Dependency and Scope: On the Translation Between Lemmon and Fitch Proofs

Hans Halvorson

Draft

Abstract

Lemmon-style and Fitch-style natural deduction are usually treated as notational variants of one another. They are, but the sense in which they are is more delicate than that description suggests, and the delicacy is instructive. A Lemmon line carries an explicit and exact record of the assumptions it rests on; a Fitch line carries no such record, and its dependencies are bounded, but not determined, by the subproofs enclosing it. We show that translation from Fitch to Lemmon is total and canonical, and that the dependency column of a Lemmon proof is therefore redundant. Translation in the other direction is not *positional*: there are Lemmon proofs whose lines cannot be carried over one for one, in order, to any Fitch proof. We exhibit two such proofs and identify the two obstructions responsible. Both, we argue, are consequences of a single structural difference. Each notation presents a derivation—which is a tree—as a directed acyclic graph, by writing each line once and citing it thereafter; but where Lemmon permits a line to be cited by any later line whatever, Fitch permits citation only along the paths its nesting of subproofs allows. The obstructions arise exactly when no such nesting accommodates the sharing the source proof uses, and they dissolve on unfolding, since a graph without sharing is a tree and a tree constrains nothing. We give a construction translating any Lemmon proof by way of its unfolding, and then show that it fails, at exactly one rule. Universal introduction imposes its side condition on the assumptions a line rests on; Lemmon can state that exactly and Fitch can state it only about scope, and where the two diverge the translated proof is incorrect. We exhibit such a proof. The repair is local and does not move anything: the Lemmon side condition says exactly what is required to license a substitution, so the offending subderivation may be renamed to a name occurring nowhere else, whereupon the Fitch condition is satisfied too. We conjecture that the construction so modified is total, and identify what a proof of it would need. The construction is implemented, and its output machine-checked against a regression corpus and sixteen proofs photographed from student work.

1 Introduction

Anyone who has taught both notations knows that a Lemmon proof and a Fitch proof of the same argument look like transcriptions of one another. In the Lemmon style, each line is annotated with the numbers of the assumptions it depends on, and a rule such as conditional proof discharges an assumption by removing its number from the annotation. In the Fitch style, the same information is carried geometrically: an assumption opens a subproof, drawn as a vertical scope line, and discharging it closes the subproof. The two conventions are so

evidently doing the same work that the translation between them is generally left as an exercise, or not mentioned.

It is not quite an exercise. The purpose of this paper is to say exactly what the translation is, where it is easy, where it is not, and what the difficulty reveals about the relationship between the two notations.

The central observation is simple enough to state here. In a Lemmon proof, the dependency set attached to a line is *exact*: it is the set of assumptions the line actually rests on. In a Fitch proof, the corresponding information is the set of assumptions of the enclosing subproofs, and this is merely an *upper bound*. A Fitch line may perfectly well sit inside a subproof whose assumption it never uses. Every asymmetry we discuss below follows from this one.

A second observation, less obvious, does most of the work in Section 4. In a Lemmon proof, the textual position of a line carries no semantic content. Lines may be permuted freely, subject only to the requirement that a line be written after the lines it cites. In a Fitch proof, position is semantic: where a line sits determines what may see it, and a line inside a subproof becomes unavailable once that subproof closes. The translation from Lemmon to Fitch must therefore invent positional information that the source does not contain, and it is possible for the source to place demands that no such invention can satisfy.

A third observation is the paper's main negative result, and we did not anticipate it. Of the twenty-one rules, exactly one imposes a condition not on the formulas involved but on the assumptions a line rests on: universal introduction, whose eigenvariable must be arbitrary. Lemmon can state that condition exactly. Fitch cannot—it has no record of what a line rests on, only of what encloses it—and must state the condition about scope instead. Since scope may properly include the dependency set, the two conditions come apart, and a correct Lemmon proof may translate to a Fitch proof that breaks its own rule. Section 5 exhibits one.

We work throughout with the rules of *How Logic Works* [1], a Lemmon-style system with twenty-one rules, and with a Fitch system chosen to match it rule for rule. Section 2 fixes notation. Section 3 treats the easy direction and draws from it the redundancy of the dependency column. Section 4 exhibits the two obstructions to a positional translation in the other direction. Section 5 introduces the graph-theoretic vocabulary needed to say what the obstructions have in common, locates the difference between the notations in how each constrains the sharing of lines, gives the construction that translates by unfolding, and refutes it. Section 6 states what a proof of the repaired construction would require, and two questions we have not settled. Section 7 reports the machine verification, and Section 8 draws the moral.

