory insight. We want truths but we also want reasons. We want to 'Trust but verify'. And so we should. As Mill tells us in *On Liberty*, to do otherwise would make mathematics a superstition

> [were it not so]… the true opinion abides in the mind, but abides as a prejudice, a belief independent of, and proof against, argument -- this is not the way in which truth ought to be held by a rational being. This is not knowing the truth. Truth, thus held, is but one superstition the more, accidentally clinging to the words which enunciate a truth (Mill, 1869).

In addition to justification, we would like some understanding; that is, some explanation or insight as to what is going on. As Wolfram writes

> At some level the purpose of a proof is to establish that something is true. But in the practice of modern mathematics proofs have taken on a broader role; indeed they have become the primary framework for the vast majority of mathematical thinking and discourse. And from this perspective [some computer proofs and some proofs generated by automatic theorem proving] are quite

> unsatisfactory. For while they make it easy at a formal level to check that certain statements are true, they do little at a more conceptual level to illuminate why this might be so. … [Some general proofs] can be arbitrarily long, and can be quite devoid of what might be considered meaningful structure (Wolfram, 2002, p. 1156).

Another example is provided by some of Zalta's, work on computational metaphysics (Zalta, 2009, 2011). Some years ago Edward Zalta published some engaging results and theorems on the ontological argument, and similar. Recently these theorems were given to an automated theorem prover to proof check, and the theorem prover came up with new proofs that showed that some of the original premises and steps were redundant – truly an interesting result. However, the course of this discovery was not as quick or a smooth as it might have been. The researchers did not understand the raw output of the theorem prover in a way that the output could be seen as derivations that could be compared to the original derivations. They had to work by hand to convert the output of the theorem prover into perspicuous derivations that would be useful to them. They should not have to do this. The theorem prover should do it, or there should be plug-in filters that do the desired transformations.

So here is a general problem. There can be the need for computers to transform their internal reasoning and their internal processes of proof to output forms that are suited to the particular cognitive and epistemological needs of humans. Certainly in some empirical areas, for example rule-based expert systems, computer programs do provide a full rationale, but this is not so common with logic, mathematics, and theorem proving.

This paper is going to focus on real reasoning in logic. Often computers use algorithms and data structures that are different to those natural to humans. A computer may operate on full graph

theoretic structures, such as networks and trees, whereas humans are more comfortable with linear reasoning or paths. In cases like these, it is not so much that the computer has to explain what it is doing, rather it should produce for us separately proper stand alone justification that we can follow and be convinced by.

There is an example subject area central to philosophy and logic. First order predicate calculus (FOPC) is, or may be, a universal logic, perhaps even *the* deductive logic for mathematics and all formal reasoning ((Lindstrom,1969), (Manzano, 1996), (Quine, 1970)). And there are different styles of theorem-provers for FOPC (Robinson & Voronkov, 2001). These work in different ways. Some use linear methods and heuristics. These might have a fact queue and a problem or interest queue and cycle back and forth between the two. Such methods typically produce proofs or output that is accessible to humans. Other more common automatic theorem-provers use non-linear methods for their data structures and algorithms. For example, many employ resolution, which rests on unification, Skolem functions, and a reduced connective set (Robinson & Voronkov, 2001). Yet others use tableaux (D'Agostino, 1999). All of these latter are of themselves somewhat opaque to humans.

2. Theorem Proving in FOPC

In the case of FOPC, what is it that humans understand and can be convinced by? What is it that human logicians, or mathematicians, would like?

Humans reason with a full connective set, with a full ‘natural deduction’ rule set for that full connective set, or something similar, and they reason linearly. The problem, then, is this. Take any ordinary rule set used by students, professors, and other humans, to write and justify their FOPC derivations; then, produce a mechanical or computational deriver or automated theorem prover for FOPC which outputs theorems in the favored rule set, any rule set. The output theorems should have a quality to them; they should be similar to or better than those produced, say, by a Professor of Logic.

First thoughts about this in terms of linear derivations are intimidating. With most rule sets: any statement that has a derivation has infinitely many different derivations; for any length of derivation there are infinitely many different derivations of that length; for any line of a derivation there are infinitely many possible next lines; for any line in the body of a derivation there are infinitely many immediate predecessor lines; if the problem is conceived of as an artificial intelligence search problem, then there is no proper evaluation function to indicate the distance of the current state from the goal state; there is not even any general way of knowing whether the goal state can be reached; theoremhood is not decidable; and so on.... There seems no obvious way of controlling the search and terminating gracefully. Certainly some counter moves can be made here. For example, possibilities can be reduced if intelligent use is made of subformula relations between the conclusion and premises, as is done, for instance with ‘normal’ proofs in natural deduction and intercalation calculi (Sieg & Byrnes, 1998) or with ‘analytic’ tableaux (Smullyan, 1968). Nevertheless, this seems to be a problem deserving of respect.

