Simplifying proofs in Fitch-style natural deduction systems

Konstantine Arkoudas
MIT Computer Science and AI Lab

November 27, 2005

Abstract

We present an algorithm for simplifying Fitch-style natural-deduction proofs in classical first-order logic. We formalize Fitch-style natural deduction as a denotational proof language, $\mathcal{NDL}$, with a rigorous syntax and semantics. Based on that formalization, we define an array of simplifying transformations and show them to be terminating and to respect the formal semantics of the language. We also show that the transformations never increase the size or complexity of a deduction—in the worst case, they produce deductions of the same size and complexity as the original. We present several examples of proofs containing various types of superfluous "detours," and explain how our procedure eliminates them, resulting in smaller and cleaner deductions. All of the transformations are fully implemented in SML-NJ, and the complete code listing is available on the Web.

1.1 Introduction

This paper is concerned with the problem of simplifying proofs in Fitch-style natural-deduction systems. The hallmark of such systems is the idea of "making arbitrary assumptions and keeping track of where they lead and for how long the assumptions are in effect" [27]. More briefly, we might say that the cornerstone of such systems is the notion of *conditional subproof*: if at some point in a proof D we wish to establish a conditional $F \Rightarrow G$, we postulate F as a provisional hypothesis and proceed to give a subproof D' that derives G. The subproof D' is free to use F along with whatever assumptions and previously obtained conclusions are available up to that point. D' is written directly underneath and to the right of the hypothesis F. The indentation serves to delineate D' as the *scope* of F. This is usually emphasized graphically by enclosing D' inside a square box or by drawing a vertical line extending from F to the end of D'.

This style of deduction was pioneered by the Polish logician Jáskowski in the early 1930s, not by Fitch. But Fitch streamlined Jáskowski's method, and it is now standard practice in the literature to speak of "Fitch systems." Such systems are the most popular pedagogical choice for teaching symbolic logic, used by numerous influential logic textbooks [22, 32, 15, 11, 8, 23, 9]. They are considered to be the most natural of the three main families of proof systems that claim to capture the way in which mathematicians present proofs in practice, the other two being the natural deduction trees deriving from Gentzen's N calculus[1] and sequent-based systems originating in Gentzen's L calculus.[2] Proof readability and writability are compromised both in tree-based and in sequent-based systems, and it is questionable[3] to what extent such systems can be said to reflect ordinary mathematical reasoning.

As mentioned above, presentations of Fitch-style natural deduction often rely on graphical devices such as boxes and lines to demarcate assumption scope. It is remarkable how little this has changed since the introduction of the method by Jáskowski in 1934; even the most recent textbooks continue to use the same lines-and-boxes approach that was used 70 years ago. We have recently given an alternative formalization of Fitch-style natural deduction [3] in the form of a denotational proof language, $\mathcal{NDL}$, that draws on contemporary programming language theory. $\mathcal{NDL}$ proofs are succinctly specified by an abstract grammar ([33]), while a big-step operational semantics [21, 29] attaches a rigorous meaning to every proof that is syntactically well-formed. Assumption scope is captured by context-free block structure, obviating the need for boxes or lines.[4]

[1]Also used by Prawitz [31] in his "Natural Deduction" [31], by Van Dalen in his "Logic and structure" [12], and by Troelstra and Schwichtenberg in "Basic Proof Theory" [34].

[2]Used in books such as "Mathematical Logic" by Ebbinghaus et al. [14] and in theorem-proving systems such as HOL [18] and Isabelle [26].

[3]E.g., Pelletier [27] states that sequent-based calculi are simply not natural deduction systems.

[4]The idea of representing assumption scope by block structure is also present in formalizations of intuitionist natural deduction in higher-order type theory [19, 20, 17] based on the λ-calculus, but there are important differences; see [3]

Semantically, the formal meaning of a $\mathcal{NDL}$ proof is specified relative to a given *assumption base*, which is a set of premises—a set of propositions that we take for granted for the purposes of a given stretch of logical discourse. If a DPL proof is sound with respect to a given assumption base, then its meaning (denotation) is the conclusion established by the proof; if the proof is unsound, then its meaning is an error token. To obtain this meaning, we *evaluate* the proof in accordance with the formal semantics of the language. Evaluation will either produce the said conclusion, which will verify that the proof is sound, or else it will generate an error, which will indicate that the proof is unsound. Therefore, evaluation becomes tantamount to proof checking.

In addition to clarity and ease of presentation, defining Fitch-style natural deduction in this manner has two further advantages. First, standard programming language implementation techniques such as parsing and interpretation become available for the purpose of mechanizing proofs. Second, a formal semantics allows us to develop a rigorous theory of observational equivalence for proofs, providing precise answers to questions such as: What does it mean for two proofs to be equivalent? When can one proof be substituted for another, i.e., under what conditions can one proof be "plugged in" inside another proof without changing the latter's meaning? When can one proof be considered more efficient than another? What kinds of optimizations can be performed on proofs? When is it safe to carry out such optimizations? And so on. This point will be of key importance in our development. In the absence of a formal abstract syntax and semantics, most of the transformations we define in this paper would be inordinately difficult to even state, let alone to prove correct.

Apart from the intrinsic theoretical interest of the subject, there are several practical motivations for this work. Automated theorem proving systems based on Fitch-style natural deduction, such as Oscar [30] and Thinker [28], often output long proofs with many redundancies. The algorithms we describe here could be easily implemented in these systems to clean up the proofs.[5] Likewise, method applications in Athena [1] and other type-ω DPLs [4] that perform proof search in a natural deduction setting are likely to produce suboptimal proofs with various detours; our procedures should prove useful there as well. In addition, Athena employs resolution-based systems such as Vampire [35] and Spass [37] for proof search [5] and their output proofs also contain many redundancies. Those resolution proofs could be converted to Fitch-style native Athena proofs and simplified with the procedures we describe here. Another potential area of application is proof-carrying code (PCC [24]), where proof size is an important practical consideration [25]. Finally, our algorithms could prove useful for educational purposes. Beginning logic students often write proofs in an immature style, deriving extraneous conclusions, placing an inference in the scope of a hypothesis on which it does not depend, etc. If students developed proofs on the computer (e.g., in $\mathcal{NDL}$ form or in a Fitch system such as Hyperproof [7]), it would be possible to immediately simplify their deductions and display the results to them.

1.2 Background

Our subject is related to proof-tree normalization in the sense of Prawitz [31] (or alternatively, cut-elimination in sequent-based systems [16, 13]). In the intuitionist case, the Curry-Howard correspondence means that Prawitz normalization coincides with reduction in the simply typed λ-calculus. Accordingly, the normalization algorithm in that case is particularly simple: keep contracting as long as there is a redex. Strong normalization and the Church-Rosser property guarantee that eventually

for a discussion of such differences.

[5]Thinker already has a "post-processor" that eliminates some redundant claims, but the procedures we describe here are much more aggressive.

we will converge to a unique normal form. In the classical case, there is some pre-processing to be done (see Section I, Chapter III of Prawitz's book [31]) before carrying out reductions.

Fitch-style systems present complications of a different combinatorial nature. One important difference is that in Fitch systems inference rules are applied to propositions rather than to entire proofs. If $\mathcal{NDL}$ were based on a proof-tree model, where inference rules are applied to proofs, we could then readily formulate local contraction rules in the style of Prawitz, such as

$$\begin{gathered}\textbf{right-and}(\textbf{both}(D_1, D_2)) \longrightarrow D_2 \\ \textbf{modus-ponens}(\textbf{assume } P \textbf{ in } D_1, D_2) \longrightarrow D_1[D_2/P]\end{gathered}$$

and so on. But in $\mathcal{NDL}$ there is not much we can infer from looking at an individual application of an inference rule (such as **left-iff** $P \Leftrightarrow Q$), so global analyses are needed to identify and eliminate detours. Essentially, because assumptions and intermediate conclusions can have limited and arbitrarily nested scopes, it is generally not possible to carry out reductions in a local manner; the overall surrounding context must usually be taken into consideration. Further, the result of one transformation might affect the applicability or outcome of another transformation, so the order in which these occur is important.

Our simplification procedure will consist of a series of transformations, which fall into two groups:

- *restructuring transformations*; and
- *contracting transformations*, or simply *contractions*.

Contracting transformations form the bedrock of the simplification process: they remove extraneous parts, thereby reducing the size and complexity of a deduction. Restructuring transformations simply rearrange the structure of a deduction so as to better expose simplification opportunities; they constitute a kind of pre-processing aimed at facilitating the contracting transformations.

Specifically, our top-level simplification procedure is defined as follows:

$$\mathit{simplify} = \mathit{contract} \cdot \mathit{restructure} \tag{1.1}$$

where $\cdot$ denotes ordinary function composition and

$$\begin{aligned} \mathit{contract} &= \mathit{fp}\ \ (\mathfrak{C} \cdot \mathfrak{P} \cdot \mathfrak{U}) &(1.2)\\ \mathit{restructure} &= \mathit{reduce}\ \ (\lambda\, f, g \,.\, \mathfrak{MS} \cdot f \cdot g)\ \ \mathfrak{MS}\ \ [\mathfrak{A}_3, \mathfrak{A}_2, \mathfrak{A}_1]. &(1.3)\end{aligned}$$

The fixed-point-finder function *fp* is defined as:

$$\begin{aligned} &\mathit{fp}\ \ f = \lambda\, D \,.\, \mathit{let}\ \ D' = f\ \ D \\ &\qquad\qquad\quad \mathit{in} \\ &\qquad\qquad\qquad D \equiv_\epsilon D' \,? \rightarrow D \diamondsuit \mathit{fp}\ \ f\ \ D' \end{aligned}$$

where *reduce* is the usual list-reducing functional and $D \equiv_\epsilon D'$ signifies that D and D' are identical (or, more precisely, that they differ only in the names of their "eigenvariables"; this is rigorously defined in Section 1.3.1). An equivalent definition of *restructure* is as follows:

$$\mathit{restructure} = \mathit{weave}\ \ \mathfrak{MS}\ \ [\mathfrak{A}_3, \mathfrak{A}_2, \mathfrak{A}_1] \tag{1.4}$$

with the weaving function defined thus:

```
$weave\ f\ L = let\ T\ [\,] = f$
                  $T\ g{::}L' = f \cdot g \cdot (T\ L')$
              $in$
                $T\ L$
```

We will continue to define functions in this informal notation, using pattern matching, recursion, etc., in the style of (strict) higher-order functional languages such as ML. Any reader moderately familiar with a programming language of that kind should be able to make sense of our definitions.[6] As a convention, we write $E\,?\rightarrow E_1 \diamondsuit E_2$ to mean "if E then E_1, else E_2." Also, we write $[x_1,\ldots,x_n]$ for the list of $x_1,\ldots,x_n$, $n \geq 0$, and $x{::}L$ for the list obtained by prepending ("consing") x in front of L. Finally, we use the symbol $\oplus$ for list concatenation.

We will show that our simplification procedure has three important properties: it always terminates; it is safe; and it never increases the size or complexity of a deduction. Specifically, the following will hold for all deductions D:

1. The computation of *simplify*(D) terminates.
2. *simplify*(D) respects the semantics of D, in a sense that will be made rigorous in Section 1.3.
3. The size of *simplify*(D) is less than or equal to the size of D.

This last point puts our work in marked contrast to cut elimination (or normalization) algorithms. Eliminating cuts from a proof will not necessarily result in a smaller or simpler proof. In fact it may result in a dramatically (e.g., exponentially) larger proof, even when the original proof is fairly short and simple. (This has led some logicians to caution against cut elimination [10].) By contrast, the result of our simplification procedure will never be larger than the original, and will indeed often be smaller and simpler.

The remainder of this paper is structured as follows. The next section briefly reviews the syntax and semantics of $\mathcal{NDL}$, along with some basic notions and results that will form the theoretical background for our transformations. Omitted proofs can be found in Chapter 6 of [2]. The following two sections discuss each group of transformations in turn: first the contractions $\mathfrak{C}$, $\mathfrak{P}$, and $\mathfrak{U}$; and then the restructuring transformations $\mathfrak{MS}$, $\mathfrak{A}_1$, $\mathfrak{A}_2$, and $\mathfrak{A}_3$. Finally,in Section 1.6 we give a number of examples illustrating the various transformations in action; the examples demonstrate that *simplify* can often result in substantial size reductions.

1.3 $\mathcal{NDL}$

1.3.1 Syntax

We assume we have a fixed *signature* consisting of a set of *constant symbols*, a set of *function symbols*, and a set of *relation symbols*. These three sets are required to be pairwise disjoint. Every function and relation symbol has a unique positive integer associated with it and known as its *arity*. We also assume the existence of a set of *variables*, disjoint from the three sets of symbols. We use the letters a, b, and c as typical constant symbols; f, g, and h as function symbols; M, P, Q, and R as relation symbols; and x, y, z, u, v, and w as variables. Symbols such as f^n (R^n) will range over function (relation) symbols of arity n. *Terms* are defined as usual: a term is either a constant symbol, or a

[6]All of the algorithms presented in this paper have been implemented in SML-NJ. The code is available from `www.cag.csail.mit.edu/~kostas/dpls/ndl`.

variable, or an "application" of the form $f^n(t_1, \ldots, t_n)$ for $n > 0$ terms $t_1, \ldots, t_n$. We will use the letters s and t to designate terms.

The formulas of $\mathcal{NDL}$ have the following abstract syntax:

$$F ::= \mathbf{true} \mid \mathbf{false} \mid R^n(t_1, \ldots, t_n) \mid \neg F_1 \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \Rightarrow F_2 \mid F_1 \Leftrightarrow F_2 \mid \forall\, x \,.\, F \mid \exists\, x \,.\, F$$

The letters F, G, H, I and J are used to denote formulas. Parsing ambiguities will be resolved by parentheses and brackets. By an *assumption base* β we will mean a finite set of formulas.

Free and bound variable occurrences in formulas are defined as usual. We write $FV(F)$ for the set of those variables that have free occurences in F. Formulas that differ only in the names of their bound variables are called *alphabetically equivalent* and will be identified. That is, we consider two formulas to be identical iff each can be obtained from the other by consistently renaming its bound variables (see [2] for a rigorous definition). For an assumption base β, $FV(\beta)$ will denote the set of all and only those variables that occur free in some element of β. We define a *substitution* as any function θ mapping variables to terms that is the identity on all but finitely many variables; that is, $\theta(x) \neq x$ only for finitely many x. The finite set comprised by these variables is called the *support* of θ, $Supp(\theta)$. Since a substitution θ is completely determined by its restriction to its support, it is customary to identify it with the finite set $\{\langle x_1, \theta(x_1)\rangle, \ldots, \langle x_k, \theta(x_k)\rangle\}$, where $\{x_1, \ldots, x_k\} = Supp(\theta)$. The more suggestive notation $\{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$ is used to represent the substitution that maps each x_i to t_i and every other variable to itself. We write $\theta[x \mapsto t]$ for the substitution that maps x to t and every other variable x' to $\theta(x')$, and define $RanVar(\theta)$ as the set of all and only those variables that occur in some term $\theta(x)$, for $x \in Supp(\theta)$. We say that two substitutions θ_1 and θ_2 are *disjoint* iff $Supp(\theta_1) \cap Supp(\theta_2) = \emptyset$; and $RanVar(\theta_i) \cap Supp(\theta_j) = \emptyset$ whenever $i, j \in \{1, 2\}, i \neq j$.

Any substitution θ can be extended to a homomorphism $\widehat{\theta}$ from terms to terms in the usual manner [36]. Since the extension is unique, we may simply write $\theta(t)$ instead of $\widehat{\theta}(t)$. Substitutions can also be applied to formulas. Further overloading our notation, we define $\theta(F)$, the result of applying a substitution θ to a formula F, as follows:

$$\begin{array}{rcl}
\theta(R(t_1, \ldots, t_n)) & = & R(\theta(t_1), \ldots, \theta(t_n)) \\
\theta(\neg F) & = & \neg\, \theta(F) \\
\theta(F \circ G) & = & \theta(F) \circ \theta(G) \\
\theta(\forall\, x \,.\, F) & = & \forall\, x \;.\; \theta[x \mapsto x](F) \\
\theta(\exists\, x \,.\, F) & = & \exists\, x \;.\; \theta[x \mapsto x](F)
\end{array}$$

for $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$. For a set of formulas Φ, we write $\theta(\Phi)$ to denote $\{\theta(F) \mid F \in \Phi\}$. To minimize notational clutter, we often write $\theta\, t$ (or $\theta\, F$, or $\theta\, \Phi$) instead of $\theta(t)$ (respectively, $\theta(F)$ or $\theta(\Phi)$).