2 The two systems

Definition 1. A *Lemmon proof* is a finite sequence of lines. A line is a quadruple $\langle \Gamma, n, \varphi, j \rangle$ where n is the line's number, φ its formula, j its justification, and Γ a set of line numbers—the *dependency set*. A justification is either A (assumption) or a rule together with the numbers of the lines it cites, all of which must be smaller than n.

The rules fall into two groups. Most of them—modus ponens, conjunction introduction, universal elimination—simply pool the dependency sets of the lines they cite. Four of them discharge: conditional proof and reductio each cite an assumption line together with a line derived from it, and disjunction and existential elimination each cite one or two such pairs. For these, the discharged assumption is removed from the pooled set. Thus in

1	(1)	P	A
2	(2)	Q	A
1,2	(3)	$P \wedge Q$	1,2 $\wedge$I
1	(4)	$Q \to (P \wedge Q)$	2,3 CP

line (4) discharges assumption (2), which accordingly leaves the dependency set.

In the implementation a line is a record and a justification an enumeration, one constructor per rule. The constructors of the discharging rules are worth noticing: each names an assumption line together with a line derived from it.

```
data ProofLine = ProofLine
  { lineNumber :: Int
  , formula :: PredFormula
  , justification :: Justification
  , references :: Set Int -- the dependency set
  }

data Justification
  = Assumption
  | MP Int Int -- modus ponens, two lines
  | AndIntro Int Int
  | CP Int Int -- assumption line, conclusion line
  | RAA Int Int
  | OrElim Int Int Int Int Int -- disjunction, then two such pairs
  | ExistsElim Int Int Int
  | ForallIntro Int
  -- ... fifteen further rules
```

Definition 2. A *Fitch proof* is a finite sequence of *items*, where an item is either a line—a number, a formula and a justification—or a *subproof*, consisting of an assumption line together with a further sequence of items. The *scope* of a line is the set of assumptions of the subproofs containing it. A line may cite another only if the cited line lies within its scope, that is, only if every subproof containing the cited line also contains the citing line.

The corresponding type is mutually recursive, an item being either a line or a subproof containing further items. A rule that discharges cites a subproof, which is written as the pair of its first and last line—the `SubRef` below—and this is exactly the pair that the Lemmon rule already names.

```
type SubRef = (Int, Int)

data FitchItem
  = FLine Int PredFormula FitchRule
  | FSub Subproof

data Subproof = Subproof
  { subAssumeLine :: Int
  , subAssumeForm :: PredFormula
  , subBody :: [FitchItem]
  }
```

```
data FitchRule
  = FPremise -- never discharged: outermost level
  | FAssume -- opens a subproof
  | FMP Int Int
  | FCP SubRef -- cites the subproof, not two lines
  | FOrE Int SubRef SubRef
  -- ... and so on
```

The same proof in Fitch:

1	P	Premise
2	$\mid\ Q$	Assume
3	$\mid\ P \wedge Q$	$\wedge$I 1,2
4	$Q \to (P \wedge Q)$	CP 2–3

Two conventions deserve comment. First, an assumption never discharged is written in Fitch as a premise at the outermost level, so the choice between "premise" and "assumption" is determined by whether the line is discharged later; nothing in the Lemmon proof marks the difference locally. Second, we take universal introduction to carry a side condition on the name generalised upon, rather than to open a subproof with a flagged constant. Both conventions are found in the literature; the first agrees with the condition already imposed in the Lemmon system, and choosing it lets universal introduction translate one line to one line.

3 From Fitch to Lemmon

This direction is unproblematic, and it is worth seeing why, because the reason is exactly what fails in the other direction.

Definition 3. Let F be a Fitch proof. Define $\delta(F)$, the Lemmon proof induced by F, by flattening F into a sequence of lines in order and assigning to each line n the set $\Gamma(n)$ given by: $\Gamma(n) = \{n\}$ if n is an assumption or premise; $\Gamma(n) = \bigcup_m \Gamma(m)$ for m ranging over the lines cited, if the rule at n discharges nothing; and for a discharging rule, the same union with the discharged assumptions removed.