3. Tableau Theorem Proving

Since the publication of Raymond Smullyan's book in 1968, tableaux have increasingly become more common ((D'Agostino, 1999), (Smullyan, 1968), (Fitting, 1996, 1998)), (Howson, 1997), (Jeffrey, 1967)). Tableaux are an interesting case to study. Mostly they are decompositional (which lends itself to designing algorithms that terminate). Decompositionality is attractive. For any particular formula, generally there is only one rule that applies (which works on the main connective or negation of the main connective). And that rule breaks the formula into shorter subformulas (which eventually themselves get decomposed if they contain suitable connectives). In a stroke many of the earlier worries are done away with. There are not infinitely many possible next steps. Some of the tableau rules are not decompositional (Fitting, 1998): for example, the identity rules are not.[1] Universal instantiation is also a special case; it decomposes to an instance, but it leaves the universal formula available for a further instantiation. We know from meta-theorems about tableaux that instantiation to every closed term in every relevant branch is enough ((Howson, 1997), (Jeffrey, 1967), (Bergmann et al, 1998)). That will eventually close a branch and, by closing branches, will close a tree—*if the tree can be closed*. However, it is certainly possible for a universal instantiation to extend a branch yielding an existential formula which itself will decompose *to yield a new constant (closed term) in the relevant branch* which in turn requires that the original universal be instantiated to that new term i.e. we have a growth generator. Sometimes these feedback generators can be by-passed (a branch can close without appeal to them). Sometimes a feedback loop can be small enough to be

[1] We can omit discussion of identity from this paper; it is something of a special case and none of the techniques explained here have any distinctive insights concerning it.

detected and a sound conclusion made that a containing branch cannot close. But, in the general case, the loops can be bigger than any particular finite size we choose to look at. We know something like this has to happen. FOPC is undecidable, so there cannot be a mechanical decision process. Even so, tableaux are attractive.

There are still tricky decisions that range across both tableaux and the other rules sets. For example, in both, there is usually a choice of what to do, of which formula or formulas to work on, to use for extending or rewriting. In an ordinary linear proof, we always have choices (for example, of whether to conjoin line 1 with itself, or to conjoin line 1 with line 2); similarly in tableaux (almost always there will be some stages in the growth of the tableau where there is a choice among which formula to use for the next extension). There is bounded indeterminism (Fitting, 1998). And different choices may well produce structurally different tableaux. For example, if you always 'split' a tree, first, rather than 'straight extend' it, you will get broad or fat trees. If, so to speak, the 'wrong' choice is made during these extensions, the eventually tableau proof is not a mistaken proof, but it may have redundant steps or inelegancies; there could be the need for optimization if a particular style of tableau is the target.

We will assume algorithms are available for closing tableaux (D'Agostino, 1999).[2]

[2] Space precludes further detailed description of those in this paper, and the question of automatically closing tableaux is not the present target.

4. Transforming Tableaux to Linear Proofs: A Construction

As we stated, we would like linear proofs. Here is a suggestion: *transform the closed tableaux into linear proofs.*

Linear proofs here include three sub-genres: Hilbert-Frege axiomatic proofs (consisting of axioms and rules of inference), Natural Deduction proofs (which have the adding and dropping of assumptions, and usually one rule for Introducing a particular connective and another rule for Eliminating that connective), and Sequent calculi (where all the sequents are unconditional truths) (Fitting, 1996).

Transforming tableaux to linear proofs is not without precedent. There is the general research area of proof transformation (a good example here is 'cut-elimination' (Smullyan, 1968), (Fitting, 1996), (Gallier, 1986)). And also, within research on tableaux, there has long been the awareness of the close relationship between tableaux and sequent proofs ((Smullyan, 1968), (Beth, 1955), (Gentzen, 1935), (Hähnle, 2001), (Letz, 1999)). But here the transformation is being done in perhaps a novel way and to novel purpose. The focus is with actual running computer programs and algorithms, which will serve as automatic theorem provers and which will generate the 'best' linear proofs for any desired linear rule set.

We have not yet decided on specific tableau rules, nor on specific linear rules. Even so, this transformation can be done, from the tips of the branches, i.e. from the leaves, to the root. Here is one way to do it.

For standard proof tableaux, the tableaux are, in a graph theoretic sense, node-labeled trees (connected, acyclic graphs with one preferred node, the 'root'). Each node is labeled with a formula: there is a node for every formula. Then tableaux can be signed or unsigned (Smullyan, 1968). A signed tableau has its component formulas tagged as being 'true' or as being 'false' (and an unsigned one does not). With a signed tableau, the aim is to extend all the root formulas and component formulas down a branch until there are enough atomic formulas signed true, or signed false, to guarantee that all the formulas in that branch have truth values as signed (this might not be possible if the root formulas are not simultaneously satisfiable). Unsigned tableaux do not have the signing, and then the assumption, or aim, is to get every formula to be true. Signed tableau can be converted into unsigned tableau merely by substituting the negation of a formula for any formula signed false (and unsigned can be converted into signed by signing false any formula with negation as its main connective and removing the negation).

For an argument, or tentative theorem, say

$P_1, P_2, \ldots P_n \therefore C$

where the P_i are premises (or assumptions) and the C is the conclusion or conjectured theorem. A tableau is constructed with the Ps as separate nodes signed true and the C, a node signed false. For suitable tableau rules, this will find the counter-example, if there is one.[3]

Here is a simple example, the argument

M&N $\therefore$ N&M

[3] And the technique is easily adapted if the focus of interest is consistency or simultaneous satisfiability etc.