The proofs (or "deductions") of $\mathcal{NDL}$ have the following abstract syntax:

$$\begin{array}{rlll}
D & = & \mathit{Prim\text{-}Rule}\ F_1, \ldots, F_n & \text{(Primitive rule applications)} \\
 & \mid & \mathbf{assume}\ F\ D & \text{(Conditional deductions)} \\
 & \mid & \mathbf{suppose\text{-}absurd}\ F\ D & \text{(Proofs by contradiction)} \\
 & \mid & D_1; D_2 & \text{(Compositions)} \\
 & \mid & \mathbf{pick\text{-}any}\ x\ D & \text{(Universal generalizations)} \\
 & \mid & \mathbf{specialize}\ \forall\, x \,.\, F\ \ \mathbf{with}\ t & \text{(Universal instantiations)} \\
 & \mid & \mathbf{ex\text{-}generalize}\ \exists\, x \,.\, F\ \ \mathbf{from}\ t & \text{(Existential generalizations)} \\
 & \mid & \mathbf{pick\text{-}witness}\ w\ \mathbf{for}\ \exists\, x \,.\, F\ D & \text{(Existential instantiations)}
\end{array}$$

where:

$$\begin{array}{lcl} \textit{Prim-Rule} & ::= & \textbf{claim} \mid \textbf{modus-ponens} \mid \textbf{true-intro} \mid \textbf{both} \mid \textbf{left-and} \\ & \mid & \textbf{right-and} \mid \textbf{double-negation} \mid \textbf{cases} \mid \textbf{left-either} \\ & \mid & \textbf{right-either} \mid \textbf{equivalence} \mid \textbf{left-iff} \mid \textbf{right-iff} \mid \textbf{absurd} \end{array}$$

Deductions of the form *Prim-Rule* $F_1, \ldots, F_n$ are called *primitive rule applications*; those of the form **assume** $F\ \ D$ and $D_1; D_2$ are *conditional* and *composite* deductions, respectively; and those of the form **suppose-absurd** $F\ \ D$ are proofs *by contradiction.* Primitive rule applications as well as universal instantiations and existential generalizations are *atomic* deductions, as they have no recursive structure, whereas all other forms are *compound* or *complex.* This distinction is reflected in the definition of $SZ(D)$, the *size* of a given D:

$$\begin{array}{rcl} SZ(\textit{Prim-Rule}\ F_1, \ldots, F_n) & = & 1 \\ SZ(\textbf{assume}\ F\ \ D) & = & 1 + SZ(D) \\ SZ(\textbf{suppose-absurd}\ F\ \ D) & = & 1 + SZ(D) \\ SZ(D_1; D_2) & = & SZ(D_1) + SZ(D_2) \\ SZ(\textbf{specialize}\ \forall\, x\, .\, F\ \ \textbf{with}\ \ t) & = & 1 \\ SZ(\textbf{ex-generalize}\ \exists\, x\, .\, F\ \ \textbf{from}\ \ t) & = & 1 \\ SZ(\textbf{pick-any}\ x\ \ D) & = & 1 + SZ(D) \\ SZ(\textbf{pick-witness}\ w\ \ \textbf{for}\ \exists\, x\, .\, F\ \ D) & = & 1 + SZ(D) \end{array}$$

Both conditional deductions (of the form **assume** $F\ \ D$) and proofs by contradiction (of the form **suppose-absurd** $F\ \ D$) are called *hypothetical deductions.* In both cases, F and D are the *hypothesis* and *body* of the deduction, respectively. We also say that the body D represents the *scope* of the hypothesis F. In universal generalizations of the form **pick-any** $x\ \ D$, we refer to x as an *eigenvariable* and to D as the *body*; we also say that D represents the *scope* of x. In existential instantiations

$$\textbf{pick-witness}\ w\ \ \textbf{for}\ \exists\, x\, .\, F\ \ D$$

the variable w is called the *witness variable* (or simply "the witness"), while D is the body of the proof; we also say that w is an eigenvariable, and that D represents the scope of w. We write $EV(D)$ for the set of eigenvariables that appear in D. A *trivial* deduction is a claim, i.e., an atomic deduction of the form **claim** F. We write u.g. and e.i. as abbreviations for "universal generalization" and "existential instantiation," respectively.

A deduction is *well-formed* iff every primitive rule application in it has one of the forms shown in Figure 1.3. Thus, loosely put, a deduction is well-formed iff the right number and kind of arguments are supplied to every application of a primitive rule. It is straightforward to check whether a deduction is well-formed; from now on we will only be concerned with well-formed deductions. We stipulate that the composition operator is right-associative. A maximal-length composition $D_1; \ldots; D_n$ is called a *thread.* The last element of a thread is said to be in a *tail position.* Ambiguities in the parsing of $\mathcal{NDL}$ deductions will be resolved by the use of **begin-end** pairs and/or parentheses.

Substitutions can also be applied to deductions in a straightforward manner. The only slight complication is presented by **pick-any** and **pick-witness**. Both of these introduce scope, so we must be careful to avoid variable capture. We will say that a substitution θ is *safe* for a deduction D iff $\textit{RanVar}(\theta) \cap EV(D) = \emptyset$.[7] In general, for any substitution θ and proof D, we define $\theta(D)$, the result

[7] We will see eventually that, because the eigenvariables of a deduction D can be renamed to our liking without altering the meaning of D, any substitution θ can be considered safe for any deduction D.

$$\frac{}{D \equiv_\epsilon D} \qquad \frac{D_1 \equiv_\epsilon D_1' \quad D_2 \equiv_\epsilon D_2'}{D_1; D_2 \equiv_\epsilon D_1'; D_2'}$$

$$\frac{D_1 \equiv_\epsilon D_2}{\textbf{assume } F \;\; D_1 \equiv_\epsilon \textbf{assume } F \;\; D_2}$$

$$\frac{D_1 \equiv_\epsilon D_2}{\textbf{suppose-absurd } F \;\; D_1 \equiv_\epsilon \textbf{suppose-absurd } F \;\; D_2}$$

$$\frac{\{x_1 \mapsto y\}\, D_1 \equiv_\epsilon \{x_2 \mapsto y\}\, D_2}{\textbf{pick-any } x_1 \;\; D_1 \equiv_\epsilon \textbf{pick-any } x_2 \;\; D_2}$$

where y does not occur in D_1, D_2, $y \notin \{x_1, x_2\}$.

$$\frac{\{w_1 \mapsto y\}\, D_1 \equiv_\epsilon \{w_2 \mapsto y\}\, D_2}{\textbf{pick-witness } w_1 \textbf{ for } F \;\; D_1 \equiv_\epsilon \textbf{pick-witness } w_2 \textbf{ for } F \;\; D_2}$$

where y does not occur in D_1, D_2, F, $y \notin \{w_1, w_2\}$.

Figure 1.1: Definition of the eigenvariance relation $\equiv_\epsilon$.

of applying θ to D, as follows:

$$\begin{array}{rcl}
\theta(\textit{Prim-Rule } F_1, \ldots, F_n) & = & \textit{Prim-Rule } \theta(F_1), \ldots, \theta(F_n) \\
\theta(\textbf{specialize } F \textbf{ with } t) & = & \textbf{specialize } \theta(F) \textbf{ with } \theta(t) \\
\theta(\textbf{ex-generalize } F \textbf{ from } t) & = & \textbf{ex-generalize } \theta(F) \textbf{ from } \theta(t) \\
\theta(\textbf{assume } F \;\; D) & = & \textbf{assume } \theta(F) \;\; \theta(D) \\
\theta(\textbf{suppose-absurd } F \;\; D) & = & \textbf{suppose-absurd } \theta(F) \;\; \theta(D) \\
\theta(D_1; D_2) & = & \theta(D_1); \theta(D_2) \\
\theta(\textbf{pick-any } x \;\; D) & = & \textbf{pick-any } x \;\; \theta[x \mapsto x](D) \\
\theta(\textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; D) & = & \textbf{pick-witness } w \textbf{ for } \theta(\exists\, x \,.\, F) \;\; \theta[w \mapsto w](D)
\end{array}$$

As with terms and formulas, we will often write $\theta\, D$ as a shorthand for $\theta(D)$.

Finally, two deductions will be considered identical iff each can be obtained from the other by consistently renaming eigenvariables. This relation of "eigenvariance," denoted by $\equiv_\epsilon$, is defined by the rules shown in Figure 1.1. The following lemma shows that the eigenvariables of any deduction can be "renamed away" from any particular set of variables; the result of the renaming will be eigenvariant to the original deduction. In tandem with Theorem 1.15, which will show that eigenvariant deductions are observationally equivalent, this will entail that consistently renaming the eigenvariables of a deduction does not affect the latter's meaning—in the same way that consistently renaming the bound variables of a λ-calculus term does not change the term's meaning.

Lemma 1.1 *There is an algorithm that takes any deduction D and any finite set of variables V and produces a deduction D' such that $D \equiv_\epsilon D'$ and $EV(D') \cap V = \emptyset$.*

The following lemmas are useful for the proofs of some subsequent results. (Proofs can be found in Chapter 6 of [2].)

$$\frac{\beta \cup \{F\} \vdash D \leadsto G}{\beta \vdash \textbf{assume}\ F \;\; D \leadsto F \Rightarrow G} \qquad \frac{\beta \cup \{F\} \vdash D \leadsto \textbf{false}}{\beta \vdash \textbf{suppose-absurd}\ F \;\; D \leadsto \neg F}$$

$$\frac{\beta \vdash D_1 \leadsto F_1 \qquad \beta \cup \{F_1\} \vdash D_2 \leadsto F_2}{\beta \vdash D_1; D_2 \leadsto F_2} \qquad \frac{\beta \vdash \{x \mapsto v\}\, D \leadsto F}{\beta \vdash \textbf{pick-any}\ x \;\; D \leadsto \forall\, v \,.\, F}$$

whenever v does not occur in β or in D.

$$\frac{}{\beta \cup \{\forall\, x \,.\, F\} \vdash \textbf{specialize}\ \forall\, x \,.\, F\ \textbf{with}\ t \leadsto \{x \mapsto t\}F}$$

$$\frac{}{\beta \cup \{x \mapsto t\}F \vdash \textbf{ex-generalize}\ \exists\, x \,.\, F\ \textbf{from}\ t \leadsto \exists\, x \,.\, F}$$

$$\frac{\beta \cup \{\exists\, x \,.\, F, \{x \mapsto v\}\, F\} \vdash \{w \mapsto v\}\, D \leadsto G}{\beta \cup \{\exists\, x \,.\, F\} \vdash \textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x \,.\, F \;\; D \leadsto G}$$

whenever v does not occur in $\beta \cup \{\exists\, x \,.\, F\}$ or in D, and $v \notin FV(G)$.

Figure 1.2: Formal $\mathcal{NDL}$ semantics

Lemma 1.2 *If θ_1 and θ_2 are disjoint then $\theta_2\,\theta_1\,F = \theta_1\,\theta_2\,F$.*

Lemma 1.3 *Let $\sigma = \{x_1 \mapsto x_2\}$, $\tau = \{x_2 \mapsto x_3\}$, $\theta = \{x_1 \mapsto x_3\}$. If $x_2 \notin FV(F)$, $\tau\,\sigma\,F = \theta\,F$.*

Lemma 1.4 *If x_2 does not occur in D then $\{x_2 \mapsto x_1\}\{x_1 \mapsto x_2\}(D) = D$.*

1.3.2 Semantics

The semantics of $\mathcal{NDL}$ are given by judgments of the form $\beta \vdash D \leadsto F$, which are read as: "Evaluating D in β produces the conclusion F." The semantics of rule applications appear in Figure 1.3. The semantics of compound deductions, universal instantiations and existential generalizations are shown in Figure 1.2. As an example, the following $\mathcal{NDL}$ deduction derives the tautology

$$\forall\, x \,.\, P(x) \Rightarrow \neg\, \exists\, x \,.\, \neg P(x)$$

in the empty assumption base (assuming that P is a unary relation symbol):

```
assume ∀ x . P(x)
  suppose-absurd ∃ x . ¬P(x)
    pick-witness w for ∃ x . ¬P(x)
      begin
        specialize ∀ x . P(x) with w;
        absurd P(w), ¬P(w)
      end
```

Theorem 1.5 (Dilution) *If $\beta \vdash D \leadsto F$ then $\beta \cup \beta' \vdash D \leadsto F$.*

$$\begin{array}{c}
\beta \cup \{F\} \vdash \textbf{claim } F \rightsquigarrow F \\
\beta \cup \{F \Rightarrow G, F\} \vdash \textbf{modus-ponens } F \Rightarrow G, F \rightsquigarrow G \\
\beta \cup \{\neg\neg F\} \vdash \textbf{double-negation } \neg\neg F \rightsquigarrow F \\
\beta \cup \{F_1, F_2\} \vdash \textbf{both } F_1, F_2 \rightsquigarrow F_1 \wedge F_2 \\
\beta \cup \{F_1 \wedge F_2\} \vdash \textbf{left-and } F_1 \wedge F_2 \rightsquigarrow F_1 \\
\beta \cup \{F_1 \wedge F_2\} \vdash \textbf{right-and } F_1 \wedge F_2 \rightsquigarrow F_2 \\
\beta \cup \{F_1\} \vdash \textbf{left-either } F_1, F_2 \rightsquigarrow F_1 \vee F_2 \\
\beta \cup \{F_2\} \vdash \textbf{right-either } F_1, F_2 \rightsquigarrow F_1 \vee F_2 \\
\beta \cup \{F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G\} \vdash \textbf{cases } F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G \rightsquigarrow G \\
\beta \cup \{F_1 \Rightarrow F_2, F_2 \Rightarrow F_1\} \vdash \textbf{equivalence } F_1 \Rightarrow F_2, F_2 \Rightarrow F_1 \rightsquigarrow F_1 \Leftrightarrow F_2 \\
\beta \cup \{F_1 \Leftrightarrow F_2\} \vdash \textbf{left-iff } F_1 \Leftrightarrow F_2 \rightsquigarrow F_1 \Rightarrow F_2 \\
\beta \cup \{F_1 \Leftrightarrow F_2\} \vdash \textbf{right-iff } F_1 \Leftrightarrow F_2 \rightsquigarrow F_2 \Rightarrow F_1 \\
\beta \cup \{F, \neg F\} \vdash \textbf{absurd } F, \neg F \rightsquigarrow \textbf{false} \\
\beta \vdash \textbf{true-intro} \rightsquigarrow \textbf{true}
\end{array}$$

Figure 1.3: Evaluation axioms for rule applications.

Theorem 1.6 *If $\beta \vdash D \rightsquigarrow F$ and θ is safe for D then $\theta\,\beta \vdash \theta\, D \rightsquigarrow \theta\, F$.*

The *conclusion* of a deduction D, denoted $\mathcal{C}(D)$, is defined by structural recursion:

$$\mathcal{C}(\textbf{specialize } \forall\, x\,.\,F \textbf{ with } t) = \{x \mapsto t\}\, F \tag{1.5}$$

$$\mathcal{C}(\textbf{ex-generalize } \exists\, x\,.\,F \textbf{ from } t) = \exists\, x\,.\,F \tag{1.6}$$

$$\mathcal{C}(\textbf{assume } F\ \ D) = F \Rightarrow \mathcal{C}(D) \tag{1.7}$$

$$\mathcal{C}(\textbf{suppose-absurd } F\ \ D) = \neg\, F \tag{1.8}$$

$$\mathcal{C}(D_1; D_2) = \mathcal{C}(D_2) \tag{1.9}$$

$$\mathcal{C}(\textbf{pick-any } x\ \ D) = \forall\, x\,.\,\mathcal{C}(D) \tag{1.10}$$

$$\mathcal{C}(\textbf{pick-witness } x \textbf{ for } F\ \ D) = \mathcal{C}(D) \tag{1.11}$$

For rule applications we have:

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{C}(\textbf{modus-ponens } F \Rightarrow G, F) & = & G & \mathcal{C}(\textbf{right-either } F, G) & = & F \vee G \\
\mathcal{C}(\textbf{double-negation } \neg\neg F) & = & F & \mathcal{C}(\textbf{cases } F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G) & = & G \\
\mathcal{C}(\textbf{both } F, G) & = & F \wedge G & \mathcal{C}(\textbf{equivalence } F \Rightarrow G, G \Rightarrow F) & = & F \Leftrightarrow G \\
\mathcal{C}(\textbf{left-and } F \wedge G) & = & F & \mathcal{C}(\textbf{left-iff } F \Leftrightarrow G) & = & F \Rightarrow G \\
\mathcal{C}(\textbf{right-and } F \wedge G) & = & G & \mathcal{C}(\textbf{right-iff } F \Leftrightarrow G) & = & G \Rightarrow F \\
\mathcal{C}(\textbf{left-either } F, G) & = & F \vee G & \mathcal{C}(\textbf{absurd } F, \neg F) & = & \textbf{false} \\
\mathcal{C}(\textbf{claim } F) & = & F & \mathcal{C}(\textbf{true-intro}) & = & \textbf{true}
\end{array}$$

Lemma 1.7 $\mathcal{C}(\theta\, D) = \theta\, \mathcal{C}(D)$.