The whole of δ's arithmetic is the following function, which computes one line's dependency set given those of the lines before it. The discharging cases subtract; every other rule takes the union of what it cites.

```
depsOf :: Map Int (Set Int) -> FitchRule -> Int -> Set Int
depsOf deps r self =
  case r of
    FPremise -> singleton self
    FAssume -> singleton self
    FCP (a, c) -> delete a (look c)
    FRAA (a, c) -> delete a (look c)
    FOrE d (a1,c1) (a2,c2) ->
      unions [ look d, delete a1 (look c1), delete a2 (look c2) ]
    FExistsE m (a, c) -> look m `union` delete a (look c)
```

```
  _ -> unions (map look (citedLines (toLemmonRule r)))
where
  look n = findWithDefault empty n deps
```

Theorem 4. *δ is total: for every Fitch proof F, $\delta(F)$ is a Lemmon proof, and F is correct if and only if $\delta(F)$ is.*

Proof sketch. The definition of Γ is a recursion on the position of the line, which terminates because a line may cite only earlier lines. That the resulting annotation satisfies the Lemmon conditions on each rule is checked rule by rule; the discharging rules are the only ones where anything happens, and for those the Fitch scope condition guarantees that the discharged assumption is available to be removed. □

Proposition 5. *For each line n of F, the set $\Gamma(n)$ assigned by δ is contained in the scope of n, and the containment may be proper.*

The containment may be proper precisely when a line sits inside a subproof whose assumption it does not use. This has a consequence worth recording.

Corollary 6. *δ is not injective. Two Fitch proofs that differ only in the placement of a line within subproofs it does not depend on have the same image.*

Consequently the round trip from Fitch through Lemmon and back is a normalisation rather than an identity: it returns a proof whose subproofs are no wider than they need to be.

The next observation is elementary but, we think, has not been stated. Because the Lemmon rules determine the dependency set of each line exactly—a system that permitted a line to cite more assumptions than it uses would not satisfy Proposition 5—the dependency column is not independent data.

Proposition 7. *Let L be a Lemmon proof and let L^- be the result of deleting its dependency column. Then L is recoverable from L^-.*

Proof. The recursion in the definition of δ makes no use of the dependency column of the source; it uses only the justifications. Applying it to L^- returns L. □

Remark 8. The pedagogical function of the dependency column is thus not to record information the reader could not otherwise obtain, but to require the student to compute it. This is a defensible reason for a notation to carry redundant data, and it is worth distinguishing from the reason a notation might carry data that is genuinely needed.

4 Two obstructions in the other direction

Definition 9. A translation from Lemmon to Fitch is *positional* if it preserves the sequence of lines: the nth line of the Lemmon proof becomes the nth line of the Fitch proof, with the same formula and the corresponding rule.

Positional translation is what one wants, when it is available. It preserves line numbers, so a student may read the two proofs side by side; and it certifies that the two proofs are the same proof in two notations, rather than two proofs of the same thing.

Theorem 10. *There are correct Lemmon proofs with no positional Fitch image.*

We give two, illustrating two independent obstructions. The implementation distinguishes them, and reports which was met and where:

```
data TranslationError
  = NotNested Int Int [Int] -- line, assumption discharged, boxes open
  | OutOfScope Int Int Int -- line, line cited, box that closed over it
  | PremiseInBox Int Int -- line, the box it was written inside
  -- ...
```

Example 11 (Discharge order)**.**

1	(1)	P	A
2	(2)	Q	A
1,2	(3)	$P \wedge Q$	1,2 $\wedge$I
2	(4)	$P \to (P \wedge Q)$	1,3 CP
	(5)	$Q \to (P \to (P \wedge Q))$	2,4 CP

Assumption (1) is made first and discharged first. In a positional Fitch image, the subproof opened by (1) would have to contain the subproof opened by (2), since (1) is opened first and subproofs are properly nested; but a subproof may be closed only when every subproof it contains has been closed, and (4) closes (1) while (2) remains open. Fitch permits discharge only in last-in-first-out order, and Lemmon imposes no such restriction.