can have a tableau that proceeds

1	M&N	SM
2	~(N&M)	SM

Figure 16.1: Stage1 of the construction of a tableau for M&N ∴ N&M

1	M&N √	SM
2	~(N&M)	SM
		
3	M	1 &D
4	N	1 &D

Figure 16.2: Stage 2 of the construction of a tableau for M&N ∴ N&M

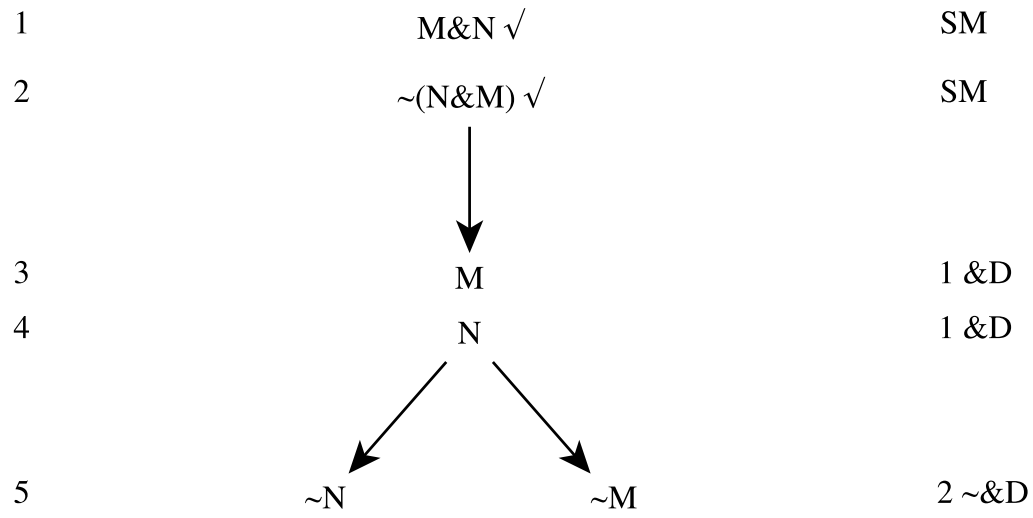

Figure 16.3: Stage 3 of the construction of a tableau for M&N ∴ N&M

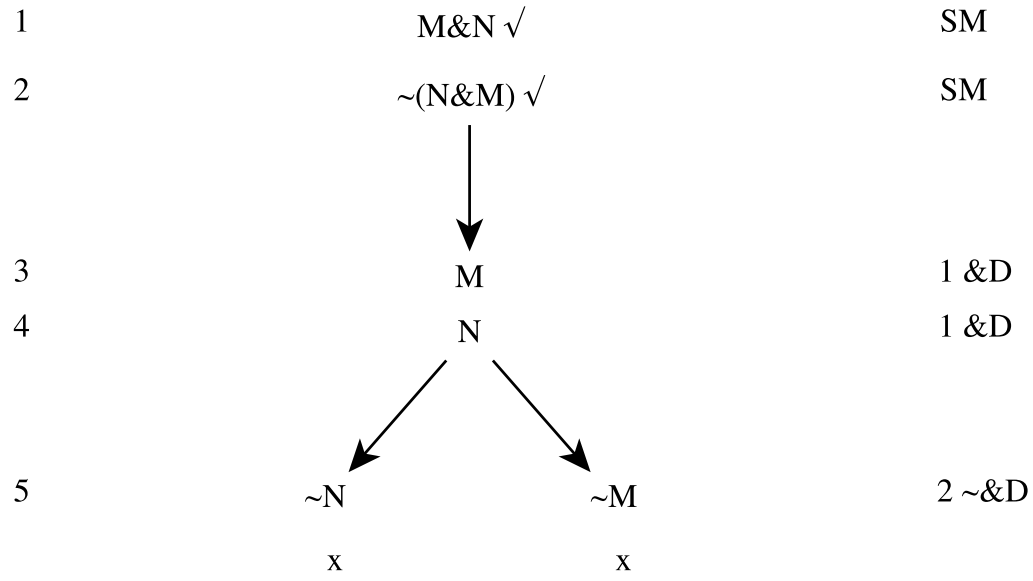

Figure 16.4: Stage 4 of the construction of a tableau for M&N ∴ N&M

This is an unsigned tableau (and line 2 is the negated conclusion, showing the desire to make the conclusion false). This tableau has six nodes in it (two for the root formulas, then application of the rules, in two steps, generates four more).

By using an additional, or alternative, labeling system this tableau can be replaced by, or transformed into, a different equivalent one, which, in this particular case, will have only four nodes in it. The motivation for the transformation is twofold. Branches need to be closed, that is one aim. We would like to carry branch information with us during the extension steps so that we can close a branch merely by looking at the label on the relevant node (instead of having to search the branch). Then, later and independently, the label will be used to construct the linear proof. The new labeling system has labels, which are sets or lists of formulas; and the formulas themselves, in these labels, are signed.

There is an additional novelty in the transformed tableau. In it, only leaf nodes can be extended. (A leaf node is a node without children.) To clarify this, using the above example. In the

transition from Stage 2 to Stage 3 the formula ~(N&M) of line 2 is extended to produce the two formulas of line 5. Line 2, or the node corresponding to it, is *not* a leaf node; it has line 3, or M, or that node, as its child. This type of extension of 'internal' non-leaf nodes, to all branches, containing them is standard in tableau techniques. But here only leaf nodes are extended. This makes two kinds of differences. The first is an algorithmic one. With standard tableau techniques, there is a bounded indeterminism in the choice of which formula to extend, and then, with that choice made, a determination has to be made of all (open) branches that contain the formula, as the extension goes to all of the branches containing it. But any leaf is in only one branch, a leaf determines its own branch uniquely, so any extension is easier. The indeterminism then becomes an indeterminism over which leaf to extend (if there are several available leaves). It turns out that this is irrelevant, so the leaves can be processed breadth or depth first. The indeterminism surfaces in what is done with a particular leaf, but more of that later. The second kind of differences arises with how this 'leaf-extension-only' promotes the desired transformation to the linear derivation.