Lemma 1.8 *If x does not occur in D then x does not occur in $\mathcal{C}(D)$.*

Theorem 1.9 *If $\beta \vdash D \rightsquigarrow F$ then $F = \mathcal{C}(D)$.*

Corollary 1.10 *If $\beta \vdash D \rightsquigarrow F_1$ and $\beta \vdash D \rightsquigarrow F_2$ then $F_1 = F_2$.*

$$\begin{array}{l}
OA(\textbf{left-either } F_1, F_2) = \{F_1\} \\
OA(\textbf{right-either } F_1, F_2) = \{F_2\} \\
OA(\textit{Prim-Rule } F_1, \ldots, F_n) = \{F_1, \ldots, F_n\} \\
OA(\textbf{specialize } \forall\, x \,.\, F \textbf{ with } t) = \{\forall\, x \,.\, F\} \\
OA(\textbf{ex-generalize } \exists\, x \,.\, F \textbf{ from } t) = \{\{x \mapsto t\}\, F\} \\
OA(\textbf{assume } F \;\; D) = OA(D) - \{F\} \\
OA(\textbf{suppose-absurd } F \;\; D) = OA(D) - \{F\} \\
OA(D_1; D_2) = OA(D_1) \cup [OA(D_2) - \{\mathcal{C}(D_1)\}] \\
OA(\textbf{pick-any } x \;\; D) = \textit{let } \; \Phi = OA(D) \\
\qquad\qquad\qquad\qquad\quad \textit{in} \\
\qquad\qquad\qquad\qquad\qquad \Phi = \textit{error} \rightarrow \textit{error} \diamondsuit\; [x \in FV(\Phi) \rightarrow \textit{error}, \Phi] \\
OA(\textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \;\; D) = \\
\quad x \in FV(\mathcal{C}(D)) \rightarrow \textit{error} \diamondsuit \\
\qquad\qquad\qquad\quad \textit{let } \; \Phi = OA(D) \\
\qquad\qquad\qquad\quad \textit{in} \\
\qquad\qquad\qquad\qquad \Phi = \textit{error} \rightarrow \textit{error} \diamondsuit\; \textit{let } \; \Psi = \Phi - \{\{y \mapsto x\}\, F\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\; x \in FV(\Psi) \rightarrow \textit{error} \diamondsuit\; \Psi \cup \{\exists\, y \,.\, F\}
\end{array}$$

Figure 1.4: Definition of $OA(D)$, the open assumptions of a proof D.

Figure 1.4 defines $OA(D)$, the set of *open assumptions* of a proof D. The elements of $OA(D)$ are formulas that D uses as premises, without proof. Note in particular the equations for hypothetical deductions: the open assumptions here are those of the body D *minus* the hypothesis F. We will say that the elements of $OA(D)$ are *strictly used* by D. A value of *error* indicates that the deduction is erroneous, in the sense that it could not possibly yield any conclusion, in any assumption base. This will be formally captured by Theorem 1.11 below, which is an important technical result stating that a deduction successfully produces its conclusion iff the assumption base contains all its open assumptions.

Observe that when we write $OA(D) \subseteq \beta$ we tacitly imply $OA(D) \neq \textit{error}$. In general, we adopt the convention that in any context in which an expression such as $OA(D)$ would have to denote a set of formulas for some enclosing expression to be meaningful, we are tacitly conjoining the qualification $OA(D) \neq \textit{error}$. Accordingly, the full content of Theorem 1.11 below is: $\beta \vdash D \leadsto \mathcal{C}(D)$ iff $OA(D) \neq$ *error* and $\beta \supseteq OA(D)$. This convention is not necessary for an identity such as $OA(D_1) = OA(D_2)$, as the values of $OA(D_1)$ and $OA(D_2)$ do not have to be sets for such an identity to be meaningful. In particular, this equality is considered valid iff both $OA(D_1)$ and $OA(D_2)$ are *error*, or else both denote the same set of formulas.

Theorem 1.11 *$\beta \vdash D \leadsto \mathcal{C}(D)$ iff $OA(D) \subseteq \beta$.*

By analogy with $OA(D)$, we define $OV(D)$, the set of "open variables" of D, as

$$OV(D) = FV(OA(D)) \cup FV(\mathcal{C}(D))$$

The following lemma is readily proved by induction on D:

Lemma 1.12 *If $v \notin OV(D)$ then $\{v \mapsto s\}(D) = D$.*

We say that two deductions D_1 and D_2 are observationally equivalent with respect to an assumption base β, written $D_1 \approx_\beta D_2$, whenever

$$\beta \vdash D_1 \leadsto F \quad \text{iff} \quad \beta \vdash D_2 \leadsto F$$

for all F. We say that D_1 and D_2 are *observationally equivalent*, written $D_1 \approx D_2$, iff we have $D_1 \approx_\beta D_2$ for all β.

Lemma 1.13 *If $D_1 \approx D_2$ then $\mathcal{C}(D_1) = \mathcal{C}(D_2)$.*

Proof: Set $\beta = OA(D_1) \cup OA(D_2)$. By Theorem 1.11, we have $\beta \vdash D_1 \leadsto \mathcal{C}(D_1)$, so the assumption $D_1 \approx D_2$ entails $\beta \vdash D_2 \leadsto \mathcal{C}(D_1)$. But Theorem 1.11 also gives $\beta \vdash D_2 \leadsto \mathcal{C}(D_2)$, hence $\mathcal{C}(D_1) = \mathcal{C}(D_2)$ by Corollary 1.10. ■

Theorem 1.14 *$D_1 \approx D_2$ iff $OA(D_1) = OA(D_2)$ and $\mathcal{C}(D_1) = \mathcal{C}(D_2)$. Therefore, observational equivalence is decidable.*

Proof: In one direction, suppose that $OA(D_1) = OA(D_2)$ and $\mathcal{C}(D_1) = \mathcal{C}(D_2)$. Then, for any β and F, we have:

$$\begin{array}{lll}
\beta \vdash D_1 \leadsto F & \text{iff} & \text{(by Theorem 1.9 and Theorem 1.11)} \\
F = \mathcal{C}(D_1) \text{ and } \beta \supseteq OA(D_1) & \text{iff} & \text{(by the assumptions } \mathcal{C}(D_1) = \mathcal{C}(D_2) \text{ and } OA(D_1) = OA(D_2)) \\
F = \mathcal{C}(D_2) \text{ and } \beta \supseteq OA(D_2) & \text{iff} & \text{(by Theorem 1.9 and Theorem 1.11)} \\
\beta \vdash D_2 \leadsto F. & &
\end{array}$$

This shows that $D_1 \approx D_2$.

Conversely, suppose that $D_1 \approx D_2$. Then $\mathcal{C}(D_1) = \mathcal{C}(D_2)$ follows from Lemma 1.13. Moreover, by Theorem 1.11, $OA(D_1) \vdash D_1 \leadsto \mathcal{C}(D_1)$, so the assumption $D_1 \approx D_2$ entails $OA(D_1) \vdash D_2 \leadsto \mathcal{C}(D_1)$. Therefore, by Theorem 1.11,

$$OA(D_1) \supseteq OA(D_2). \tag{1.12}$$

Likewise, we have $OA(D_2) \vdash D_2 \leadsto \mathcal{C}(D_2)$, so $D_1 \approx D_2$ implies $OA(D_2) \vdash D_1 \leadsto \mathcal{C}(D_2)$, and hence Theorem 1.11 gives

$$OA(D_2) \supseteq OA(D_1) \tag{1.13}$$

and now $OA(D_1) = OA(D_2)$ follows from 1.12 and 1.13. ■

Theorem 1.15 *$\equiv_\epsilon \subseteq \approx$; i.e., eigenvariant deductions are observationally equivalent.*

Observational equivalence is a very strong condition. Oftentimes we are only interested in replacing a deduction D_1 by some D_2 on the assumption that D_1 will yield its conclusion in the intended β (i.e., on the assumption that its evaluation will not lead to error), even though we might have $D_1 \not\approx D_2$. To take a simple example, although we have **claim** $F; D \not\approx D$ (pick D to be **true-intro** and consider any β that does not contain F), it is true that in any given assumption base, *if* **claim** $F; D$ produces some conclusion G *then* so will D. (In fact this observation will be the formal justification for a transformation we will introduce later for removing redundant claims.) We formalize this relation as follows.

We write $D_1 \rightarrowtail_\beta D_2$ to mean that, for all F,

$$\text{if } \beta \vdash D_1 \leadsto F \text{ then } \beta \vdash D_2 \leadsto F.$$

And we write $D_1 \rightarrowtail D_2$ to mean that $D_1 \rightarrowtail_\beta D_2$ for *all* β.

Clearly, $\rightarrowtail$ is not a symmetric relation: We vacuously have **claim false**; **true-intro** $\rightarrowtail$ **true-intro**, but the converse does not hold. However, $\rightarrowtail$ is a quasi-order (reflexive and transitive), and in fact $\approx$ is the contensive equality generated by the weaker relation's symmetric closure.

Lemma 1.16 *$\rightarrowtail$ is a quasi-order whose symmetric closure coincides with $\approx$. Accordingly, $D_1 \approx D_2$ iff $D_1 \rightarrowtail D_2$ and $D_2 \rightarrowtail D_1$.*

It will be useful to note that $\rightarrowtail$ is compatible with the syntactic constructs of $\mathcal{NDL}$:

Lemma 1.17 *If $D_1 \rightarrowtail D_1'$, $D_2 \rightarrowtail D_2'$ then* **assume** F **in** $D_1 \rightarrowtail$ **assume** F **in** D_1', $D_1; D_2 \rightarrowtail D_1'; D_2'$; **suppose-absurd** F **in** $D_1 \rightarrowtail$ **suppose-absurd** F **in** D_1'; **pick-any** x $D_1 \rightarrowtail$ **pick-any** x D_1' *and* **pick-witness** w **for** $\exists\, x\,.\, F$ $D_1 \rightarrowtail$ **pick-witness** w **for** $\exists\, x\,.\, F$ D_1'.

Reasoning similar to that used in the proof of Theorem 1.11 will show:

Theorem 1.18 *$D_1 \rightarrowtail D_2$ iff $\mathcal{C}(D_1) = \mathcal{C}(D_2)$ and $OA(D_1) \supseteq OA(D_2)$. Therefore, the relation $\rightarrowtail$ is decidable.*

Finally, the following two results will help us to justify a "hoisting" transformation that we will define later:

Theorem 1.19 *If $F \notin OA(D_1)$ then* (a) **assume** F $(D_1; D_2) \rightarrowtail D_1;$ **assume** F D_2;

(b) **suppose-absurd** F $(D_1; D_2) \rightarrowtail D_1;$ **suppose-absurd** F D_2.

Proof: We prove part (a); part (b) is similar. Suppose $\beta \vdash$ **assume** F $(D_1; D_2) \leadsto F \Rightarrow Q$, so that $\beta \cup \{F\} \vdash D_1; D_2 \leadsto Q$. Accordingly,

$$\beta \cup \{F\} \vdash D_1 \leadsto F_1 \tag{1.14}$$

and

$$\beta \cup \{F\} \cup \{F_1\} \vdash D_2 \leadsto Q \tag{1.15}$$

(where, of course, $F_1 = \mathcal{C}(D_1)$, $Q = \mathcal{C}(D_2)$). Thus 1.15 gives

$$\beta \cup \{F_1\} \vdash \textbf{assume } F \;\; D_2 \leadsto F \Rightarrow Q. \tag{1.16}$$

From 1.14 and Theorem 1.11, $\beta \cup \{F\} \supseteq OA(D_1)$, hence, since $F \notin OA(D_1)$,

$$\beta \supseteq OA(D_1). \tag{1.17}$$

Therefore,

$$\beta \vdash D_1 \leadsto F_1 \tag{1.18}$$

so, from 1.16 and 1.18, rule $[R_3]$ gives $\beta \vdash D_1;$ **assume** F $D_2 \leadsto F \Rightarrow Q$, which establishes (a). ■

Theorem 1.20 *If* $x \notin OV(D_1)$ *then*

$$\textbf{pick-any } x \ (D_1; D_2) \rightarrowtail D_1; \textbf{pick-any } x \ D_2$$

and $\textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ (D_1; D_2) \rightarrowtail D_1; \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ D_2$.

The relation $\rightarrowtail$ will serve as our formal notion of safety for the transformations that will be introduced. That is, whenever a transformation maps a deduction D_1 to some D_2, we will have $D_1 \rightarrowtail D_2$. This is an appropriate notion of safety in the context of certificates [6], because if D_1 is a certificate then presumably we already know that it works; we are only interested in making it more efficient or succinct. For other applications, however, if we wish our transformations to be perfectly safe then we should insist on observational equivalence. For $D_1 \approx D_2$ means that the two deductions behave identically in all contexts, i.e., in all assumption bases. For any β, if D_1 fails in β then D_2 will fail in β as well; while if D_1 produces a conclusion F in β, then D_2 will produce that same conclusion in β. Accordingly, the replacement of D_1 by D_2 would be a completely semantics-preserving transformation.

We close this section by introducing three derived inference rules that will come handy in the sequel: the binary rule **cond** and the two unary rules **neg** and $\textbf{gen}_x$ (parameterized over a variable x). Applications of these rules are defined as syntax sugar in terms of existing rules as follows:

$$\begin{aligned} \textbf{cond } F, G &\implies \textbf{assume } F \textbf{ claim } G \\ \textbf{neg } F &\implies \textbf{suppose-absurd } F \textbf{ claim false} \\ \textbf{gen}_x \ F &\implies \textbf{pick-any } x \textbf{ claim } F \end{aligned}$$

The reader will verify the following:

Lemma 1.21 (a) $\beta \cup \{G\} \vdash \textbf{cond } F, G \leadsto F \Rightarrow G$*;* $OA(\textbf{cond } F, G) = \{G\}$*;* $\mathcal{C}(\textbf{cond } F, G) = F \Rightarrow G$*.* (b) $\beta \cup \{\textbf{false}\} \vdash \textbf{neg } F \leadsto \neg F$*;* $OA(\textbf{neg } F) = \{\textbf{false}\}$*;* $\mathcal{C}(\textbf{neg } F) = \neg F$*.* (c) *If* $x \notin FV(F)$ *then* $\beta \cup \{F\} \vdash \textbf{gen}_x \ F \leadsto \forall\, x \,.\, F$*;* $OA(\textbf{gen}_x \ F) = \{F\}$*; and* $\mathcal{C}(\textbf{gen}_x \ F) = \forall\, x \,.\, F$*.*

1.4 Contracting transformations

Informally, our contracting transformations will be based on two simple principles:

Productivity: *Every intermediate conclusion should be used at some later point as an argument to a primitive inference rule.*

Parsimony: *At no point should a non-trivial deduction establish something that has already been established, or something that has been hypothetically postulated.*

These principles are respectively based on the notions of *redundancies* and *repetitions*, which we will now study in detail.

1.4.1 Redundancies

Intuitively, a deduction contains redundancies if it derives conclusions which are not subsequently used. For all practical purposes, such derivations are useless "noise." We will see that they can be systematically eliminated. Redundancy-free deductions will be called *strict*. As a very simple counterexample, the following deduction, which proves $F \wedge G \Rightarrow F$, is not strict:

$$\textbf{assume } F \wedge G \textbf{ in begin right-and } F \wedge G; \textbf{left-and } F \wedge G; \textbf{end}$$

$$\frac{}{\vdash_S D} \text{ whenever } D \text{ is atomic} \qquad \frac{\vdash_S D}{\vdash_S \textbf{assume } F \;\; D}$$

$$\frac{\vdash_S D}{\vdash_S \textbf{suppose-absurd } F \;\; D} \qquad \frac{\vdash_S D_1 \quad \vdash_S D_2 \quad \mathcal{C}(D_1) \in OA(D_2)}{\vdash_S D_1; D_2}$$

$$\frac{\vdash_S D}{\vdash_S \textbf{pick-any } x \;\; D} \qquad \frac{\vdash_S D \quad \{x \mapsto w\}F \in OA(D)}{\vdash_S \textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; D}$$

Figure 1.5: Definition of *strict* deductions.

The redundancy here is the application of **right-and** to derive G. This is superfluous because it plays no role in the derivation of the final conclusion. We formally define the judgment $\vdash_S D$, "D is strict," in Figure 1.5. Verbally, the definition can be put as follows:

- *Atomic deductions are always strict.*
- *Hypothetical deductions and universal generalizations are strict if their respective bodies are strict.*
- *An existential instantiation is strict if its body is strict and strictly uses the witness premise.*
- *A composite deduction $D_1; D_2$ is strict if both D_1 and D_2 are strict, and the conclusion of D_1 is strictly used in D_2.*

The last of the above clauses is the most important one. Note that we require that $\mathcal{C}(D_1)$ be *strictly* used in D_2. Accordingly, the deduction

$$\textbf{left-and } F \wedge G; \textbf{ assume } F \;\; \textbf{both } F, F$$

is not strict: the derivation of F via **left-and** is extraneous because the only subsequent use of F, as a premise to **both** inside the **assume**, has been "buffered" by the hypothetical postulation of F.