Example 12 (Positional trapping)**.**

1	(1)	P	A
2	(2)	Q	A
1	(3)	$P \vee R$	1 $\vee$I
2	(4)	$Q \wedge Q$	2,2 $\wedge$I
	(5)	$Q \to (Q \wedge Q)$	2,4 CP
1	(6)	$(P \vee R) \wedge (Q \to (Q \wedge Q))$	3,5 $\wedge$I

Here the discharge order is unobjectionable. The difficulty is that line (3) is written between assumption (2) and its discharge at (5), and so in any positional image lies inside the subproof opened by (2). It does not depend on (2)—its dependency set is $\{1\}$—but that is of no help: when the subproof closes at (5), line (3) passes out of scope, and line (6) cites it. In the Lemmon proof nothing goes out of scope at all. Discharging an assumption changes the dependency set of the new line; it does not affect the availability of any earlier line.

Example 12 is the more instructive of the two, and it isolates the second of our opening observations. The Lemmon proof does not say that line (3) is "inside" anything. It is written where it is written, and that is a fact about the page, not about the derivation. A positional translation is obliged to read the page as though position meant something, and here it does not.

Remark 13. Both examples can be repaired by permuting lines. Swapping (1) and (2) in Example 11 yields a proof in which the discharges nest; moving (3) before (2) in Example 12 lifts it out of the subproof. Whether permutation always suffices is Conjecture 27 below.

5 Sharing, and two kinds of graph

Both obstructions concern the linear order of lines, and it is natural to ask what becomes of them when the linear order is set aside. Doing so requires a little graph-theoretic vocabulary,

which we give in full, since it is not universally familiar.

5.1 Definitions

Definition 14. A *directed graph* consists of a set of *nodes* together with a set of ordered pairs of nodes, the *edges*. We write $m \to n$ for an edge from m to n, and call m a *parent* of n. A *path* is a sequence of nodes each joined to the next by an edge; a *cycle* is a path returning to its starting node.

Definition 15. A *directed acyclic graph*, or *DAG*, is a directed graph with no cycles. A *rooted tree* is a DAG with a distinguished node, the *root*, such that every node other than the root has exactly one parent and every node is reachable from the root.

The difference that matters here is the one in the parent count. In a tree, each node hangs from exactly one place. In a DAG, a node may have several parents—and when it does, we say the node is *shared*: it does duty in more than one context at once, without being written down more than once.

Definition 16. Let G be a DAG with root r. Its *unfolding* is the tree whose nodes are the paths from r, with the obvious edges. Unfolding replaces each shared node by one copy per context in which it is used. The unfolding of a tree is itself; the unfolding of a DAG with genuine sharing is strictly larger, and may be exponentially so.

5.2 What each notation presents

The object that both notations record is a *derivation*: a tree whose root is the conclusion, whose internal nodes are applications of rules, and whose children are the premises those rules require. This is Gentzen's natural deduction in its original tree form, and neither Lemmon nor Fitch writes it down as a tree. Both write it as a DAG, because both write each line once and then refer back to it, which is precisely what sharing is.

Definition 17. The *citation graph* of a Lemmon proof has the lines as nodes, with an edge $m \to n$ whenever line n cites line m.

Proposition 18. *The citation graph of a Lemmon proof is a DAG, and every line is available to be cited by every later line.*

Proof. Acyclicity is immediate, since a line may cite only lines with smaller numbers. That every earlier line is available is a matter of the notation containing no device by which a line could cease to be. Discharging an assumption alters the dependency set recorded on a *new* line; it does nothing to any line already written. □

Fitch does have such a device.

Definition 19. The *scope tree* of a Fitch proof has the subproofs as nodes, together with a root representing the whole proof, and an edge from each subproof to those immediately contained in it. A line *belongs* to the innermost subproof containing it. A line m is *visible* to a later line n when the node m belongs to lies on the path from the root to the node n belongs to.

Proposition 20. *The scope tree of a Fitch proof is a rooted tree, and the citation graph of a Fitch proof is a DAG in which every edge $m \to n$ has m visible to n.*

We can now say exactly where the two notations part company, and it is not where the slogan "Lemmon is a graph, Fitch is a tree" would put it. Both present a derivation as a DAG; both share. The difference is that Fitch constrains its sharing and Lemmon does not. In a Lemmon proof any line may be shared with any later line whatever. In a Fitch proof a line may be shared only along the paths its scope tree permits, and once a subproof closes, everything belonging to it is permanently unavailable.