The new labeling is as follows. The (single) root node is labeled

$P_1, P_2, \ldots P_n \therefore C$

the Ps are the premises (or signed true formulas) and the C is the conclusion (or a signed false formula).[4] So the label is the set of signed formulas

$\{True(P_1), True(P_2), \ldots True(P_n), False(C)\}$

[4] Although the premises are being depicted here as something of a list, conceptually they are a set. (There is no need to complicate this with 'structural' rules.)

but, for heuristic purposes, it is clearer to introduce the therefore symbol '∴' and to think of the label as

$P_1, P_2, \ldots P_n \therefore C$

The labeling for each of the lower nodes is obtained by transformation of the relevant parent labels. Most of the time the individual formula used for the tree extension step itself will be dropped from the set of formulas in the label for the extension, and the products for the extension will be incorporated (the exception is extension from the universal quantifier where both the formula and its product will be in the label for any children). That the original extended formula is dropped is just a book-keeping implementation issue. It is standard with ordinary tableau techniques to mark a formula as 'dead' when it has been used. This is merely a non-essential counterpart to that.

Schematically, the label for a 'straight ahead' extension might look something like

$P_1 \& P_2, P_3, \ldots P_n \therefore C$

Extended to

$P_1, P_2, P_3, \ldots P_n \therefore C$ (*note here that the $P_1 \& P_2$ is dropped from the P_i*)[5]

And the labels for 'splitting' extension might look something like

[5] Here the notation '$P_1 \& P_2$' is being used as a pattern to show a formula with & as its main connective and P_1 and P_2 as subformulas. The premises are a set, but writing the formula that is being processed first helps draw attention to it.

P_1 v P_2, P_3, … P_n ∴ C

Extended to

P_1, P_3, … P_n ∴ C (*note here that the PjvPk is dropped from the Pi*)

P_2, P_3, … P_n ∴ C (*note here that the PjvPk is dropped from the Pi*)

Some extensions steps can similarly transform C, the signed false formula. Here is a 'splitting' example

P_1, P_2, … P_n ∴ (C_1 & C_2)

Extended to

P_1, P_2, … P_n ∴ C_1

P_1, P_2, … P_n ∴ C_2

In an ordinary tableau, a branch can, and should, be closed when it contains a formula that tries to be both true and false. With this new labeling scheme it is not so much branches that are closed, but instead individual leaf nodes are closed. That can occur in two ways: a formula can be signed true in one place in a label and signed false elsewhere, or a formula and its negation can both be signed true. So a label for a 'closure leaf' has form, either

P_1, P_2, P_i … P_n ∴ P_i (*P_i signed both true and false.*)

or

$P_1, P_2, P_i, \sim P_i \ldots P_n \therefore$ <anything> (*P_i and $\sim P_i$ both signed true*)

In the example

M&N ∴ N&M

is the label for the root node

M, N ∴ N&M

is the label for the single node for lines 3 and 4.

M, N ∴ N (* N is signed true as a premise and false as a conclusion *)

and

M, N ∴ M (* M signed true and false*)

are the labels for the two nodes of line 5.

And here is the newly constructed tableau depicted in full.

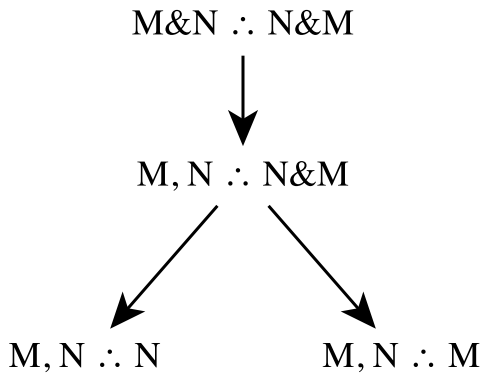

Figure 16.5: Newly Constructed Tableau for M&N ∴ N&M

Individual labels can be large and complex. A particular label could easily have 50 'premise' formulas left of the therefore symbol. This scheme is not designed for humans to write out with a pencil on a sheet of paper. This is part of a computer program. And a computer can very easily copy or share a label, or parts of a label.

A remark is perhaps in order here on the relationship between this labeling and ordinary tableaux. An ordinary tableau is a set of branches and, in turn, a branch is a set of formulas. Here a tableau is a set of branches, each branch has a unique leaf, and, in turn, a leaf is a set of (signed) formulas.

Another observation should be made about the process of extension. In an ordinary tableau, each node usually has a unique extension (exceptions being identity, Universal Instantiation, etc.). And there is indeterminism as to which node in a branch to extend. With the new labeling system, usually a leaf node can be extended in several different ways. For example, the leaf node

Fa&Gb, (∀x)(Fx⊃Hx) ∴ Gb&Ha

can be extended via the main connective *and* or the main connective *the universal quantifier* from among the 'premises' or via the main connective *and* of the 'conclusion'. The indeterminism of extension re-asserts itself at the leaf extension level.

The new labels are intended also to play a role depicting some Gentzen sequents (Gentzen, 1935). A Gentzen sequent has the form

$A_1, A_2, \ldots A_n \therefore S_1, S_2, \ldots S_n$

Where the A_i are 'antecedents' and Si 'succedents'. Without loss of generality for what we are going to do, we can assume that there is no more than one succedent. There can be zero or more antecedents.

The Gentzen sequents for the example are, going down the tree,

M&N ∴ N&M

M, N ∴ N&M

M, N ∴ N and M, N ∴ M

Then, as a result or provable theorem,

a) there is an immediate (and trivial) linear derivation of the sequent corresponding to any closure leaf (either a premise is identical to the conclusion, or there is a Reductio proof from two premises which contradict), and

b) if there are linear derivations of the sequent labels showing the results of the children of a single tableau decomposition step, then, from these, a linear derivation can be constructed of the sequent corresponding to the parent node used in that decomposition.

Thus, constructed derivations can be driven in the reverse direction up the tree, from the leaves to the root, and this provides a derivation of the root sequent (which is what was sought).