We will now present a transformation algorithm $\mathfrak{U}$ that converts a given deduction D into a strict deduction D'. We will prove that $\vdash_S D'$, and also that the semantics of D are conservatively preserved in the sense that $D \rightarrowtail D'$. The transformation is defined by structural recursion:

$$
\begin{aligned}
&\mathfrak{U}(\textbf{assume } F \;\; D) = \textbf{assume } F \;\; \mathfrak{U}(D)\\
&\mathfrak{U}(\textbf{suppose-absurd } F \;\; D) = \textbf{suppose-absurd } F \;\; \mathfrak{U}(D)\\
&\mathfrak{U}(D_1; D_2) = \mathit{let} \;\; D_1' = \mathfrak{U}(D_1)\\
&\qquad\qquad\qquad\quad\;\; D_2' = \mathfrak{U}(D_2)\\
&\qquad\qquad\quad\;\; \mathit{in}\\
&\qquad\qquad\qquad \mathcal{C}(D_1') \notin OA(D_2') \rightarrow D_2' \diamondsuit D_1'; D_2'\\
&\mathfrak{U}(\textbf{pick-any } x \;\; D) = \textbf{pick-any } x \;\; \mathfrak{U}(D)\\
&\mathfrak{U}(\textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; D) = \mathit{let} \;\; D' = \mathfrak{U}(D)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\;\; \mathit{in}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\;\; \{x \mapsto w\}F \in OA(D') \rightarrow \textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; D' \diamondsuit D'\\
&\mathfrak{U}(D) = D
\end{aligned}
$$

Informally, it is easy to see that $D \rightarrowtail \mathfrak{U}(D)$ because $\mathfrak{U}(D)$ does not introduce any additional open assumptions (though it might eliminate some of the open assumptions of D), and does not alter $\mathcal{C}(D)$. Therefore, by Theorem 1.18, we have $D \rightarrowtail \mathfrak{U}(D)$. More precisely:

Theorem 1.22 (a) $\mathfrak{U}$ *always terminates;* (b) $\mathfrak{U}(D)$ *is strict;* (c) $D \rightarrowtail \mathfrak{U}(D)$.

Proof: Termination is clear, since the size of the argument strictly decreases with each recursive call. We prove (b) and (c) simultaneously by structural induction on D.

The basis case of atomic deductions is immediate. When D is of the form **assume** $F\ D_b$, we have

$$\mathfrak{U}(D) = \textbf{assume } F\ \mathfrak{U}(D_b). \tag{1.19}$$

By the inductive hypothesis, $\mathfrak{U}(D_b)$ is strict, hence so is $\mathfrak{U}(D)$, by the definition of strictness. Further, again by the inductive hypothesis, we have $D_b \rightarrowtail \mathfrak{U}(D_b)$, hence by Lemma 1.17 we get

$$\textbf{assume } F\ D_b \rightarrowtail \textbf{assume } F\ \mathfrak{U}(D_b)$$

which is to say, by virtue of (1.19), that $D \rightarrowtail \mathfrak{U}(D)$. The reasoning for proofs by contradiction is similar.

Next, suppose that D is a composite deduction $D_1; D_2$ and let $D'_1 = \mathfrak{U}(D_1), D'_2 = \mathfrak{U}(D_2)$. Either $\mathcal{C}(D'_1) \in OA(D'_2)$ or not. If so, then $\mathfrak{U}(D) = D'_1; D'_2$, and strictness follows from the inductive hypothesis and our supposition that $\mathcal{C}(D'_1) \in OA(D'_2)$, according to the definition of $\vdash_S$; while $D \rightarrowtail \mathfrak{U}(D)$ in this case means $D_1; D_2 \rightarrowtail D'_1; D'_2$, which follows from the inductive hypotheses in tandem with Lemma 1.17. By contrast, suppose that $\mathcal{C}(D'_1) \notin OA(D'_2)$, so that $\mathfrak{U}(D) = D'_2$. Since $D = D_1; D_2 \rightarrowtail D'_1; D'_2$ follows from the inductive hypotheses and Lemma 1.17, if we can show that $D'_1; D'_2 \rightarrowtail D'_2$ then $D \rightarrowtail D'_2 = \mathfrak{U}(D)$ will follow from the transitivity of $\rightarrowtail$ (Lemma 1.16). Accordingly, pick any β and G, and suppose that $\beta \vdash D'_1; D'_2 \rightsquigarrow G$ (where, of course, by Theorem 1.9 we must have $G = \mathcal{C}(D'_1; D'_2) = \mathcal{C}(D'_2)$). By Theorem 1.11, this means that

$$\beta \supseteq OA(D'_1; D'_2) \tag{1.20}$$

But the supposition $\mathcal{C}(D'_1) \notin OA(D'_2)$ entails, by the definition of open assumptions, that $OA(D'_1; D'_2) = OA(D'_1) \cup OA(D'_2)$, so (1.20) gives $\beta \supseteq OA(D'_2)$. Therefore, Theorem 1.11 implies $\beta \vdash D'_2 \rightsquigarrow \mathcal{C}(D'_2) = G$. We have thus shown that for any β and G, if $\beta \vdash D'_1; D'_2 \rightsquigarrow G$ then $\beta \vdash D'_2 \rightsquigarrow G$, which is to say $D'_1; D'_2 \rightarrowtail D'_2$. It follows from our earlier remarks that $D = D_1; D_2 \rightarrowtail D'_2 = \mathfrak{U}(D)$.

When D is of the form **pick-any** $x\ D$, the result follows directly from the inductive hypothesis and Lemma 1.17.

Finally, suppose that D is an existential instantiation of the form

$$\textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F\ D_b$$

Let $D'_b = \mathfrak{U}(D_b)$. We distinguish two cases:

1. $\{x \mapsto w\}F \notin OA(D'_b)$. In that case $\mathfrak{U}(D) = D'_b$, so we need to show that D'_b is strict and that $D \rightarrowtail D'_b$. The former follows immediately from the inductive hypothesis. For the latter, consider an arbitrary assumption base β and suppose that $\beta \vdash D \rightsquigarrow G$. By the semantics of **pick-witness**, this entails $\exists\, x \,.\, F \in \beta$ and

$$\beta \cup \{\{x \mapsto z\}\, F\} \vdash \{w \mapsto z\}\, D_b \rightsquigarrow G \tag{1.21}$$

for some fresh variable z (not occuring in D or in β) and such that $z \notin FV(G)$. The substitution $\{z \mapsto w\}$ is safe for $\{w \mapsto z\}D_b$ (we can always ensure this by renaming the eigenvariables of $\{w \mapsto z\}D_b$ on the basis of Lemma 1.1 if necessary), hence (1.21) and Theorem 1.6 imply

$$\{z \mapsto w\}\,\beta \cup \{\{z \mapsto w\}\{x \mapsto z\}\,F\} \vdash \{z \mapsto w\}\{w \mapsto z\}\,D_b \rightsquigarrow \{z \mapsto w\}\,G \tag{1.22}$$

Since z does not occur in β or in G, we have $\{z \mapsto w\}\,\beta = \beta$ and $\{z \mapsto w\}\,G = G$. Further, since $z \notin FV(F)$, Lemma 1.3 gives

$$\{z \mapsto w\}\{x \mapsto z\}\,F = \{x \mapsto w\}\,F$$

Finally, by Lemma 1.4 and the assumption that z does not occur in D_b we infer

$$\{z \mapsto w\}\{w \mapsto z\}\,D_b = D_b$$

Accordingly, (1.22) becomes

$$\beta \cup \{\{x \mapsto w\}\,F\} \vdash D_b \rightsquigarrow G \tag{1.23}$$

Now since $D_b \rightarrowtail D'_b$ (by the inductive hypothesis), (1.23) gives

$$\beta \cup \{\{x \mapsto w\}F\} \vdash D'_b \rightsquigarrow G \tag{1.24}$$

By virtue of Theorem 1.11, (1.24) gives

$$\beta \cup \{\{x \mapsto w\}F\} \supseteq OA(D'_b) \tag{1.25}$$

But $\{x \mapsto w\}F \notin OA(D'_b)$, hence (1.25) yields $\beta \supseteq OA(D'_b)$. Therefore, by Theorem 1.11 we conclude $\beta \vdash D'_b \rightsquigarrow G$.

2. $\{x \mapsto w\}F \in OA(D'_b)$. In that case the result of the algorithm is

$$D_r = \textbf{pick-witness } w \textbf{ for } \exists\, x\,.\,F \;\; D'_b$$

By the inductive hypothesis, D'_b is strict, therefore, by the formal definition of strictness and the supposition $\{x \mapsto w\}F \in OA(D'_b)$, we infer that D_r is also strict. Also by the inductive hypothesis, $D_b \rightarrowtail D'_b$, hence $D \rightarrowtail D_r$ by Lemma 1.17.

This completes the case analysis and the inductive proof. ■

As an illustration, suppose we wish to use the algorithm to remove redundancies from the deduction

$$D_1; D_2; \textbf{both } F, G; \textbf{left-either } F, H \tag{1.26}$$

where $\mathcal{C}(D_1) = F, \mathcal{C}(D_2) = G$. Assuming that D_1 and D_2 are already strict, the interesting reduction steps taken by the algorithm, in temporal order, may be depicted as follows (where we use the arrow $\Longrightarrow$ to represent a reduction step):

1. $\textbf{both } F, G; \textbf{left-either } F, H \Longrightarrow \textbf{left-either } F, H$ (as $F \wedge G \notin OA(\textbf{left-either}$ F, H))
2. $D_2; \textbf{left-either } A, H \Longrightarrow \textbf{left-either } F, H$ (as $\mathcal{C}(D_2) = G \notin OA(\textbf{left-either}$ F, H))
3. $D_2; \textbf{both } F, G; \textbf{left-either } F, H \Longrightarrow D_2; \textbf{left-either } F, H$ (from 1)
4. $D_2; \textbf{both } F, G; \textbf{left-either } F, H \Longrightarrow \textbf{left-either } F, H$ (from 2 and 3)
5. $D_1; D_2; \textbf{both } F, G; \textbf{left-either } F, H \Longrightarrow D_1; \textbf{left-either } F, H$ (from 4)

Thus the original deduction becomes reduced to $D_1; \textbf{left-either } F, H$.

1.4.2 Repetitions

The principle of productivity alone cannot guarantee that a deduction will not have superfluous components. For instance, consider a slight modification of example (1.26):

$$D_1; D_2; \textbf{both } F, G; \textbf{left-and } F \wedge G \tag{1.27}$$

where again $\mathcal{C}(D_1) = F$, $\mathcal{C}(D_2) = G$. The difference with (1.26) is that the last deduction is

$$\textbf{left-and } F \wedge G$$

instead of **left-either** F, H. In this case algorithm $\mathfrak{U}$ will have no effect because the deduction is already strict: D_1 establishes F; D_2 establishes G; then we use both F and G to obtain $F \wedge G$; and finally we use **left-and** $F \wedge G$ to get F. Thus the principle of productivity is observed. The principle of parsimony, however, is violated: the **left-and** deduction establishes something (F) which has already been established by D_1. For that reason, it is extraneous, and hence so are the derivations of G and $F \wedge G$.

This example illustrates what Prawitz called a *detour*: the gratuitous application of an introduction rule followed by the application of a corresponding elimination rule that gets us back to a premise that was supplied to the introduction rule. The reason why these are detours is because elimination rules are the inverses of introduction rules. Prawitz enunciated this intuition with a statement that he called "the inversion principle."

It is important to realize that Prawitz's reductions are not readily applicable in $\mathcal{NDL}$. Detours may not be freely replaced by their obvious contractions; the greater context in which the subdeduction occurs will determine whether the replacement is permissible. For example, the boxed subdeduction below indicates a detour, but we may not blindly simplify it because $\mathcal{C}(D_2)$, or $\mathcal{C}(D_1) \wedge \mathcal{C}(D_2)$, or both, might be needed inside D':

$$\cdots; D_1; \boxed{D_2; \textbf{both } \mathcal{C}(D_1), \mathcal{C}(D_2); \textbf{left-and } \mathcal{C}(D_1) \wedge \mathcal{C}(D_2)}; \cdots D' \cdots$$

What we can do, however, is replace the inference **left-and** $\mathcal{C}(D_1) \wedge \mathcal{C}(D_2)$ by the trivial **claim** $\mathcal{C}(D_1)$. A subsequent strictness analysis will determine whether $\mathcal{C}(D_2)$ or $\mathcal{C}(D_1) \wedge \mathcal{C}(D_2)$ are needed at any later point. If not, then we can be sure that the deductions D_2 and **both** $\mathcal{C}(D_1), \mathcal{C}(D_2)$ were indeed a detour, and algorithm $\mathfrak{U}$ will eliminate them. We will see that this simple technique of

1. replacing every deduction whose conclusion P has already been established by the trivial deduction that claims P, and then

2. removing redundancies with our productivity analysis

will be sufficient for the elimination of most Prawitz-type detours. The first step can result in a deduction with various trivial claims sprinkled throughout. This is mostly a cosmetic annoyance; a simple contracting analysis that we will present shortly will eliminate all extraneous claims. That analysis will always be performed at the end of all other transformations in order to clean up the final result.

Figure 1.6 depicts an algorithm $\mathfrak{P}$ for performing the first step of the above process.

Lemma 1.23 If $\beta \vdash RR(D, \Phi) \rightsquigarrow F$ then $\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \rightsquigarrow F$.

```
$\mathfrak{P}(D) = RR(D, \emptyset)$
where
$RR(D, \Phi) = \mathcal{C}(D) \in \Phi \rightarrow$ claim $\mathcal{C}(D) \diamondsuit$
  match $D$
    assume $F$ in $D_b \rightarrow$ assume $F$ in $RR(D_b, \Phi \cup \{F\})$
    suppose-absurd $F$ in $D_b \rightarrow$ suppose-absurd $F$ in $RR(D_b, \Phi \cup \{F\})$
    $D_1; D_2 \rightarrow$ let $D_1' = RR(D_1, \Phi)$
               in
                 $D_1'; RR(D_2, \Phi \cup \{\mathcal{C}(D_1')\})$
    pick-any $x$ $D_b \rightarrow$ pick-any $x$ $RR(D_b, \Phi)$
    pick-witness $w$ for $\exists\, x\,.\,F$ $D_b \rightarrow$ pick-witness $w$ for $\exists\, x\,.\,F$ $RR(D_b, \Phi \cup \{\{x \mapsto w\}\, F\})$
    $D \rightarrow D$
```

Figure 1.6: Algorithm for removing repetitions.

Proof: By induction on D. Suppose first that $\mathcal{C}(D) \in \Phi$, so that $RR(D, \Phi) =$ **claim** $\mathcal{C}(D)$. In that case, by the semantics of **claim**, the assumption $\beta \vdash RR(D, \Phi) \leadsto F = \mathcal{C}(D)$ entails $\mathcal{C}(D) \in \beta$. Therefore,

$$\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) = \textbf{claim } \mathcal{C}(D) \leadsto F$$

Now suppose $\mathcal{C}(D) \notin \Phi$. We proceed by a case analysis of the structure of D. Suppose first that D is of the form **assume** H D_b. We then have

$$\beta \vdash \textbf{assume } H \;\; RR(D_b, \Phi \cup \{H\}) \leadsto F = H \Rightarrow G$$

for some G, so that

$$\beta \cup \{H\} \vdash RR(D_b, \Phi \cup \{H\}) \leadsto G$$

By the inductive hypothesis,

$$\beta \cup \Psi \cup \{H\} \vdash RR(D_b, \Phi \cup \Psi \cup \{H\}) \leadsto G$$

hence

$$\beta \cup \Psi \vdash \textbf{assume } H \;\; RR(D_b, \Phi \cup \Psi \cup \{H\}) \leadsto H \Rightarrow G = F$$

i.e., $\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \leadsto F$.

When D is of the form **suppose-absurd** H D_b, we have:

$$\beta \vdash \textbf{suppose-absurd } H \;\; RR(D_b, \Phi \cup \{H\}) \leadsto F = \neg H$$

so that $\beta \cup \{H\} \vdash RR(D_b, \Phi \cup \{H\}) \leadsto$ **false**. Inductively,

$$\beta \cup \Psi \cup \{H\} \vdash RR(D_b, \Phi \cup \Psi \cup \{H\}) \leadsto \textbf{false}$$

and therefore

$$\beta \cup \Psi \vdash \textbf{suppose-absurd } H \;\; RR(D_b, \Phi \cup \Psi \cup \{H\}) \leadsto \neg H = F$$

i.e., $\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \leadsto F$.