5.3 The translation, restated

The two obstructions of Section 4 can now be seen for what they are. By Proposition 20, translating a Lemmon proof to Fitch means finding a scope tree under which every citation the source makes is permitted. Both examples are cases where no scope tree does: in Example 11 because the discharges demand incompatible nestings, in Example 12 because a line is cited after the only subproof it could have belonged to has closed.

When no scope tree suffices, one may unfold. Unfolding removes sharing, and a DAG without sharing is a tree; a tree imposes no constraint that could be violated, since nothing is shared to be shared wrongly.

A derivation is represented directly as a tree. Where the Lemmon justification names line numbers, the derivation names subderivations, and a line cited twice becomes two subtrees.

```
data Deriv = Deriv { dForm :: PredFormula, dRule :: DRule }

data DRule
  = DAssume Int -- discharged by an ancestor
  | DPremise Int -- never discharged
  | DMP Deriv Deriv
  | DCP Int PredFormula Deriv -- discharges the named assumption
  | DOrE Deriv Int PredFormula Deriv Int PredFormula Deriv
  -- ...
```

Translation is then two functions and a choice between them. The positional translation is attempted first, because it preserves numbering and structure; unfolding is the fallback, and the result records which was used.

```
data Route = Direct | ViaTree

lemmonToFitch :: Proof -> Either TranslationError (Route, FitchProof)
lemmonToFitch prf =
  case lemmonToFitchDirect prf of
    Right fp -> Right (Direct, fp)
    Left _ -> do d <- toDerivation prf
                 pure (ViaTree, derivationToFitch d)
```

Definition 21 (The unfolding translation)**.** Given a correct Lemmon proof, unfold its citation graph into a derivation rooted at the last line. Then traverse the derivation, emitting first the premises, at the outermost level and once each however often they are used, and thereafter,

for each node, the subderivations its rule requires followed by the line applying it, opening a subproof at each discharge.

Theorem 22. *The construction of Definition 21 does not always yield a correct Fitch proof.*

Proof. Consider

1	(1) $\forall x\, G(x)$	A
2	(2) $F(a)$	A
1	(3) $G(a)$	1 ∀E
1	(4) $\forall y\, G(y)$	3 ∀I
1,2	(5) $F(a) \wedge \forall y\, G(y)$	2,4 ∧I
1	(6) $F(a) \to (F(a) \wedge \forall y\, G(y))$	2,5 CP

which is correct. Line (4) generalises on a, and $\Gamma(4) = \{1\}$; the sole assumption in that set has formula $\forall x\, G(x)$, in which a does not occur. The Lemmon side condition is satisfied.

The construction returns

1	$\forall x\, G(x)$	Premise
2	$\vert\ F(a)$	Assume
3	$\vert\ G(a)$	∀E 1
4	$\vert\ \forall y\, G(y)$	∀I 3
5	$\vert\ F(a) \wedge \forall y\, G(y)$	∧I 2,4
6	$F(a) \to (F(a) \wedge \forall y\, G(y))$	CP 2–5

in which line 4 applies universal introduction on a while $F(a)$ stands as an undischarged assumption in its scope. That is contrary to the side condition, so the proof is not correct. □

It is worth being clear that this is not an artefact of the Fitch system we chose in Section 2. One might think to repair matters by restating Fitch's side condition against the assumptions a line *depends on* rather than those in scope. But a Fitch line does not record what it depends on; that is precisely the difference between the two notations, and the reason the dependency column of a Lemmon proof, though redundant in the sense of Proposition 7, is not idle. Fitch must state its eigenvariable condition against scope, because scope is the only thing it has. The failure is therefore a genuine one: at this rule, Fitch's coarser bookkeeping is not merely less informative but licenses strictly less.

Nothing is lost in the way of provability. The sequent above is Fitch-provable by other means, and the two systems remain equivalent in the sense of proving the same sequents. What fails is the structure-preserving translation.

There are two ways to repair it. One is to move the offending line: emit each line at the depth of the innermost assumption in its dependency set, so that scope and dependency coincide, and the two side conditions with them. That would work, but it requires the subproofs so induced to nest properly, which is Conjecture 28 below and is not known.

The other is to move nothing and rename instead. It is available because the Lemmon side condition says precisely what is needed to license a substitution.