Conceptually, the linear derivations are assembled as follows:

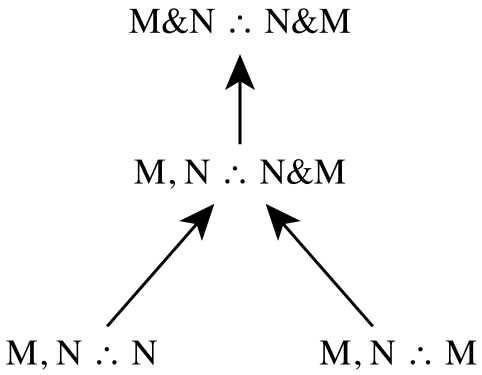

Figure 16.6: The Tableau Guide for constructing a linear proof of M&N ∴ N&M

In the example, the two leaf nodes have linear proofs

1	M	Ass
2	N	Ass
3	N	2 R

Figure 16.7 Linear Proof of the leaf M,N ∴ N

1	M	Ass
2	N	Ass
3	M	1 R

Figure 16.8 Linear Proof of the leaf M,N ∴ M

Then, these two linear proofs can be merged together to form a linear proof of

M, N ∴ N&M

the sequent for the node for lines 3 and 4.

1	M	Ass
2	N	Ass
3	N	2 R
4	M	1 R
5	N&M	3,4 &I

Figure 16.9 Merging the leaf linear proofs to obtain a linear proof of M, N ∴ N&M

Finally a linear derivation for the sequent M, N ∴ N&M can be converted into a linear derivation for M&N ∴ N&M, which is the sequent corresponding to the root node.

1	M&N	Ass
2	N	1 &E
3	M	1 &E
4	N&M	2,3 &I

Figure 16.10 Converting the linear derivation of M, N ∴ N&M to one for the root M&N ∴ N&M

This is a very simple example. But the technique has been proved to be appropriate and correct for the full rule set including quantifiers and sub-proofs, and it has been implemented and used over many years ((Copeland & Murdoch, 1989), (Frické, 1989a, 1989b)) with many different tableau rules and many different linear rules (you can see it at http://softoption.us in use on a variety of rule sets).

The technique is not tied to any particular Gentzen sequent calculus or its associated rule set. But the framework of thinking in terms of sequents is convenient for identifying assumptions and the nature of the proof steps.

There is some benefit to comparing this technique with ordinary sequent calculus proofs as devised in (Gentzen, 1935) and illustrated, for example, in (Fitting, 1996, p. 94). A sequent

calculus proof or derivation is tree of sequents in which the leaves are axioms, the root is the theorem, and the derivation proceeds from the leaves to the root. It is also true that to an extent Gentzen used sequent trees as an intermediary in proof transformations from one system to another. However, a) the present labeling scheme rests on semantics not syntax, i.e. whether the formulas can have truth values as signed, b) sequent tree proofs can extend from internal nodes, the present labeling extends only from leaves, c) the leaf closing via premise contradiction could not normally be done in a Gentzen sequent proof because that is not an axiom (although it could be added as an axiom), d) the present construction is not, and is not intended to be, a proof in itself, instead it is a stepping stone to the generation of proofs (be they Hilbert-Frege proofs, Natural Deduction Proofs, or Sequent proofs). In sum, standard Gentzen sequent proofs are a historical antecedent of the present labeling transformation, but what is being undertaken here is not identical to producing sequent proofs. What is being offered here is an algorithm.

Quantifiers add a level of complexity. In the running example programs, Skolem functions are used for Existential Quantifiers, a standard technique ((Fitting, 1996), (Gallier, 1986), (Hähnle, 2001)). And Unification, and/or caching of prior instantiations, are used for Universal Quantifiers ((Fitting, 1996), (Gallier, 1986), (Hähnle, 2001)). So, a Universal Instantiation-like tableau step is the following

$(\forall x)$<scope x>, P_2, … P_n ∴ C

Extended to

$(\forall x)$<scope x> ,<scope t/x>, P_2, … P_n ∴ C (*note here that the $(\forall x)$<scope x> is retained in the P_i*)

The Universal formula is retained in its entirety, and a legitimate substitution instance of its scope is added. There is an implementation question that arises here. When Universal Instantiation is done in a standard tableau, when trying to close that tableau,

> there is a need to instantiate the Universal formula at least once
>
> there is never any need to instantiate it twice to the same closed term,
>
> and there is never any need to instantiate to anything other than the closed terms that occur in its branch.

A common difficulty is that, for a computer program, it is often unclear, at the time of the desired instantiation, which instance is required. A response is to do the instantiation 'lazily', that is, to instantiate initially to a free variable (Fitting, 1996), and then to allow Unification to indicate later what that instantiation should be. So for example, a label/node like this

(∀x)Fx, ~Fc ∴ C

might be instantiated to

(∀x)Fx, Fx, ~Fc ∴ C

and then the leaf closing algorithm will detect, via Unification, that were the free variable x to be valued as the constant term c, the leaf can be closed. So, the earlier Universal instantiation needs to be to the constant c. There is computer work in the background here. At the end of the day, though, a Universal Instantiation extension is to one specific term, and the explanation offered in this paper will proceed on that basis.

Little has been said about either the tableau rules sets or the linear rules sets. The fact is: the technique is general; providing the tableau and linear sets are complete, a minimal adequacy requirement, the technique can be used. Actual running programs that have been written usually use twenty tableau extension rules: one each for {&, v, ⊃, ≡, ∀, ∃} and {~~, ~&, ~v, ~⊃, ~≡, ~∀, ~∃} signed true (i.e. from among the premises) and similarly, with some optimizations, for those types of formulas signed false (i.e. as the conclusion). Many of these rules are very similar, in terms of what they do, so there is common algorithmic code, but they are tagged individually to assist with the later generation of the linear proofs. Several different examples of linear rule sets have been used (often with slight tuning of the tableaux rules due to different requirements with, for example, Universal Generalization).