Next, suppose that D is a composition $D_1; D_2$. Then the assumption $\beta \vdash RR(D, \Phi) \leadsto F$ entails

$$\beta \vdash RR(D_1, \Phi); RR(D_2, \Phi \cup \{\mathcal{C}(RR(D_1, \Phi))\}) \leadsto F \tag{1.28}$$

so that

$$\beta \vdash RR(D_1, \Phi) \leadsto G \tag{1.29}$$

and

$$\beta \cup \{G\} \vdash RR(D_2, \Phi \cup \{G\}) \leadsto F \tag{1.30}$$

where $G = \mathcal{C}(RR(D_1, \Phi))$. By the inductive hypothesis, (1.29) and (1.30) yield, respectively,

$$\beta \cup \Psi \vdash RR(D_1, \Phi \cup \Psi) \leadsto G \tag{1.31}$$

and

$$\beta \cup \Psi \cup \{G\} \vdash RR(D_2, \Phi \cup \Psi \cup \{G\}) \leadsto F \tag{1.32}$$

which means $\mathcal{C}(RR(D_1, \Phi \cup \Psi)) = G$. Therefore,

$$\beta \cup \Psi \vdash RR(D_1, \Phi \cup \Psi); RR(D_2, \Phi \cup \Psi \cup \{G\}) \leadsto F$$

i.e., $\beta \cup \Psi \vdash RR(D_1; D_2, \Phi \cup \Psi) \leadsto F$.

When D is of the form **pick-any** $x \;\; D_b$, we have $RR(D, \Phi) =$ **pick-any** $x \;\; RR(D_b, \Phi)$, so the assumption $\beta \vdash RR(D, \Phi) \leadsto F$ means

$$\beta \vdash \textbf{pick-any } x \;\; RR(D_b, \Phi) \leadsto F \tag{1.33}$$

By the semantics of **pick-any**, (1.33) means that $F = \forall\, z \,.\, G$ for some G and some z that does not occur in $RR(D, \Phi)$ or in β, and such that

$$\beta \vdash \{x \mapsto z\}\, RR(D_b, \Phi) \leadsto G \tag{1.34}$$

Without loss of generality, we may assume that $\{z \mapsto x\}$ is safe for $\{x \mapsto z\}\, RR(D_b, \Phi)$ (we can always ensure this by renaming the eigenvariables of $\{x \mapsto z\}\, RR(D_b, \Phi)$), hence Theorem 1.6 and (1.34) imply

$$\{z \mapsto x\}\, \beta \vdash \{z \mapsto x\}\, \{x \mapsto z\}\, RR(D_b, \Phi) \leadsto \{z \mapsto x\}\, G \tag{1.35}$$

Since z does not occur in $RR(D_b, \Phi)$, Lemma 1.4 gives

$$\{z \mapsto x\}\, \{x \mapsto z\}\, RR(D_b, \Phi) = RR(D_b, \Phi)$$

and, since z also does not occur in β, we have $\{z \mapsto x\}\, \beta = \beta$. Thus (1.35) becomes:

$$\beta \vdash RR(D_b, \Phi) \leadsto \{z \mapsto x\}G \tag{1.36}$$

The inductive hypothesis now gives

$$\beta \cup \Psi \vdash RR(D_b, \Phi \cup \Psi) \leadsto \{z \mapsto x\}G \tag{1.37}$$

The substitution $\{x \mapsto z\}$ is safe for $RR(D_b, \Phi \cup \Psi)$, hence Theorem 1.6 and (1.37) entail

$$\{x \mapsto z\}\, \beta \cup \{x \mapsto z\}\, \Psi \vdash \{x \mapsto z\}\, RR(D_b, \Phi \cup \Psi) \leadsto \{x \mapsto z\}\, \{z \mapsto x\}G \tag{1.38}$$

Without loss of generality, we may assume that x does not occur in β or in Ψ (we can always rename D to ensure this), and therefore

$$\{x \mapsto z\}\,\beta = \beta \tag{1.39}$$

and

$$\{x \mapsto z\}\,\Psi = \Psi \tag{1.40}$$

Moreover, x does not occur in G (this follows from (1.34), Lemma 1.8, Theorem 1.9, and the fact that x does not occur in $\{x \mapsto z\}\,RR(D_b, \Phi)$). Accordingly, by Lemma 1.3 we get

$$\{x \mapsto z\}\,\{z \mapsto x\}\,G = G \tag{1.41}$$

Now (1.39), (1.40), and (1.41) transform (1.38) into:

$$\beta \cup \Psi \vdash \{x \mapsto z\}\,RR(D_b, \Phi \cup \Psi) \rightsquigarrow G$$

and hence, by the semantics of universal generalizations,

$$\beta \cup \Psi \vdash \textbf{pick-any}\ x\ \ RR(D_b, \Phi \cup \Psi) \rightsquigarrow \forall\, z\,.\,G = F$$

which is to say $\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \rightsquigarrow F$.

When D is an existential instantiation of the form **pick-witness** w **for** $\exists x\,.\,G\ \ D_b$, the assumption $\beta \vdash RR(D, \Phi) \rightsquigarrow F$ means that

$$\beta \vdash \textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x\,.\,G\ \ RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\}) \rightsquigarrow F \tag{1.42}$$

so that $\exists\, x\,.\,G \in \beta$ and, for some fresh z,

$$\beta \cup \{\{x \mapsto z\}G\} \vdash \{w \mapsto z\}\,RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\}) \rightsquigarrow F \tag{1.43}$$

where z does not occur in F. Since $\{z \mapsto w\}$ is safe for $\{w \mapsto z\}\,RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\})$, Theorem 1.6 and (1.43) imply

$$\{z \mapsto w\}\,\beta \cup \{\{z \mapsto w\}\,\{x \mapsto z\}\,G\} \vdash \{z \mapsto w\}\,\{w \mapsto z\}\,RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\}) \rightsquigarrow \{z \mapsto w\}\,F$$

Because z is fresh, we have $\{z \mapsto w\}\beta = \beta$; $\{z \mapsto w\}\,\{x \mapsto z\}\,G = \{x \mapsto w\}\,G$ (by Lemma 1.3); and $\{z \mapsto w\}\,\{w \mapsto z\}\,RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\}) = RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\})$ (by Lemma 1.4). Therefore, the preceding evaluation judgment becomes:

$$\beta \cup \{\{x \mapsto w\}G\} \vdash RR(D_b, \Phi \cup \{\{x \mapsto w\}\,G\}) \rightsquigarrow F \tag{1.44}$$

Inductively, (1.44) gives

$$\beta \cup \Psi \cup \{\{x \mapsto w\}G\} \vdash RR(D_b, \Phi \cup \Psi \cup \{\{x \mapsto w\}\,G\}) \rightsquigarrow F \tag{1.45}$$

Now $\{w \mapsto z\}$ is safe for $RR(D_b, \Phi \cup \Psi \cup \{\{x \mapsto w\}\,G\})$, hence Theorem 1.6 and (1.45) give

$$\begin{gathered}\{w \mapsto z\}\,\beta \cup \{w \mapsto z\}\,\Psi \cup \{\{w \mapsto z\}\,\{x \mapsto w\}G\} \vdash \\ \{w \mapsto z\}\,RR(D_b, \Phi \cup \Psi \cup \{\{x \mapsto w\}\,G\}) \rightsquigarrow \{w \mapsto z\}\,F\end{gathered} \tag{1.46}$$

Without loss of generality, we may assume that w does not occur in β, or in Ψ or in G, and therefore

$$\begin{aligned}\{w \mapsto z\}\,\beta &= \beta && (1.47)\\ \{w \mapsto z\}\,\Psi &= \Psi && (1.48)\\ \{w \mapsto z\}\,\{x \mapsto w\}\,G &= \{x \mapsto z\}\,G \quad \text{(by Lemma 1.3)} && (1.49)\end{aligned}$$

In addition, w does not occur $RR(D_b, \Phi \cup \Psi \cup \{\{x \mapsto w\}\, G\})$, hence, by Lemma 1.8, (1.43), and Theorem 1.9, we infer that w does not occur in F, so that

$$\{w \mapsto z\}\, F = F \tag{1.50}$$

Finally, (1.46) along with (1.47), (1.48), (1.49), and (1.50) yield

$$\begin{gathered}\beta \cup \Psi \cup \{\{x \mapsto z\}\, G\} \vdash \\ \{w \mapsto z\}\, RR(D_b, \Phi \cup \Psi \cup \{\{x \mapsto w\}\, G\}) \rightsquigarrow F\end{gathered} \tag{1.51}$$

so, by the semantics of **pick-witness** and $\exists\, x \,.\, G \in \beta$, we obtain

$$\beta \cup \Psi \vdash \textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, G \;\; RR(D_b, \Phi \cup \Psi \cup \{\{x \mapsto w\}\, G\}) \rightsquigarrow F$$

which is to say $\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \rightsquigarrow F$.

Finally, suppose that $\mathcal{C}(D) \notin \Phi$ and D is not of any of the above forms. Then $RR(D, \Phi) = D$, so we have

$$\beta \vdash D \rightsquigarrow F = \mathcal{C}(D) \tag{1.52}$$

We distinguish two cases:

1. $\mathcal{C}(D) \in \Psi$: In that case $RR(D, \Phi \cup \Psi) = \textbf{claim } \mathcal{C}(D)$, so

   $$\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \rightsquigarrow \mathcal{C}(D) = F$$

   follows by the semantics of **claim**.

2. $\mathcal{C}(D) \notin \Psi$: Then $\mathcal{C}(D) \notin \Phi \cup \Psi$, and hence

   $$RR(D, \Phi \cup \Psi) = D \tag{1.53}$$

   The desired judgment $\beta \cup \Psi \vdash RR(D, \Phi \cup \Psi) \rightsquigarrow F$ now follows from (1.52), (1.53), and dilution.

This completes the case analysis and the inductive argument. ■

Theorem 1.24 $D \rightarrowtail \mathfrak{P}(D)$.

Proof: We will prove that $D \rightarrowtail RR(D, \emptyset)$ by induction on D. When D is an atomic deduction, $RR(D, \emptyset) = D$, so the result is immediate since $\rightarrowtail$ is reflexive. When D is of the form

$$\textbf{assume } F \;\; D_b,$$

$RR(D, \emptyset) = \textbf{assume } F \;\; RR(D_b, \{F\})$, so to show $D \rightarrowtail RR(D, \emptyset)$ we need to prove that if

$$\beta \vdash \textbf{assume } F \;\; D_b \rightsquigarrow F \Rightarrow G \tag{1.54}$$

then

$$\beta \vdash \textbf{assume } F \;\; RR(D_b, \{F\}) \rightsquigarrow F \Rightarrow G. \tag{1.55}$$

On the assumption that (1.54) holds, we have

$$\beta \cup \{F\} \vdash D_b \rightsquigarrow G. \tag{1.56}$$

By the inductive hypothesis, $D_b \rightarrowtail RR(D_b, \emptyset)$, so from (1.56) we get

$$\beta \cup \{F\} \vdash RR(D_b, \emptyset) \rightsquigarrow G$$

and by Lemma 1.23, $\beta \cup \{F\} \vdash RR(D_b, \{F\}) \rightsquigarrow G$. Therefore,

$$\beta \vdash \textbf{assume } F \;\; RR(D_b, \{F\}) \rightsquigarrow F \Rightarrow G$$

which is the desired (1.55).

Proofs by contradiction are handled similarly. Specifically, suppose that D is of the form

$$\textbf{suppose-absurd } F \;\; D_b$$

and assume $\beta \vdash \textbf{suppose-absurd } F \;\; D_b \rightsquigarrow \neg F$, for arbitrary β, so that

$$\beta \cup \{F\} \vdash D_b \rightsquigarrow \textbf{false}. \tag{1.57}$$

Inductively, $D_b \rightarrowtail RR(D_b, \emptyset)$, so (1.57) gives

$$\beta \cup \{F\} \vdash RR(D_b, \emptyset) \rightsquigarrow \textbf{false}.$$

Therefore, Lemma 1.23 yields

$$\beta \cup \{F\} \vdash RR(D_b, \{F\}) \rightsquigarrow \textbf{false}$$

and this implies $\beta \vdash \textbf{suppose-absurd } F \;\; RR(D_b, \{F\}) \rightsquigarrow \neg F$. We have thus shown that

$$\textbf{suppose-absurd } F \;\; D_b \rightarrowtail \textbf{suppose-absurd } F \;\; RR(D_b, \{F\})$$

which is to say $D \rightarrowtail RR(D, \emptyset)$.

Finally, suppose that D is of the form $D_1; D_2$ and that $\beta \vdash D_1; D_2 \rightsquigarrow G$, so that

$$\beta \vdash D_1 \rightsquigarrow F \tag{1.58}$$

and

$$\beta \cup \{F\} \vdash D_2 \rightsquigarrow G. \tag{1.59}$$

We have $RR(D, \emptyset) = D_1'; D_2'$, where

$$D_1' = RR(D_1, \emptyset) \tag{1.60}$$

and

$$D_2' = RR(D_2, \mathcal{C}(D_1')) \tag{1.61}$$

From the inductive hypothesis, $D_1 \rightarrowtail RR(D_1, \emptyset)$, hence from (1.58),

$$\beta \vdash RR(D_1, \emptyset) \rightsquigarrow F \tag{1.62}$$

so from (1.60),

$$\beta \vdash D_1' \rightsquigarrow F \tag{1.63}$$

and

$$\mathcal{C}(D_1') = F. \tag{1.64}$$

Likewise, $D_2 \rightarrowtail RR(D_2, \emptyset)$, so from (1.59),

$$\beta \cup \{F\} \vdash RR(D_2, \emptyset) \rightsquigarrow G$$

and from Lemma 1.23,

$$\beta \cup \{F\} \vdash RR(D_2, \{F\}) \rightsquigarrow G$$

which, from (1.61) and (1.64) means

$$\beta \cup \{F\} \vdash D_2' \rightsquigarrow G. \tag{1.65}$$

Finally, from (1.63) and (1.65) we obtain $\beta \vdash D_1'; D_2' \rightsquigarrow G$, and thus we infer $D \rightarrowtail RR(D, \emptyset) = D_1'; D_2'$.

When D is of the form **pick-any** $x\;\; D_b$, the inductive hypothesis gives $D_b \rightarrowtail RR(D_b, \emptyset)$, and now $D \rightarrowtail RR(D, \emptyset)$ follows from Lemma 1.17 and the definition of RR.

Finally, suppose that D is an existential instantiation **pick-witness** w **for** $\exists\, x \,.\, F\;\; D_b$ and assume $\beta \vdash D \rightsquigarrow G$, so that

$$\beta \cup \{\{x \mapsto z\}\, F\} \vdash \{w \mapsto z\}\, D_b \rightsquigarrow G \tag{1.66}$$

for some fresh variable z, where $z \notin FV(G)$ and $\exists\, x \,.\, F \in \beta$. The substitution $\{z \mapsto w\}$ is safe for $\{w \mapsto z\}\, D_b$, hence Theorem 1.6 and (1.66) imply

$$\{z \mapsto w\}\, \beta \cup \{\{z \mapsto w\}\, \{x \mapsto z\}\, F\} \vdash \{z \mapsto w\}\, \{w \mapsto z\}\, D_b \rightsquigarrow \{z \mapsto w\}\, G \tag{1.67}$$

Since z does not occur in $\beta \cup \{F\}$ or in D_b, Lemma 1.3 and Lemma 1.4 transform (1.67) into

$$\beta \cup \{\{x \mapsto w\}\, F\} \vdash D_b \rightsquigarrow \{z \mapsto w\}\, G \tag{1.68}$$

By the inductive hypothesis, $D_b \rightarrowtail RR(D_b, \emptyset)$, so (1.68) yields

$$\beta \cup \{\{x \mapsto w\}\, F\} \vdash RR(D_b, \emptyset) \rightsquigarrow \{z \mapsto w\}\, G \tag{1.69}$$

By Lemma 1.23, (1.69) gives

$$\beta \cup \{\{x \mapsto w\}\, F\} \vdash RR(D_b, \{\{x \mapsto w\}\, F\}) \rightsquigarrow \{z \mapsto w\}\, G \tag{1.70}$$

Now $\{w \mapsto z\}$ is safe for $RR(D_b, \{\{x \mapsto w\}\, F\})$, hence Theorem 1.6 and (1.70) yield

$$\{w \mapsto z\}\, \beta \cup \{\{w \mapsto z\}\, \{x \mapsto w\}\, F\} \vdash \{w \mapsto z\}\, RR(D_b, \{\{x \mapsto w\}\, F\}) \rightsquigarrow \{w \mapsto z\}\, \{z \mapsto w\}\, G \tag{1.71}$$

Without loss of generality, we may assume that w does not occur in F or in β (we can always rename w to ensure this), hence $\{w \mapsto z\}\, \beta = \beta$, while Lemma 1.3 implies

$$\{w \mapsto z\}\, \{x \mapsto w\}\, F = \{x \mapsto z\} F$$

Moreover, w does not occur in G (this follows from (1.66), Lemma 1.8, Theorem 1.9, and the fact that w does not occur in $\{w \mapsto z\}\, D_b$), hence

$$\{w \mapsto z\}\, \{z \mapsto w\}\, G = G$$

Accordingly, (1.71) becomes

$$\beta \cup \{\{x \mapsto z\}\, F\} \vdash \{w \mapsto z\}\, RR(D_b, \{\{x \mapsto w\}\, F\}) \rightsquigarrow G$$

and therefore, since $\exists\, x \,.\, F \in \beta$,

$$\beta \vdash \textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x \,.\, F\;\; RR(D_b, \{\{x \mapsto w\}\, F\}) \rightsquigarrow G$$

which is to say $\beta \vdash RR(D, \emptyset) \rightsquigarrow G$.