Lemma 23 (Renaming)**.** *Let D be a derivation of $\Gamma \vdash \psi$, and let b be a name occurring nowhere in D. Then $D[b/a]$ is a derivation of $\Gamma[b/a] \vdash \psi[b/a]$.*

Corollary 24. *If in addition a does not occur in Γ, then $\Gamma[b/a] = \Gamma$, and so $\Gamma \vdash \psi[b/a]$.*

Lemma 23 is an instance of the substitution theorem of [2, Thm. 4.5.14]: for a reconstrual $*$ with admissibility conditions Δ for its function symbols, if $\phi \vdash \psi$ then $\Delta, \phi^* \vdash \psi^*$. Renaming a to b is the reconstrual sending $a = y$ to $b = y$, whose admissibility condition $\exists! y\,(b = y)$ is a theorem and so discharges, leaving $\phi^* \vdash \psi^*$.

That proof does include the case for universal introduction. It is given, however, for a system in which generalisation is on a free variable, the side condition being that x is not free in Γ; the step goes through because reconstrual preserves free variables, so that x is not free in Γ^* either.

The system of [1] generalises on a name rather than a variable, and there the step is not free. Substituting one name for another *can* introduce into Γ the very name a universal introduction is required to avoid—which is precisely what the free-variable lemma rules out in the other formulation. The condition that b be fresh is what closes that gap, and since it is the step at issue here we give it.

Universal introduction. Suppose $\Gamma \vdash \forall x\,\psi(x)$ is derived by $\forall$I from $\Gamma \vdash \psi(c)$, where by the side condition c occurs neither in Γ nor in $\psi(x)$, and suppose the result holds for the latter.

If $c \neq a$, then $\psi(c)[b/a] = (\psi[b/a])(c)$, and the inductive hypothesis gives $\Gamma[b/a] \vdash (\psi[b/a])(c)$. Universal introduction on c applies: c does not occur in $\Gamma[b/a]$, since it did not occur in Γ and the substitution introduces only b, which differs from c because b occurs nowhere in D while c does; and c does not occur in $\psi(x)[b/a]$ for the same reason. So $\Gamma[b/a] \vdash \forall x\,(\psi[b/a])(x)$, which is $(\forall x\,\psi(x))[b/a]$.

If $c = a$, then a occurs neither in Γ nor in $\psi(x)$, so $\Gamma[b/a] = \Gamma$ and $\psi(a)[b/a] = \psi(b)$. The inductive hypothesis gives $\Gamma \vdash \psi(b)$. Universal introduction on b applies: b occurs neither in Γ nor in $\psi(x)$, since it occurs nowhere in D at all. So $\Gamma \vdash \forall x\,\psi(x)$, and since a does not occur in $\forall x\,\psi(x)$, this is $(\forall x\,\psi(x))[b/a]$. □

Existential elimination goes the same way, its witness condition being of the same shape; the remaining rules impose no condition on names and follow the pattern of the propositional cases.

Remark 25. The freshness of b is not a convenience, and the contrast with the free-variable formulation shows why. There, reconstrual preserves free variables, so a universal introduction licensed before the substitution is licensed after it. Here, substitution of one name for another can *add* a name to Γ: if b already occurs in Γ, a universal introduction on b somewhere within D may be licensed before the substitution and not after, since the substitution places b among the assumptions that introduction is required to avoid. It is the second case of the proof above that this breaks.

The repair follows, by Corollary 24. Wherever the construction is about to emit a universal introduction whose eigenvariable occurs in an assumption in scope, rename the subderivation feeding it to a name occurring nowhere in the proof—nowhere, and not merely nowhere in the subderivation, for the reason given in Remark 25. Its open assumptions are untouched, since by the Lemmon side condition a does not occur in them. Its conclusion is untouched, since abstraction removes every occurrence of a. And the new name occurs in no assumption whatever, so *a fortiori* in none in scope. Note that the unfolding shares nothing, so renaming within one subderivation cannot disturb any other.

Applied to the proof of Theorem 22, a single line changes:

1	$\forall x\, G(x)$	Premise
2	$\mid\ F(a)$	Assume
3	$\mid\ G(b)$	$\forall$E 1
4	$\mid\ \forall y\, G(y)$	$\forall$I 3
5	$\mid\ F(a) \wedge \forall y\, G(y)$	$\wedge$I 2,4
6	$F(a) \to (F(a) \wedge \forall y\, G(y))$	CP 2–5

Line 3 instantiates to b rather than a, and line 4 generalises on a name that occurs in no assumption anywhere. Nothing else moves.