This construction will yield linear proofs, even though the underlying theorem prover is a tableau prover. And, actually, the same tableau data structure can generate different linear proofs for the different linear rule sets— essentially a filter can be applied to produce the desired output.

5. Improving the Linear Proofs

What has been done thus far will produce linear proofs, but not necessarily 'good' proofs.

Certain improvements can be made blind, so to speak, for example, redundant and unused lines can be removed. For example, a raw proof of

M&N ∴ N

might come out with the mildly inelegant derivation

1	M&N	Ass
2	M	1 &E
3	N	1 &E

Figure 16.11 An inelegant derivation of M&N ∴ N

What has happened here is that the tableau extension step takes by rote both conjuncts when extending a conjunction (it takes both M and N when extending M&N), and the conversion to linear proof retains these. Apparently there are two ways around this: to pay attention to goals when extending tableau (e.g. to extend only to N in this case) or to clean up afterwards. Implemented systems do the latter. They do a dependency analysis, after the case. Line 3, the conclusion line, depends on line 1. Nothing depends on line 2, so line 2 will be discarded to yield

1	M&N	Ass
2	N	1 &E

Figure 16.12 Discarding line 2 to improve the derivation of M&N ∴ N

In addition to dependency analysis, there are deeper, more structural issues, to consider.

There are some other fairly obvious optimization steps that can be made. For example, if the theorem prover is doing a Reductio proof and it has available a negated compound formula, for instance ~(M&N), it will be alert to the possibility that this formula is the Negative Horn of the contradiction and thus consider whether (M&N) can be proved as the Positive Horn (as opposed to, say, breaking ~(M&N) apart eventually to get a contradiction in terms of literals).

So-called β-extensions, or splitting extensions ((Smullyan, 1968), (Fitting, 1996)), also merit some special attention. For example, (A v B) among the premises can be split into two branches, one starting with A and the other with B. An output derivation corresponding to this step might well be Modus Ponens (if ~A can be proved on its own), Modus Tollens (if ~B can be proved on its own), or Or-Elimination (if neither ~A nor ~B is available as a standalone proof). Modus Ponens is common, easy, and quick; Modus Tollens is less common, but also easy and quick. Genuine Or-Elimination is somewhat involved. So, β-extension is more attractive if there is a known closing lemma to one of the branches. The tableau algorithm tests for this. The technique here is related to 'cut' or the use of lemmas ((Smullyan, 1968), (Fitting, 1996), (Gallier, 1986)). The lemmas are not lemmas from outside the proof under consideration, using new predicates, constants or formulas. They are lemmas using sub-formulas of what is there. Small trees, produced on the side, are spliced in to close some of the branches. So one example heuristic here is: if you are considering extending from A ⊃ B , do it if you have A or if you have ~B, otherwise just wait on it – you might not need to do it, and other possibilities may emerge later. (Remember the parable. The King was going to execute a man. The man said 'If you do not

execute me now, I will teach your horse to whistle within a year'. (Who knows what will happen in a year? The King may die. The horse may die. The horse may learn to whistle).)

Any proof can be done in different ways. Even with tableaux, there are usually different tableaux that can be built from the same root (and these different tableaux will usually produce different linear proofs). A certain amount of massaging can take place on the rule sets and good rewrites, using 'laws of thought', can help (for example, a rewrite, or replacement, de Morgan law can cut 12 lines out of many proofs).

Of course, for most roots, once it is known that a root node leads to a closed tableau, it is easy to generate most all the closed tableaux the root can have, and thus most all linear proofs, available within the constraints of the construction. Then the British Museum algorithm (Black, 2005), exhaustive search, would produce the 'best' proof (however 'best' was construed). This approach would run into complexity issues—it overwhelms computing resources for large cases. (The qualifications. 'most roots', arise because some roots can have infinitely many closed tableau, for example, when there is a loop generator in there or when there is pointless instantiation of a Universal Quantifier.) A researcher always aspires to produce a short cut to a British Museum algorithm. Can this be done here?

There is an answer, perhaps a surprising one. The notion of best proof has an empirical component. All possible theorems, or all possible proofs, distribute evenly, in some sense or other, among the possible uses of inference steps. As examples, there are as many Modus Tollenses as Modus Ponenses, there are as many inferences from A, B to A&B as there are inferences from A, B to B&A, and there are as many inferences from A to AvB as there are

inferences from A to BvA. However, with the proofs of interest to humans this uniformity of distribution does not hold. Humans favor Modus Ponens over Modus Tollens; humans do not like Reductio; and there are many other preferences. The author's research has taken many sample questions and proofs from logic texts, scholastic tests, and similar sources, and analyzed the best proofs for them ('best' here usually meaning shortest). This empirical research gives rise to some suggestions on heuristic, which will be given in Appendix A. Whether this particular empirical data is respected is secondary to the general point: humans are attuned to certain forms inference, empirical research can tell us what those are, and that empirical research can educate as to how tableau theorem provers, and their symbiotic liners counterparts, should run.

6. Conclusion

Tableau theorem provers, coupled with transformations to linear proofs and empirically sourced heuristic, can provide transparent and accessible theorem proving.

Appendix A: The Extension Heuristic

This is a heuristic (and algorithm) for the construction of tableaux that seems to do well, on average, on the linear proofs that it will generate, across a wide range of proofs. Details can differ, depending on what rules are available. But this is the core.