■

1.4.3 Claim elimination

The third and final contracting transformation we will present is particularly simple: it eliminates all claims in non-tail positions. It is readily verified that all such claims are superfluous. For example, the claim in

$$D = \textbf{dn}\ \neg\neg F; \textbf{claim}\ G; \textbf{both}\ F, F$$

can be removed because $D \rightarrowtail \textbf{dn}\ \neg\neg F; \textbf{both}\ F, F$.

Claims in tail positions cannot in general be removed, since they serve as conclusions. One exception, however, occurs when the claim of some F is the last element of a thread whose immediately preceding element concludes F. In those cases the claim can be removed despite being in tail position. An example is

$$\textbf{dn}\ \neg\neg F; \textbf{both}\ F, G; \textbf{claim}\ F \wedge G.$$

Here the tail claim of $F \wedge G$ can be eliminated because it is derived by the immediately dominating deduction **both** F, G.

The following algorithm removes all claims in non-tail positions, as well as all extraneous tail claims of the sort discussed above:

```
$\mathfrak{C}(D) =$
  $match\ \ D$
    $\textbf{assume}\ F\ \ D_b \to \textbf{assume}\ F\ \ \mathfrak{C}(D_b)$
    $\textbf{suppose-absurd}\ F\ \ D_b \to \textbf{suppose-absurd}\ F\ \ \mathfrak{C}(D_b)$
    $D_1; D_2 \to\ \ let\ \ D'_1 = \mathfrak{C}(D_1)$
                  $D'_2 = \mathfrak{C}(D_2)$
             $in$
               $claim?(D'_1) \to D'_2 \diamondsuit claim?(D'_2)\ \ and\ \ \mathcal{C}(D'_1) = \mathcal{C}(D'_2) \to D'_1 \diamondsuit D'_1; D'_2$
    $\textbf{pick-any}\ x\ \ D_b \to \textbf{pick-any}\ x\ \ \mathfrak{C}(D_b)$
    $\textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x\,.\, F\ \ D_b \to \textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x\,.\, F\ \ \mathfrak{C}(D_b)$
    $D \to D$
```

where *claim?*(D) returns true iff D is an application of **claim**. We have:

Lemma 1.25 **claim** $F; D \rightarrowtail D$. *Further,* $D;$ **claim** $F \rightarrowtail D$ *whenever* $\mathcal{C}(D) = F$.

Using this lemma, a straightforward induction will show that $D \rightarrowtail \mathfrak{C}(D)$. Termination is immediate.

Theorem 1.26 $\mathfrak{C}$ *always terminates. In addition,* $D \rightarrowtail \mathfrak{C}(D)$.

Another property that will prove useful is the following:

Lemma 1.27 *Let* $D_1; \ldots; D_n; D_{n+1}$ *be a chain in* $\mathfrak{C}(D)$, $n > 0$. *Then* $\forall\, i \in \{1, \ldots, n\}$, D_i *is not a claim.*

Recall from (1.2) that the contracting phase of *simplify* is defined as

$$contract = fp\ \ (\mathfrak{C} \cdot \mathfrak{P} \cdot \mathfrak{U})$$

For any given D, let us write $NT(D)$ to denote the number of non-trivial subdeductions of D, i.e., the number of subdeductions of D that are not claims. Define a quantity $Q(D)$ as the pair $(SZ(D), NT(D))$. A simple induction on D will show:

- $\mathfrak{U}\ D = D$ or $SZ(\mathfrak{U}\ D) < SZ(D)$;
- $\mathfrak{P}\ D = D$ or else $SZ(\mathfrak{P}\ D) < SZ(D)$ or $SZ(\mathfrak{P}\ D) = SZ(D)$ and $NT(\mathfrak{P}\ D) < NT(D)$;
- $\mathfrak{C}\ D = D$ or $SZ(\mathfrak{C}\ D) < SZ(D)$.

Therefore, writing $(a_1, b_1) <_{lex} (a_2, b_2)$ to mean that the pair (a_1, b_1) is lexicographically $<$-smaller than (a_2, b_2), we have:

Lemma 1.28 *For all D, either $(\mathfrak{C} \cdot \mathfrak{P} \cdot \mathfrak{U})\ D = D$ or else $Q((\mathfrak{C} \cdot \mathfrak{P} \cdot \mathfrak{U})\ D) <_{lex} Q(D)$.*

It follows that the fixed-point algorithm will eventually converge, since an infinitely long chain of distinct deductions $D_1, D_2, \ldots$ produced by repeated applications of $\mathfrak{C} \cdot \mathfrak{P} \cdot \mathfrak{U}$ would entail

$$Q(D_i) >_{lex} Q(D_{i+1})$$

for all i, which is impossible since $>_{lex}$ is well-founded. It also follows from Lemma 1.28 that the size of the final result of *contract* will not be greater than the size of the original input. Finally, $D \rightarrowtail contract(D)$ follows from Theorem 1.22, Theorem 1.24, Theorem 1.26, and the transitivity of $\rightarrowtail$. We summarize:

Theorem 1.29 *The contraction procedure always terminates. In addition, $D \rightarrowtail contract(D)$ and $SZ(contract(D)) \leq SZ(D)$.*

1.5 Restructuring transformations

1.5.1 Scope maximization

The most fundamental restructuring transformation is *scope maximization.* Intuitively, this aims at making the conclusion of a subdeduction available to as many subsequent inferences as possible. Scope can be limited in three ways: with bracketing (**begin-end** pairs); with assumption scope; and with eigenvariable scope. We examine each case below.

Left-linear compositions

One factor that can affect conclusion visibility is left-linear composition, namely, deductions of the form $(D_1; D_2); D_3$, where the conclusion of D_1 is only available to D_2. Such deductions are rare in practice because the natural threading style in $\mathcal{NDL}$ is right-associative (which is why composition associates to the right by default). When they occur, left-linear compositions can complicate our parsimony analysis. Consider, for instance, $D = (D_1; D_2); D_3$ where $\mathcal{C}(D_1) = \mathcal{C}(D_3)$. Algorithm $\mathfrak{P}$ might well find D to be repetition-free even though, intuitively, it is clear that D_3 unnecessarily duplicates the work of D_1. The problem is the limited scope of D_1: As long as D_2 does not replicate the conclusion of D_1 and D_3 the conclusion of $D_1; D_2$, i.e., the conclusion of D_2, then D will be deemed repetition-free. The problem can be avoided by right-associating D, thereby maximizing the scope of D_1. The algorithm $\mathfrak{RL}$ presented below converts every subdeduction of the form $(D_1; D_2); D_3$ into $D_1; (D_2; D_3)$. Our proof that this is a safe transformation will be based on Lemma 1.30. The proof of the lemma is straightforward and omitted, but the intuition is important: in both cases D_1 is available to D_2, and D_2 to D_3, but in $D_1; (D_2; D_3)$ we *also* have D_1 available to D_3. So if $(D_1; D_2); D_3$ goes through, then certainly $D_1; (D_2; D_3)$ will do too.

Lemma 1.30 *$(D_1; D_2); D_3 \rightarrowtail D_1; (D_2; D_3)$.*

Let us say that a deduction D is *right-linear* iff it has no subdeductions of the form $(D_1; D_2); D_3$. The following is immediate:

Lemma 1.31 *If D is right-linear then so is any hypothetical deduction, u.g., or e.i. with body D. Moreover, if D_1 and D_2 are right-linear and D_1 is not a composite deduction then $D_1; D_2$ is right-linear.*

Algorithm $\mathfrak{RL}$ will transform any given D into a right-linear D' such that $D \rightarrowtail D'$:

$$\begin{array}{rcl}
\mathfrak{RL}(\textbf{assume } F \;\; D) & = & \textbf{assume } F \;\; \mathfrak{RL}(D) \\
\mathfrak{RL}(\textbf{suppose-absurd } F \;\; D) & = & \textbf{suppose-absurd } F \;\; \mathfrak{RL}(D) \\
 & & \textit{match } D_l \\
\mathfrak{RL}(D_l; D_r) & = & \quad D_1; D_2 \rightarrow \mathfrak{RL}(D_1; (D_2; D_r)) \\
 & & \quad \_ \rightarrow \mathfrak{RL}(D_l); \mathfrak{RL}(D_r) \\
\mathfrak{RL}(\textbf{pick-any } x \;\; D) & = & \textbf{pick-any } x \;\; \mathfrak{RL}(D) \\
\mathfrak{RL}(\textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; D) & = & \textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; \mathfrak{RL}(D) \\
\mathfrak{RL}(D) & = & D
\end{array}$$

For our termination proof, let us define a quantity $LSZ(D)$ as follows: if D is of the form $D_1; D_2$ then $LSZ(D) = SZ(D_1)$; otherwise $LSZ(D) = 0$.

Theorem 1.32 $\mathfrak{RL}$ *always terminates.*

Proof: We claim that with each recursive call, the pair $(SZ(D), LSZ(D))$ strictly decreases lexicographically.[8] This can be seen by checking each recursive call: in the recursive calls for **assume**, **suppose-absurd**, **pick-any** and **pick-witness**, the size of D strictly decreases. In the recursive call $\mathfrak{RL}(D_1; (D_2; D_r))$ the size does not increase, while the quantity LSZ strictly decreases. Finally, in both recursive calls in the next line, the size strictly decreases. ■

Theorem 1.33 $\mathfrak{RL}(D)$ *is right-linear. Furthermore,* $D \rightarrowtail \mathfrak{RL}(D)$.

Proof: Let us write $D_1 \prec D_2$ to mean that the pair $(SZ(D_1), LSZ(D_1))$ is lexicographically smaller than $(SZ(D_2), LSZ(D_2))$. We will use well-founded induction on the relation $\prec$, i.e., we will show that for all deductions D, if the result holds for every D' such that $D' \prec D$ then it also holds for D. We proceed by a case analysis of an arbitrary D. If D is an atomic deduction then $\mathfrak{RL}(D) = D$ and the result follows immediately. If D is a conditional deduction with hypothesis F and body D' then $\mathfrak{RL}(D) = \textbf{assume } F \;\; \mathfrak{RL}(D')$. Since $D' \prec D$, the inductive hypothesis entails that $\mathfrak{RL}(D')$ is right-linear and that $D' \rightarrowtail \mathfrak{RL}(D')$. The same reasoning is used for proofs by contradiction. The result now follows from Lemma 1.31 and Lemma 1.17.

Finally, suppose that D is of the form $D_l; D_r$. Then either D_l is of the form $D_1; D_2$, or not. In the first case we have

$$\mathfrak{RL}(D) = \mathfrak{RL}(D_1; (D_2; D_r)) \tag{1.72}$$

and since $D = (D_1; D_2); D_r \succ D_1; (D_2; D_r)$, we conclude inductively that

(i) $\mathfrak{RL}(D_1; (D_2; D_r))$ is right linear, and

(ii) $D_1; (D_2; D_r) \rightarrowtail \mathfrak{RL}(D_1; (D_2; D_r))$.

[8] Using the lexicographic extension of $<$ to pairs of natural numbers: (n_1, n_2) is smaller than (n_1', n_2') iff $n_1 < n_1'$ or else $n_1 = n_1'$ and $n_2 < n_2'$.

Thus the conclusion that $\mathfrak{RL}(D)$ is right-linear follows from (i) and (1.72), while

$$D \rightarrowtail \mathfrak{RL}(D) = \mathfrak{RL}(D_1; (D_2; D_r))$$

follows from (ii) and the transitivity of $\rightarrowtail$, since $D \rightarrowtail D_1; (D_2; D_r)$ from Lemma 1.30. If D_l is not of the form $D_1; D_2$ then

$$\mathfrak{RL}(D) = \mathfrak{RL}(D_l); \mathfrak{RL}(D_r) \tag{1.73}$$

and since $D_l \prec D$, $D_r \prec D$, the inductive hypothesis entails that (i) $\mathfrak{RL}(D_l)$ and $\mathfrak{RL}(D_r)$ are right-linear, and (ii) $D_l \rightarrowtail \mathfrak{RL}(D_l)$, $D_r \rightarrowtail \mathfrak{RL}(D_r)$. Because D_l is not a composite deduction, neither is $\mathfrak{RL}(D_l)$ (a necessary condition for $\mathfrak{RL}(D)$ to be composite is that D be composite), hence it follows from Lemma 1.31 and (1.73) that $\mathfrak{RL}(D)$ is right-linear. Further, $D \rightarrowtail \mathfrak{RL}(D)$ follows from (ii) and Lemma 1.17. This concludes the case analysis and the inductive argument. ■

Assumption and eigenvariable scope

The other cases of undue scope limitation arise in hypothetical deductions and in eigenvariable deductions (universal generalizations and existential instantiations). Consider a hypothetical deduction with body D_b and hypothesis F. If D is a subdeduction of D_b then its scope cannot extend beyond D_b. But this need not be the case if D is not strictly dependent on the hypothesis F. If there is no such dependence, then D is unnecessarily restricted by being inside D_b. Its scope should be maximized by *hoisting* it outside D_b. As a simple example, consider

```
assume $G$ in
  begin
    double-negation $\neg\neg F$;
    both $F, G$
  end
```

Here the subdeduction **double-negation** $\neg\neg F$ makes no use of the hypothesis G, and therefore it is appropriate to pull it outside, resulting in

```
double-negation $\neg\neg F$;
assume $G$ in
  both $F, G$
```

This deduction is observationally equivalent to the first one, and has a cleaner structure that better reflects the various logical dependencies. Besides increased clarity, hoisting will greatly facilitate our repetition analysis later on. Repetitions are much easier to detect and eliminate when they are in the same scope. Consider, for instance, the deduction

```
assume $G$ in
  begin
    double-negation $\neg\neg F$;
    both $F, G$
  end;
left-and $F \wedge H$;
both $F, G \Rightarrow F \wedge G$
```

In view of **double-negation** $\neg\neg F$, the deduction **left-and** $F \wedge H$ is superfluous, but this is not easy to determine mechanically because the former deduction lies inside the scope of the hypothesis G. More importantly, neither deduction can be safely eliminated as things stand, even though it is clearly extraneous to have both of them. If we eliminated the **double-negation** then the **assume** might fail; while if we eliminated the **left-and**, the composition might fail. But if we hoist the double negation outside of the **assume**, resulting in

```
double-negation $\neg\neg F$;
assume $G$ in
  both $F, G$;
left-and $F \wedge H$;
both $F, G \Rightarrow F \wedge G$
```

then the repetition becomes much easier to detect, and the **left-and** can be confidently eliminated. Similar remarks apply when an inference inside the body of an u.g. or an e.i. does not depend on the respective eigenvariable, as in:

```
pick-any $x$
  begin
    left-and $R(z) \wedge R(a)$;
    $D$
  end
```

In what follows we will be dealing with lists of deductions $[D_1, \ldots, D_n]$. We will use the letter Δ to denote such lists. For a non-empty list $\Delta = [D_1, \ldots, D_n]$, $n > 0$, we define $\overline{\Delta}$ as the thread $D_1; \ldots; D_n$. The following will come handy later:

Lemma 1.34 $\overline{\Delta_1 \oplus \Delta_2} = \overline{\Delta_1}; \overline{\Delta_2}$

We adopt the convention that when Δ is empty the expresssion $\overline{\Delta}; D$ stands for D.

The algorithm H in Figure 1.7 examines a right-linear thread $D = D_1; \ldots; D_n$ (we make the simplifying convention that we might have $n = 1$, in which case D_1 will not be composite, since we are assuming that D is right-linear) and pulls out every D_i that is not transitively dependent on a set of assumptions Φ or a set of variables V.

Each hoisted D_i is replaced in-place in D by the trivial deduction **claim** $\mathcal{C}(D_i)$. Specifically, $H(D, \Phi, V)$ returns a triple (D', Ψ, Δ), where

- D' is obtained from D by replacing every D_i that does not transitively depend on Φ or on V by $\mathcal{C}(D_i)$.
- $\Psi \supseteq \Phi$ is monotonically obtained from Φ by incorporating the conclusions of those deductions D_j that do depend (transitively) on Φ (or on V). This is essential in order to handle transitive dependence.
- Δ is a list $[D_{i_1}, \ldots, D_{i_k}]$, $1 \leq i_j \leq n$, $j = 1, \ldots, k \geq 0$, of those deductions that do not depend on Φ. The order is important for preserving dominance constraints: we have $i_a < i_b$ for $a < b$, since, e.g., D_5 and D_8 might not be dependent on Φ or on V, but D_8 might depend on D_5. Accordingly, Δ should respect the original ordering.

$$\begin{array}{rcl} H(D_1; D_2, \Phi, V) & = & \begin{array}{l} \mathit{let}\ (D_1', \Phi_1, \Delta_1) = H(D_1, \Phi, V) \\ \quad (D_2', \Phi_2, \Delta_2) = H(D_2, \Phi_1, V) \\ \mathit{in} \\ \quad (D_1'; D_2', \Phi_2, \Delta_1 \oplus \Delta_2) \end{array} \\ \\ H(D, \Phi, V) & = & [OA(D) \cap \Phi = \emptyset \text{ and } OV(D) \cap V = \emptyset] \rightarrow \\ & & (\textbf{claim}\ \mathcal{C}(D), \Phi, [D]) \diamondsuit (D, \Phi \cup \{\mathcal{C}(D)\}, []) \end{array}$$

Figure 1.7: The kernel of the hoisting algorithm.