Conjecture 26. *Let the construction of Definition 21 be modified as above, renaming the eigenvariable of a universal introduction whenever it occurs in an assumption in scope. So modified, it yields for every correct Lemmon proof a well-formed and correct Fitch proof with the same conclusion.*

Two things recommend this repair over the first. It is local: no line moves, no subproof changes shape, and the translation remains positional up to the renaming of constants. And it is independent of Conjecture 28, since it raises no question about where lines are placed. A proof of Conjecture 26 therefore requires the four lemmas of Section 6 together with Lemma 23, and nothing else that is currently open.

The construction has a cost, which is the cost of unfolding: a line shared k times is derived k times, and in the worst case the result is exponential in the size of the source. There is also a small point of bookkeeping. If a subproof's conclusion is a line from outside it—as in deriving $Q \to P$ from the premise P—then the subproof would contain nothing at all, and Fitch requires a reiteration step to bring the line in. Lemmon has no such rule and needs none, by Proposition 18; and the reiterated line is a tautological consequence of the line it repeats, so nothing is added to the system.

6 What remains to be proved

6.1 Towards the totality conjecture

A proof of Conjecture 26 requires four lemmas about the construction of Definition 21. The first three are routine; the fourth is not, for one rule.

(L1) *Unfolding terminates.* A line may cite only lines with smaller numbers, so the recursion is well-founded on the line number, and the derivation is finite.

(L2) *Citations precede uses.* By induction on the derivation: the traversal emits every subderivation a node requires before the line applying it, so every line number a rule cites is smaller than the number of the line citing it.

(L3) *Every citation is in scope.* By induction, using two facts about the traversal: premises are emitted at the outermost level before anything else, and an assumption is emitted as the first line of its own subproof, so that uses of it within that subproof are visible to it. This is the lemma that the implementation initially failed—it emitted premises wherever the source had written them, which put some inside subproofs—and it is therefore the one to state most carefully.

(L4) *Each rule transfers.* For each of the twenty-one rules: if the Lemmon rule licenses φ from the lines it cites, the corresponding Fitch rule licenses φ from the corresponding lines and subproofs. Eighteen of these are immediate, the rule being the same rule with a different citation convention. Conditional proof, reductio, disjunction elimination and existential elimination require checking that the subproof emitted really does run from the assumption to the conclusion the Lemmon rule named. Universal introduction is where the unmodified construction fails (Theorem 22); for the modified construction the case reduces to Lemma 23, which itself wants a proof for this rule set.

6.2 Why universal introduction, and no other rule

Theorem 22 turned on universal introduction, and it is worth recording why that rule and no other.

Every other rule in the system is a condition on the formulas cited and the formula concluded. Universal introduction alone imposes a condition on the *assumptions*: $\forall x\varphi(x)$ may be inferred from $\varphi(a)$ only if a is arbitrary, and arbitrariness is a matter of what the line rests on. Lemmon can say this exactly, because a Lemmon line records exactly what it rests on. Fitch can only say it about scope. Wherever scope properly includes the dependency set—which by Proposition 5 it may—the two conditions come apart, and Theorem 22 exhibits a proof where they do.

That the repair is a renaming rather than a rearrangement is a point in favour of the diagnosis. The difficulty is not that the line is in the wrong place; it is that the *name* is doing double duty, standing both as the arbitrary individual of the generalisation and as the individual the enclosing assumption is about. Lemmon can tell those two roles apart because it knows which assumptions the line answers to. Fitch cannot, and so a name that is arbitrary in the one sense is not visibly arbitrary in the other. Giving the generalisation a name of its own removes the ambiguity without touching the structure of the proof at all.

6.3 Two further questions

Conjecture 27. *Every correct Lemmon proof can be permuted—preserving the requirement that a line follow the lines it cites—into one with a positional Fitch image.*

Conjecture 28. *No translation requires duplication. That is, every correct Lemmon proof has a Fitch image in which each line of the source occurs exactly once.*

Conjecture 28 is suggested by the following consideration. If line L is cited by line M under a non-discharging rule, then $\Gamma(L) \subseteq \Gamma(M)$. Placing every line at the depth of the innermost assumption in its dependency set therefore places L no deeper than M, and hence within M's scope. What this argument does not establish is that the resulting family of subproofs is laminar, which is what a Fitch proof requires, and we have not been able to settle whether it must be.