The overriding control principle looks through everything available, as extendable formulas in a leaf sequent label, in a series of 'sweeps'.[6]

If any sweep produces a step or extension that can be made, that step is made and the whole series of sweeps is started again on the resulting formulas.

Sweep 1:

> any formula signed true that has **implication** as its main connective, provided that its 'if clause' can be proved (this is tested for, recursively). This is Modus Ponens.

Sweep 2:

> any formula signed true that is a **double negation**, an **and**, or an **equivalence**. These steps are Double Negation Elimination, And Elimination, and Equivalence Elimination.

Sweep 3:

> any formula signed false that is an **implication**, a **negation**, or an **equivalence**. These steps are Conditional Proof, Reductio, and Equivalence Introduction

[6] The extension steps correspond to one or more lines in a linear proof.

Sweep 4:

any formula signed true that is an **or**. This step is **Or Elimination**.

Sweep 5:

any formula signed true that has the **existential quantifier** as its main connective, and which can be instantiated **without change of bound variable**. This step is **Existential Instantiation (without change of variable).**

Sweep 6:

any formula signed true that has the **existential quantifier** as its main connective. This step is **Existential Instantiation** (with change of variable—sweep 5 picks up the ones that do not need it).

Sweep 7:

any formula signed false that has the **universal quantifier** as its main connective, and which can be generalized to **without change of bound variable**. This step is **Universal Generalization (without change of variable).**

Sweep 8:

any formula signed false that has the **universal quantifier** as its main connective. This step is **Universal Generalization** (with change of variable—sweep 7 picks up the ones that do not need it).

Sweep 9:

any formula signed false which is a **double negation**, an **and**, or an **or**. These steps are **Double Negation Introduction**, **And Introduction**, and **Or Introduction**.

Sweep 10:

any formula signed true that has the **universal quantifier** as its main connective, and which can be instantiated **without change of bound variable**. This step is **Universal Instantiation (without change of variable)**.

Sweep 11:

any formula signed false that has the **existential quantifier** as its main connective. This step is **Existential Generalization**.

Sweep 12:

any formula signed true that has the **universal quantifier** as its main connective. This step is **Universal Instantiation**.

Sweep 13:

any formula signed true that is the **negation of a universally quantified formula** or the **negation of an existentially quantified formula**. This step is **Negation of Quantifier** (usually a rewrite).

Sweep 14:

any formula signed true that is an **implication**, the **negation of an and**, **the negation of an or**, **the negation of an equivalence** or **the negation of a implication**. These steps are usually complex lemmas.

Sweep 15:

any formula signed true EXCEPT atomic and the negation of atomic formulas. (The default catch-all.)

Sweep 16:

any formula signed false EXCEPT atomic and the negation of atomic formulas. (The default catch-all.)

Appendix B: A Small Example

Here is one running implementation being asked to permute existential quantifiers

1	?	? <<
2	(∃x)(∃y)Fxy≡(∃y)(∃x)Fxy	?

Figure 16.13 Starting to automatically derive (∃x)(∃y)Fxy⊃(∃y)(∃x)Fxy

And this is the output when using Introduction-Elimination rules

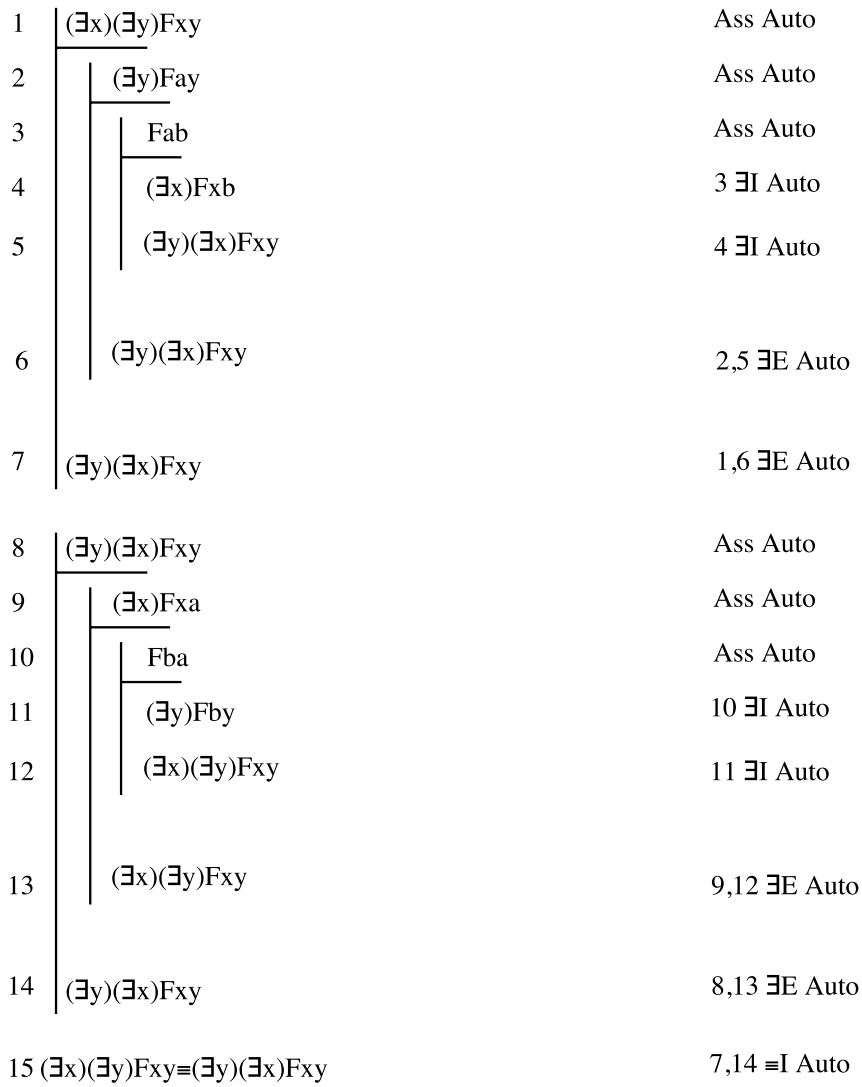

Figure 16.14 An automatic derivation of (∃x)(∃y)Fxy⊃(∃y)(∃x)Fxy using Introduction-Elimination Rules