As Theorem 1.40 will prove, the idea is that we will have $D \rightarrowtail \overline{\Delta}; D'$. The thread $D_1; \cdots ; D_n$ can be thought of as the body of a hypothetical deduction with hypothesis F, with Φ and V respectively as $\{F\}$ and $\emptyset$. Alternatively, we can think of $D_1; \cdots ; D_n$ as the body of an u.g. or an e.i. with eigenvariable x, in which case $\Phi = \emptyset$ and $V = \{x\}$. Then if $H(D_1; \ldots ; D_n, \Phi, V) = (D', \Psi, \Delta)$, D' will be the new body of the deduction, and the thread $\overline{\Delta}$ will comprise the hoisted deductions, with a dominance relation that respects the original ordering $1, \ldots, n$.

Lemma 1.35 *Let $H(D_1, \Phi_1, V) = (D_2, \Phi_2, \Delta)$. Then for all $D \in \Delta$,* (a) *$\Phi_1 \cap OA(D) = \emptyset$;* (b) *$V \cap OV(D) = \emptyset$; and* (c) *$D$ is not a composition.*

Proof: By induction on D_1. Suppose first that D_1 is not composite. There are two cases: either $OA(D_1) \cap \Phi_1 = \emptyset$ and $OV(D_1) \cap V = \emptyset$; or not. If not, then $\Delta = []$ so the result holds vacuously. Otherwise, $\Delta = [D_1]$, and the result holds by supposition. If D_1 is a composition $D_l; D_r$ then $\Delta = \Delta_l \oplus \Delta_r$, where $H(D_l, \Phi_1, V) = (D_l', \Phi_l, \Delta_l)$ and $H(D_r, \Phi_l, V) = (D_r', \Phi_r, \Delta_r)$. Inductively,

$$\forall D \in \Delta_l\ [\Phi_1 \cap OA(D) = \emptyset \text{ and } V \cap OV(D) = \emptyset] \tag{1.74}$$

and

$$\forall D \in \Delta_r\ [\Phi_l \cap OA(D) = \emptyset \text{ and } V \cap OV(D) = \emptyset \tag{1.75}$$

while every D in Δ_l and Δ_r is a non-composition. Since $\Phi_1 \subseteq \Phi_l$, (1.75) entails

$$\forall D \in \Delta_r, \Phi_1 \cap OA(D) = \emptyset \tag{1.76}$$

Parts (a) and (b) now follow from (1.74), (1.75), and (1.76), since $\Delta = \Delta_l \oplus \Delta_r$; while (c) follows directly from the inductive hypotheses. ■

We will also need the following four results, whose proofs are simple and omitted:

Lemma 1.36 *Let $H(D_1, \Phi_1, V) = (D_2, \Phi_2, \Delta)$. If D_1 is right-linear then D_2 is right-linear, and every $D \in \Delta$ is right-linear too.*

Lemma 1.37 *Let $(D', \Psi, \Delta) = H(D, \Phi, V)$. Then either*

1. *$D' = D$; or else*

2. *D' is a claim; or*

3. *D is a chain $D_1, \ldots, D_n, D_{n+1}$ and D' is a chain $D'_1, \ldots, D'_n, D'_{n+1}$, where for all i, either $D'_i = D_i$ or else D'_i is a claim.*

Lemma 1.38 *If $\mathcal{C}(D) \notin OA(D_i)$ for $i = 1, \ldots, n$ then*

$$D; D_1; \ldots; D_n; D' \rightarrowtail D_1; \ldots; D_n; D; D'.$$

Lemma 1.39 **claim** $F; D_1; \ldots; D_n; D \rightarrowtail D_1; \ldots; D_n;$ **claim** $F; D$.

Theorem 1.40 *If D is right-linear and $H(D, \Phi, V) = (D', \Psi, \Delta)$ then $D \rightarrowtail \overline{\Delta}; D'$.*
Proof: By induction on D. Suppose first that D is not a composition. Then either

$$OA(D) \cap \Phi = OV(D) \cap V = \emptyset$$

or not. If not, then $D' = D$ and $\Delta = []$, so the result is immediate. Otherwise, $D' =$ **claim** $\mathcal{C}(D)$ and $\Delta = [D]$, so again the result follows immediately. In contradistinction, suppose that D is a composition $D_1; D_2$. Then, letting

$$H(D_1, \Phi, V) = (D'_1, \Phi_1, \Delta_1) \tag{1.77}$$

and

$$H(D_2, \Phi_1, V) = (D'_2, \Phi_2, \Delta_2) \tag{1.78}$$

we have $D' = D'_1; D'_2$ and $\Delta = \Delta_1 \oplus \Delta_2$, so we have to show

$$D \rightarrowtail \overline{\Delta_1 \oplus \Delta_2}; D'_1; D'_2. \tag{1.79}$$

From (1.77), (1.78), and the inductive hypothesis, we have

$$D_1 \rightarrowtail \overline{\Delta_1}; D'_1 \tag{1.80}$$

and

$$D_2 \rightarrowtail \overline{\Delta_2}; D'_2 \tag{1.81}$$

Therefore,

$$D = D_1; D_2 \rightarrowtail \overline{\Delta_1}; D'_1; \overline{\Delta_2}; D'_2 \tag{1.82}$$

Since we are assuming that D is right-linear, D_1 cannot be composite, so we again distinguish two cases:

$$OA(D_1) \cap \Phi = OV(D_1) \cap V = \emptyset$$

or not. If not, then $D'_1 = D_1, \Phi_1 = \Phi \cup \{\mathcal{C}(D_1)\}$, and $\Delta_1 = []$. From (1.78) and Lemma 1.35 it follows that for every $D_x \in \Delta_2$, $\Phi_1 \cap OA(D_x) = \emptyset$, and since $\mathcal{C}(D_1) \in \Phi_1$, this means that $\mathcal{C}(D_1) \notin OA(D_x)$. Hence, by Lemma 1.38 (and remembering that $D'_1 = D_1$):

$$D'_1; \overline{\Delta_2}; D'_2 \rightarrowtail \overline{\Delta_2}; D'_1; D'_2$$

and thus

$$\overline{\Delta_1}; D'_1; \overline{\Delta_2}; D'_2 \rightarrowtail \overline{\Delta_1}; \overline{\Delta_2}; D'_1; D'_2 \tag{1.83}$$

By contrast, if $OA(D_1) \cap \Phi = OV(D_1) \cap V = \emptyset$ then $D'_1 = \mathcal{C}(D_1)$, so by Lemma 1.39 we have

$$D'_1; \overline{\Delta_2}; D'_2 \rightarrowtail \overline{\Delta_2}; D'_1; D'_2$$

and hence (1.83) follows again. Thus we have shown that in either case (1.83) holds, and since $\overline{\Delta_1 \oplus \Delta_2} = \overline{\Delta_1} \oplus \overline{\Delta_2}$, it now follows from (1.82), (1.83), and the transitivity of $\rightarrowtail$ that

$$D \rightarrowtail \overline{\Delta_1 \oplus \Delta_2}; D_1'; D_2'$$

which is precisely our goal (1.79). ■

Theorem 1.41 *If D is right-linear and $H(D, \{F\}, \emptyset) = (D', \Phi, \Delta)$ then*

(a) **assume** $F\ \ D \rightarrowtail \overline{\Delta};$ **assume** $F\ \ D'$*; and*

(b) **suppose-absurd** $F\ \ D \rightarrowtail \overline{\Delta};$ **suppose-absurd** $F\ \ D'$.

Proof: We prove (a); the proof of (b) is similar. We will first use induction on the list Δ to show that

$$\textbf{assume } F\ \ \overline{\Delta}; D' \rightarrowtail \overline{\Delta}; \textbf{assume } F\ \ D' \tag{1.84}$$

When Δ is the empty list this is immediate. For the inductive step, suppose that Δ is of the form $D_1{::}\Delta_1$. Lemma 1.35 gives $\{F\} \cap OA(D_1) = \emptyset$, so Theorem 1.19 yields

$$\textbf{assume } F\ \ D_1; \overline{\Delta_1}; D' \rightarrowtail D_1; \textbf{assume } F\ \ \overline{\Delta_1}; D' \tag{1.85}$$

Inductively, **assume** $F\ \ \overline{\Delta_1}; D' \rightarrowtail \overline{\Delta_1};$ **assume** $F\ \ D'$, so, by Lemma 1.17,

$$D_1; \textbf{assume } F\ \ \overline{\Delta_1}; D' \rightarrowtail D_1; \overline{\Delta_1}; \textbf{assume } F\ \ D' = \overline{\Delta}; \textbf{assume } F\ \ D' \tag{1.86}$$

The goal (1.84) now follows from (1.85), (1.86), and the transitivity of $\rightarrowtail$, and the induction is complete. Now by Theorem 1.40, $D \rightarrowtail \overline{\Delta}; D'$, hence Lemma 1.17 gives

$$\textbf{assume } F\ \ D \rightarrowtail \textbf{assume } F\ \ \overline{\Delta}; D' \tag{1.87}$$

(a) follows from (1.84), (1.87), and the transitivity of $\rightarrowtail$. ■

Theorem 1.42 *If D is right-linear and $H(D, \emptyset, \{x\}) = (D', \Phi, \Delta)$ then*

(a) **pick-any** $x\ \ D \rightarrowtail \overline{\Delta};$ **pick-any** $x\ \ D'$;

and (b) **pick-witness** x **for** $\exists\, y\,.\,F\ \ D \rightarrowtail \overline{\Delta};$ **pick-witness** x **for** $\exists\, y\,.\,F\ \ D'$.

Proof: For part (a), we will use induction on the list Δ to show that

$$\textbf{pick-any } x\ \ \overline{\Delta}; D' \rightarrowtail \overline{\Delta}; \textbf{pick-any } x\ \ D' \tag{1.88}$$

Since Theorem 1.40 gives $D \rightarrowtail \overline{\Delta}; D'$, Lemma 1.17 implies

$$\textbf{pick-any } x\ \ D \rightarrowtail \textbf{pick-any } x\ \ \overline{\Delta}; D' \tag{1.89}$$

Hence, once (1.88) is proven, part (a) will follow from it, (1.89), and the transitivity of $\rightarrowtail$.

When Δ is the empty list, (1.88) is immediate. When Δ is of the form $D_1{::}\Delta_1$, Lemma 1.35 implies $\{x\} \cap OV(D_1) = \emptyset$, so Theorem 1.20 yields

$$\textbf{pick-any } x\ \ D_1; \overline{\Delta_1}; D' \rightarrowtail D_1; \textbf{pick-any } x\ \ \overline{\Delta_1}; D' \tag{1.90}$$

By the inductive hypothesis,

$$\textbf{pick-any } x \ \overline{\Delta_1}; D' \rightarrowtail \overline{\Delta_1}; \textbf{pick-any } x \ D' \tag{1.91}$$

therefore, by Lemma 1.17,

$$D_1; \textbf{pick-any } x \ \overline{\Delta_1}; D' \rightarrowtail D_1; \overline{\Delta_1}; \textbf{pick-any } x \ D'$$

which is to say (by virtue of Lemma 1.34)

$$D_1; \textbf{pick-any } x \ \overline{\Delta_1}; D' \rightarrowtail \overline{\Delta}; \textbf{pick-any } x \ D' \tag{1.92}$$

Now (1.88) follows from (1.90), (1.92), and the transitivity of $\rightarrowtail$.

We use a similar technique for part (b). We note that (by Theorem 1.40) $D \rightarrowtail \overline{\Delta}; D'$ and hence (by Lemma 1.17):

$$\textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ D \rightarrowtail \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ \overline{\Delta}; D' \tag{1.93}$$

As we did above, we will use induction on Δ to show

$$\textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ \overline{\Delta}; D' \rightarrowtail \overline{\Delta}; \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ D' \rightarrowtail \tag{1.94}$$

When Δ is empty the result is immediate. When Δ is of the form $D_1{::}\Delta_1$, Lemma 1.35 tells us that $\{x\} \cap OV(D_1) = \emptyset$, so Theorem 1.20 gives

$$\begin{array}{c} \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ \overline{\Delta}; D' = \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ D_1; \overline{\Delta_1}; D' \\ \rightarrowtail \\ D_1; \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ \overline{\Delta_1}; D' \end{array} \tag{1.95}$$

Inductively,

$$\textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ \overline{\Delta_1}; D' \rightarrowtail \overline{\Delta_1}; \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ D' \tag{1.96}$$

hence, by Lemma 1.17,

$$D_1; \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ \overline{\Delta_1}; D' \rightarrowtail D_1; \overline{\Delta_1}; \textbf{pick-witness } x \textbf{ for } \exists\, y \,.\, F \ D' \tag{1.97}$$

and now (1.94) follows from (1.95), (1.97), and the transitivity of $\rightarrowtail$. The desired result finally follows from (1.93), (1.94), and the transitivity of $\rightarrowtail$. ■

As an illustration of the algorithm, let D be the deduction

1. **modus-ponens** $F \Rightarrow G \wedge H, F$;
2. **double-negation** $\neg\neg I$;
3. **left-and** $G \wedge H$;
4. **right-either** J, I;
5. **both** $G, J \vee I$

and consider the call $H(D, \{F\})$. Let D_1–D_5 refer to the deductions in lines 1–5, respectively. Since D is composite, the first clause of the algorithm will be chosen, so the first recursive call will be $H(D_1, \{F\})$, which, since D_1 is not composite and $OA(D_1) \cap \{F\} \neq \emptyset$, will yield the result

$(D_1, \{F, G \wedge H\}, [])$. The second recursive call is $H(D_2; D_3; D_4; D_5, \{F, G \wedge H\})$. This in turn gives rise to the recursive calls $H(D_2, \{F, G \wedge H\})$, which returns

$$(\textbf{claim } I, \{F, G \wedge H\}, [\textbf{double-negation } \neg\neg I]),$$

and $H(D_3; D_4; D_5, \{F, G \wedge H\})$. The latter will spawn $H(D_3, \{F, G \wedge H\})$, which will produce

$$(D_3, \{F, G \wedge H, G\}, []),$$

and $H(D_4; D_5, \{F, G \wedge H, G\})$. In the same fashion, the latter will invoke $H(D_4, \{F, G \wedge H, G\})$, which will return

$$(\textbf{claim } J \vee I, \{F, G \wedge H, G\}, [\textbf{right-either } J, I]),$$

and $H(D_5, \{F, G \wedge H, G\})$, which will produce $(D_5, \{F, G \wedge H, G, G \wedge (J \vee I)\}, [])$. Moving up the recursion tree, we eventually obtain the final result (D', Ψ, Δ), where D' is the deduction

1.**modus-ponens** $F \Rightarrow G \wedge H, F$;
2.**claim** I;
3.**left-and** $G \wedge H$;
4.**claim** $J \vee I$;
5.**both** $G, J \vee I$

while $\Psi = \{F, G \wedge H, G, G \wedge (J \vee I)\}$ and $\Delta = [\textbf{double-negation } \neg\neg I, \textbf{right-either } J, I]$. Thus $\overline{\Delta}; D'$ is the deduction

double-negation $\neg\neg I$;
right-either J, I;

---

modus-ponens $F \Rightarrow G \wedge H, F$;
claim I;
left-and $G \wedge H$;
claim $J \vee I$;
both $G, J \vee I$

The horizontal line demarcates the hoisted inferences from D'.

If D were the body of a hypothetical deduction with hypothesis F, then the result of the hoisting would be $\overline{\Delta}; \textbf{assume } F \;\; D'$, namely,

```
double-negation ¬¬I;
right-either J, I;
assume F
  begin
    modus-ponens F ⇒ G ∧ H, F;
    claim I;
    left-and G ∧ H;
    claim J ∨ I;
    both G, J ∨ I
  end
```

A subsequent contracting transformation to remove claims (algorithm $\mathfrak{C}$) would result in

```
**double-negation** $\neg\neg I$;
**right-either** $J, I$;
**assume** $F$
  **begin**
    **modus-ponens** $F \Rightarrow G \wedge H, F$;
    **left-and** $G \wedge H$;
    **both** $G, J \vee I$
  **end**
```

The hoisting algorithm should be applied to every hypothetical deduction, every u.g., and every e.i. inside a given D. This must be done in stages and in a bottom-up direction in order for hoisted inferences to "bubble" as far up as possible (to maximize their scope). Specifically, let D be a given deduction. The hoisting will proceed in stages $i = 1, \ldots, n, \ldots$, where we begin with $D_1 = D$. At each stage i we replace certain *candidate* subdeductions of D_i by new deductions, and the result we obtain from these replacements becomes D_{i+1}. We keep going until we reach a fixed point, i.e., until $D_{i+1} = D_i$.