Neither conjecture is now load-bearing. An earlier version of this paper repaired Theorem 22 by placing lines at the depth of their dependencies, which made Conjecture 26 depend on Conjecture 28; the renaming repair does not, and the two questions below are independent curiosities about how economical a translation can be, rather than obstacles to its existing.

Some evidence bears on both. Examples 11 and 12 translate through the construction to Fitch proofs of five and six lines respectively—exactly the lengths of the originals. No line

was duplicated in either case. This is what one would expect if Conjecture 28 holds and the unfolding is simply doing more work than it needs to.

7 Machine verification

The translations in both directions were implemented in Haskell against the proof checker for *How Logic Works*. What follows is evidence for Conjecture 26, not a substitute for a proof of it: every item below is a statement about particular proofs, and the conjecture is a statement about all of them.

Every proof in a regression corpus of 78 cases was translated, where valid, from Lemmon to Fitch and back. Twenty-five admitted a positional translation, and for each the round trip returned the original proof line for line, *including its dependency sets*—an instance of Proposition 7, since the sets were recomputed and not copied. The two proofs of Section 4 were translated through derivation trees; the results were submitted to the proof checker and found correct, with the same conclusions.

Sixteen further proofs, transcribed from photographs of student work, were translated. Fourteen were well formed; all fourteen admitted positional translations and round-tripped exactly. The remaining two contained malformed justifications on the page—a rule cited with the wrong number of lines—and were rejected by the parser, as they should be.

We record the empirical finding for what it is worth: neither obstruction occurred in any student proof. They are not artefacts, as Section 4 shows, but they appear to be rare in practice.

One methodological point deserves recording, because it cost us two false negatives. The round trip is a weaker test than it appears. Since δ recomputes dependency sets from the rules rather than reading them off the page, a Fitch proof that is *malformed*—one violating the conditions in Definition 2—may still return the Lemmon proof it came from, exactly. Two proofs in the corpus did precisely this, both by placing a premise inside a subproof, where its dependency set recomputes correctly while the subproof appears to derive the premise from its assumption. Neither was detected by the round trip; the first was noticed by looking at the rendered output. The remedy is to check the conditions of the notation directly, and separately:

```
-- Conditions the notation imposes, which a round trip cannot detect:
-- * a premise may occur only at the outermost level;
-- * a line may cite only what is in scope.
fitchWellFormed :: FitchProof -> Maybe String
```

The moral generalises beyond this case. A test that verifies an invariant is preserved under translation cannot, by itself, establish that either side satisfies the conditions of its own notation.

8 What kind of equivalence is this?

It is tempting to summarise all of this by saying that Lemmon and Fitch notation are intertranslatable, and to leave it there. The results above suggest that the summary conceals more than it reports.

Translation from Fitch to Lemmon is total, canonical, and not injective (Theorem 4, Corollary 6). Translation from Lemmon to Fitch is not positional (Theorem 10), though we conjecture it is total by way of the unfolding (Conjecture 26), the unmodified construction having been refuted by Theorem 22 and repaired by renaming. Composing in one order returns the original proof; composing in the other returns a normal form. In the vocabulary of structural relations between formalisms, Lemmon proofs are a *retract* of Fitch proofs, not an isomorphic copy of them.

The reason is that the two notations record different amounts. A Lemmon proof carries its dependency structure exactly, and carries no positional information at all. A Fitch proof carries its dependency structure only up to an upper bound, and carries positional information in addition—information which is, from the point of view of the derivation, arbitrary. Neither is a richer notation than the other. They are differently lossy.

That is a more interesting relationship than notational variance, and it is the sort of relationship that arises whenever two formalisms are said to say the same thing. The case is unusually clean: both formalisms are small, completely specified, and mechanically checkable, so the question of what survives translation can be settled rather than argued. We suggest it is worth having such a case in view when the same question is asked of theories, where it cannot.

References

[1] Hans Halvorson. *How Logic Works: A User's Guide*. Princeton University Press, 2020.

[2] Hans Halvorson. *The Logic in Philosophy of Science*. Cambridge University Press, 2019.

[3] E. J. Lemmon. *Beginning Logic*. Nelson, 1965.

[4] Frederic B. Fitch. *Symbolic Logic: An Introduction*. Ronald Press, 1952.

[5] Patrick Suppes. *Introduction to Logic*. Van Nostrand, 1957.