and here is the same theorem/proof provided using Copi-style linear rules

1	(∃x)(∃y)Fxy	AP Auto
2	(∃y)Fxy	1 ∃E Auto
3	Fxy	2 ∃E Auto
4	(∃x)Fxy	3 ∃I Auto
5	(∃y)(∃x)Fxy	4 ∃I Auto
6	(∃x)(∃y)Fxy⊃(∃y)(∃x)Fxy	5 CP Auto
7	(∃y)(∃x)Fxy	AP Auto
8	(∃x)Fxy	1 ∃E Auto
9	Fxy	2 ∃E Auto
10	(∃y)Fxy	3 ∃I Auto
11	(∃x)(∃y)Fxy	4 ∃I Auto
12	(∃y)(∃x)Fxy⊃(∃x)(∃y)Fxy	11 CP Auto
13	((∃x)(∃y)Fxy⊃(∃y)(∃x)Fxy).((∃y)(∃x)Fxy⊃(∃x)(∃y)Fxy)	6,12 Conj Auto
14	(∃x)(∃y)Fxy≡(∃y)(∃x)Fxy	13 Equiv Auto

Figure 16.15 The same derivation using Copi style Rules

These are quite different in that Copi brings in equivalence by definition, or replacement, from conjoined conjunctions, and it has its own version of Existential Instantiation (which it calls ‘Elimination’). The Intro-Elim rule set makes much more extensive use of sub-proofs.

This is a small example. The point of it is not that one rule set is better than another. Rather it is that the User can choose whatever they would like, and suitable software can provide it for them.

The reader of this paper can try examples of their own derivations at

http://softoption.us/content/node/165 for Intro-Elim

and

http://softoption.us/content/node/286 for Copi

References

Bergmann, M., Moor, J. and Nelson, J. (1998). *The Logic Book* 4 ed. McGraw-Hill.

Beth, E.W. (1969). Semantic Entailment and Formal Derivability. In Hintikka, J. (ed.), *The Philosophy of Mathematics*. Oxford University Press, 9-41.

Black, P.E. (2005). *British Museum technique*. Dictionary of Algorithms and Data Structures. Retrieved 7/2009, from http://www.itl.nist.gov/div897/sqg/dads/HTML/britishMuseum.html.

Copeland, B.J. and Murdoch, D.R. (1991). The Arthur Prior Memorial Conference: Christchurch 1989. *The Journal of Symbolic Logic 56(1)*, 372-82.

D'Agostino, M., et al., eds. (1999). *Handbook of Tableau Methods*. Kluwer Academic Publishers.

Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving* 2 ed. Springer-Verlag: Berlin.

Fitting, M. (1998). Introduction. In D'Agostino, M., et al. (ed.), *Handbook of Tableau Methods*. Kluwer Academic Publishers.

Frické, M. (1989a). *Derivation Planner*. Unisoft, Dunedin.

Frické, M. (1989b). *Deriver Plus*. Kinko's Academic Courseware Exchange: Ventura, CA.

Gallier, J.H. (1986). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper Row: New York.

Gentzen, G. (1935). Investigations into logical deduction. In Szabo, M.E. (ed.), *The Collected Papers of Gerhard Gentzen*. North-Holland.

Hähnle, R. (2001). Tableaux and Related Methods. In Robinson, J.A. and Voronkov, A. (ed.), *Handbook of Automated Reasoning*. MIT Press.

Howson, C. (1997). *Logic with trees: An introduction to symbolic logic*. Routledge: London.

Jeffrey, R.C. (1967). *Formal Logic: its Scope and Limits*. McGraw Hill.

Letz, R. (1999). First-order Tableau Methods. In D'Agostino, M. et al. (ed.), *Handbook of Tableau Methods*. Kluwer Academic Publishers.

Lindstrom, P. (1969). On extensions of elementary logic*, Theoria 35*, 1-11.

Manzano, M. (1996). *Extensions of First-Order Logic*. Cambridge University Press: Cambridge, UK.

Mill, J.S. (1869). II. Of the Liberty of Thought and Discussion. In: J.S. Mill (ed.), *On Liberty*. Longman, Roberts & Green: London.

Quine, W.V. (1970). *Philosophy of Logic* 2 ed. Oxford University Press: Oxford.

Robinson , J.A. and Voronkov, A., eds. (2001). *Handbook of Automated Reasoning*. MIT Press.

Sieg, W. and Byrnes, J. (1998). Normal natural deduction proofs (in classical logic). *Studia Logica 60*, 67-106.

Smullyan, R. (1968). *First Order Logic*. Springer: Berlin.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Inc.

Zalta, E.N. (2009). *Achieving Leibniz's Goal of a Computational Metaphysics*, in *The 2009 North American Conference on Computing and Philosophy*. Bloomington: Indiana.

Zalta, E.N., Fitelson, B., & Oppenheimer, P. (2011). *Computational Metaphysics*. Retrieved 18/11/2011, from http://mally.stanford.edu/cm/