At each point in the process every hypothetical deduction (as well as every u.g. and every e.i.) inside D_i is either *marked*, indicating that its body has already been processed, or unmarked. An invariant we will maintain throughout is that a marked subdeduction will never contain unmarked deductions. This will be enforced by the way in which we will be choosing our candidates, and will ensure that hoisting proceeds in a bottom-up direction. Initially, every hypothetical deduction as well as every u.g. and e.i. inside $D_1 = D$ is unmarked. On stage i, an unmarked subdeduction of D_i is a candidate for hoisting iff it is as deep as possible, i.e., iff it does not itself contain any unmarked subdeductions. For each such candidate D_c of the form **assume** $F \;\; D_b$ or **suppose-absurd** $F \;\; D_b$ occurring in position $u \in Dom(D_i)$, we compute $(D'_b, \Psi, \Delta) = H(D_b, \{F\}, \emptyset)$, and we replace D_c in position u of D_i by $\overline{\Delta}; \underline{\textbf{assume}}\; F \;\; D'_b$ (or $\overline{\Delta}; \underline{\textbf{suppose-absurd}}\; F \;\; D'_b$, respectively), where the **assume** (or **suppose-absurd**) is now marked to indicate that its body D'_b has been combed bottom-up and we are thus finished with it—it can no longer serve as a candidate. Likewise, for each candidate D_c of the form **pick-any** $x \;\; D_b$ or **pick-witness** x **for** $\exists\, y\,.\, F \;\; D_b$ occuring in position $u \in Dom(D_i)$, we compute $(D'_b, \Psi, \Delta) = H(D_b, \emptyset, \{x\})$, and we replace D_c in u by $\overline{\Delta}; \underline{\textbf{pick-any}}\; x \;\; D'_b$ (or $\overline{\Delta}; \underline{\textbf{pick-witness}}\; x$ **for** $\exists\, y\,.\, F \;\; D'_b$, respectively). The deduction we obtain from D_i by carrying out these replacements becomes D_{i+1}. One pitfall to be avoided: the replacements might introduce left-linear subdeductions in D_{i+1}. Algorithm H, however, expects its argument to be right-linear, so after the replacements are performed we need to apply $\mathfrak{RL}$ to D_{i+1} before continuing on to the next stage.

We will say that a deduction D is *fully marked* iff every **assume**, **suppose-absurd**, **pick-any**, and **pick-witness** inside D is marked. Algorithm *Hoist* below replaces every candidate subdeduction of a given D in the manner discussed above and marks the processed subdeduction:

```
$Hoist(D) = match\;\; D$
            **assume** $F \;\; D_b \rightarrow$
              *Is* $D_b$ *fully marked?* $\rightarrow$
                *let* $(D'_b, \_, \Delta) = H(D_b, \{F\}, \emptyset)$
                *in*
                  $\overline{\Delta}; \underline{\textbf{assume}}\; F \;\; D'_b \diamondsuit$
                **assume** $F \;\; Hoist(D_b)$
            **suppose-absurd** $F \;\; D_b \rightarrow$
              *Is* $D_b$ *fully marked?* $\rightarrow$
```

```
        $let\ (D_b', \_, \Delta) = H(D_b, \{F\}, \emptyset)$
        $in$
          $\overline{\Delta}; \underline{\textbf{suppose-absurd}}\ F\ D_b' \diamondsuit$
        $\textbf{suppose-absurd}\ F\ Hoist(D_b)$
$\textbf{pick-any}\ x\ D_b \rightarrow$
    $Is\ D_b\ fully\ marked? \rightarrow$
        $let\ (D_b', \_, \Delta) = H(D_b, \emptyset, \{x\})$
        $in$
          $\overline{\Delta}; \underline{\textbf{pick-any}}\ x\ D_b' \diamondsuit$
        $\textbf{pick-any}\ x\ Hoist(D_b)$
$\textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x\,.\, F\ D_b \rightarrow$
    $Is\ D_b\ fully\ marked? \rightarrow$
        $let\ (D_b', \_, \Delta) = H(D_b, \emptyset, \{w\})$
        $in$
          $\overline{\Delta}; \underline{\textbf{pick-witness}}\ w\ \textbf{for}\ \exists\, x\,.\, F\ D_b' \diamondsuit$
        $\textbf{pick-witness}\ w\ \textbf{for}\ \exists\, x\,.\, F\ Hoist(D_b)$
$D_1; D_2 \rightarrow Hoist(D_1); Hoist(D_2)$
$D \rightarrow D$
```

Using Theorem 1.41, Theorem 1.42, and Lemma 1.17, a straightforward induction on D will prove:

Theorem 1.43 *If D is right-linear then $D \rightarrowtail Hoist(D)$.*

We can now formulate our scope-maximization transformation as:

$$\mathfrak{MS}\ D = fp\ (\mathfrak{RL} \cdot Hoist)\ (\mathfrak{RL}\ D)$$

where *fp* is as defined in Section 1.1. That $\mathfrak{MS}$ always terminates follows from the fact that *Hoist* does not introduce any additional hypothetical deductions, universal generalizations, or existential instantiations; and either outputs the same result unchanged or a deduction with at least one more subdeduction marked. Since any deduction only has a finite number of subdeductions, this means that $\mathfrak{MS}$ will eventually converge to a fixed point. Further, $D \rightarrowtail \mathfrak{MS}(D)$ follows from the corresponding property of $\mathfrak{RL}$, from Theorem 1.43, and from the transitivity of the $\rightarrowtail$ relation. The right-linearity of the result follows directly from the definition of $\mathfrak{MS}$. We summarize:

Theorem 1.44 (a) $\mathfrak{MS}$ *always terminates;* (b) $\mathfrak{MS}(D)$ *is right-linear;* (c) $D \rightarrowtail \mathfrak{MS}(D)$.

We close by addressing the question of whether this restructuring algorithm might ever increase the size of a deduction. Since $\mathfrak{MS}$ works by repeatedly applying the composition of *Hoist* with $\mathfrak{RL}$, it will follow that $\mathfrak{MS}$ preserves the size of its argument if both $\mathfrak{RL}$ and *Hoist* do. This is readily verified for $\mathfrak{RL}$; we have $SZ(\mathfrak{RL}(D)) = SZ(D)$ for all D. Consider now the hoisting transformation H, which is the core of *Hoist*. When *Hoist* applies H to the body of a hypothetical deduction (or u.g. or e.i.), say **assume** $F\ D_b$, thereby obtaining a new deduction $\overline{\Delta}; \textbf{assume}\ F\ D_b'$, the new part $\overline{\Delta}$ is obtained by trimming down the body D_b, so, intuitively, we should have $SZ(D_b) = SZ(D_b') + SZ(\overline{\Delta})$. But that will not always be true because the new body D_b' might contain some claims where the hoisted deductions used to be, and those claims will cause the size of the result to be somewhat larger than that of the original. However, most such claims will be subsequently removed by the claim-elimination algorithm presented earlier, and this will rebalance the final size—even in the worst-case scenario in which the hoisting did not expose any new contraction opportunities. This is evinced by Lemma 1.37: claims inserted in D_b' in non-tail positions will be eliminated by $\mathfrak{C}$, as guaranteed by Lemma 1.27.

There is only one exception, again as prescribed by Lemma 1.37: when the new body D'_b is a chain of the form $D_1; \ldots; D_n$, $n \geq 1$, and the last element of the thread, D_n, is a newly inserted claim. Such a claim, being in a tail position, will not be removed by the claim-elimination algorithm. As a simple example, consider

$$D = \textbf{assume } F \ \textbf{ double-negation } \neg\neg G. \tag{1.98}$$

Here the body does not depend on the hypothesis F, so hoisting it outside results in the deduction

$$\textbf{double-negation } \neg\neg G; \textbf{assume } F \ \textbf{ claim } G$$

which is slightly larger than the original (1.98). But this minor wrinkle is easily rectified using **cond** (or **neg** and **gen** in the case of **suppose-absurd** and **pick-any**, respectively). Specifically, by the way H is defined, if a trivial deduction **claim** G is inserted in the last slot of D'_b (viewing D'_b as a chain of length $n \geq 1$), then the last element of the produced list Δ will be a deduction with conclusion G. Therefore, in that case, instead of producing $\overline{\Delta}; \textbf{assume } F \ D'_b$ we may simply output $\overline{\Delta}; \textbf{cond } F, \mathcal{C}(\overline{\Delta})$; or, in the case of proofs by contradiction, $\overline{\Delta}; \textbf{neg } F$. Accordingly, we modify *Hoist* by replacing the line $\overline{\Delta}; \underline{\textbf{assume}} \ F \ D'_b$, by

$$\Delta \neq [] \ \ \textit{and} \ \ \mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)\,? \to \overline{\Delta}; \textbf{cond } F, \mathcal{C}(D'_b) \diamondsuit \overline{\Delta}; \underline{\textbf{assume}} \ F \ D'_b,$$

Note that "$\Delta \neq []$ and $\mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)$" is not a sufficient condition for guaranteeing that the hoisting algorithm has inserted a trivial claim in the tail position of D'_b (although it is necessary). The transformation is safe nevertheless because the identity $\mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)$ ensures that

$$\overline{\Delta}; \underline{\textbf{assume}} \ F \ D'_b \rightarrowtail \overline{\Delta}; \textbf{cond } F, \mathcal{C}(D'_b)$$

Likewise, the line $\overline{\Delta}; \underline{\textbf{suppose-absurd}} \ F \ D'_b$ is replaced by

$$\Delta \neq [] \ \ \textit{and} \ \ \mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)\,? \to \overline{\Delta}; \textbf{neg } F \diamondsuit \overline{\Delta}; \underline{\textbf{suppose-absurd}} \ F \ D'_b$$

which is safe because if $\mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)$ then $\mathcal{C}(\overline{\Delta}) = \textbf{false}$. The line $\overline{\Delta}; \underline{\textbf{pick-any}} \ x \ D'_b$ is replaced by

$$\Delta \neq [] \ \ \textit{and} \ \ \mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)\,? \to \overline{\Delta}; \textbf{gen}_x \ \mathcal{C}(D'_b) \diamondsuit \overline{\Delta}; \underline{\textbf{pick-any}} \ x \ D'_b$$

which is safe because $OV(D) \cap \{x\} = \emptyset$ for all $D \in \Delta$ (Lemma 1.35), which means that

$$x \notin FV(\mathcal{C}(\overline{\Delta})) = FV(\mathcal{C}(D'_b))$$

Finally, we replace the line $\overline{\Delta}; \underline{\textbf{pick-witness}} \ w \ \textbf{for} \ \exists\, x \,.\, F \ D'_b$ by

$$\Delta \neq [] \ \ \textit{and} \ \ \mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'_b)\,? \to \overline{\Delta} \diamondsuit \overline{\Delta}; \underline{\textbf{pick-witness}} \ w \ \textbf{for} \ \exists\, x \,.\, F \ D'_b$$

which is safe because $\mathcal{C}(\textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \ D'_b) = \mathcal{C}(D'_b)$. It is readily verified that these changes do not affect Theorem 1.44, while ensuring that all claims inserted by H will be subsequently elimimated (during the contraction phase).

1.5.2 Global transformations of hypothetical deductions

The hoisting algorithm is a focused, local transformation: we delve inside a given deduction D and work on subdeductions of the form **assume** $F \ D_b$ or **suppose-absurd** $F \ D_b$, taking into account only the hypothesis F and the body D_b. We do not utilize any knowledge from a wider context.

More intelligent transformations become possible if we look at the big picture, namely, at how F and D_b relate to other parts of the enclosing deduction D. In this section we will present three such transformations, $\mathfrak{A}_1$, $\mathfrak{A}_2$, and $\mathfrak{A}_3$. All three of them perform a global analysis of a given deduction D and replace every hypothetical subdeduction D' of it by some other deduction D'' (where we might have $D'' = D'$). These transformations expect their input deductions to have been processed by $\mathfrak{MS}$, but their output deductions might contain left-linear compositions or hoisting possibilities that were not previously visible. It is for this reason that their composition must be interleaved with the scope-maximization procedure $\mathfrak{MS}$, as specified in (1.3) (or (1.4)).

The first transformation, $\mathfrak{A}_1$, targets every hypothetical subdeduction of D of the form $D' =$ **assume** $F\ \ D_b$ whose hypothesis F is an open assumption of D, i.e., such that $F \in OA(D)$. Clearly, D can only be successfully evaluated in an assumption base that contains F (Theorem 1.11). But if we must evaluate D in an assumption base that contains F, then there is no need to hide D_b behind that hypothesis; we can pull it outside. Accordingly, this analysis will replace D' by the composition $D'' = D_b;$ **cond** $F, \mathcal{C}(D_b)$. Thus the final conclusion is unaffected (it is still the conditional $F \Rightarrow \mathcal{C}(D_b)$), but the scope of D_b is enlarged. An analogous transformation is performed for proofs by contradiction. Specifically, we define:

```
$\mathfrak{A}_1(D) = T(D)$
where
$T($**assume** $F\ \ D_b) =$
    $let\ D'_b = T(D_b)$
    $in$
      $F \in OA(D)\ ? \to D'_b;$ **cond** $F, \mathcal{C}(D'_b) \diamondsuit$ **assume** $F\ \ D'_b$
$T($**suppose-absurd** $F\ \ D_b) =$
    $let\ D'_b = T(D_b)$
    $in$
      $F \in OA(D)\ ? \to D'_b;$ **neg** $F \diamondsuit$ **suppose-absurd** $F\ \ D'_b$
$T(D_l; D_r) = T(D_l); T(D_r)$
$T($**pick-any** $x\ \ D_b) =$ **pick-any** $x\ \ T(D_b)$
$T($**pick-witness** $w$ **for** $\exists\, x\,.\,F\ \ D_b) =$ **pick-witness** $w$ **for** $\exists\, x\,.\,F\ \ T(D_b)$
$T(D) = D$
```

Note that we first process D_b recursively and then pull it out, since D_b might itself contain hypothetical deductions with open assumptions as hypotheses. For example, if D is the deduction

```
assume F in
 begin
  both F, F;
  assume G in
    both G, F ∧ F
 end;
both F, G;
both F ∧ G, F ⇒ G ⇒ G ∧ F ∧ F
```

where both conditional deductions have open assumptions as hypotheses (F and G) then $\mathfrak{A}_1(D)$ will be:

```
begin
  begin
```

```
    **both** $F, F$;
    **both** $G, F \wedge F$;
    **cond** $G, G \wedge F \wedge F$
  **end**;
  **cond** $F, G \Rightarrow G \wedge F \wedge F$
**end**;
**both** $F, G$;
**both** $F \wedge G, F \Rightarrow G \Rightarrow G \wedge F \wedge F$
```

Observe that the output deduction is heavily skewed to the left (when viewed as a tree). After a pass of the right-linearization algorithm, we will obtain the following:

```
**both** $F, F$;
**both** $G, F \wedge F$;
**cond** $G, G \wedge F \wedge F$;
**cond** $F, G \Rightarrow G \wedge F \wedge F$;
**both** $F, G$;
**both** $F \wedge G, F \Rightarrow G \Rightarrow G \wedge F \wedge F$
```

A straightforward induction will show:

Lemma 1.45 $\mathfrak{A}_1$ *terminates. Moreover,* $D \rightarrowtail \mathfrak{A}_1(D)$ *and* $SZ(\mathfrak{A}_1(D)) \leq SZ(D)$.

The two remaining transformations turn not on whether the supposition of a hypothetical deduction is an open assumption, but on whether it is deduced at some prior or subsequent point. For the second transformation, $\mathfrak{A}_2$, suppose that during our evaluation of D we come to a conditional subdeduction $D' =$ **assume** $F\ D_b$ whose hypothesis F either has already been established or else has already been hypothetically postulated (e.g., D' is itself nested within an **assume** with hypothesis F). Then we may again pull D_b out, replacing D' by the composition $D'' = D_b;$ **cond** $F, \mathcal{C}(D_b)$. (More precisely, just as in $\mathfrak{A}_1$, we first have to process D_b recursively before hoisting it.) A similar transformation is possible for proofs by contradiction.

To motivate this transformation, consider the following deduction:

```
**left-and** $\neg\neg F \wedge H$;
**assume** $\neg\neg F$
 **begin**
   **dn** $\neg\neg F$;
   **both** $F, G$
 **end**;
**modus-ponens** $\neg\neg F \Rightarrow F \wedge G, \neg\neg F$
```

This deduction illustrates one of the detours we discussed earlier, whereby F_2 is derived by first inferring F_1, then $F_1 \Rightarrow F_2$, and then using **modus-ponens** on $F_1 \Rightarrow F_2$ and F_1. The detour arises because the hypothesis F_1 is in fact deducible, and hence there is no need for the implication $F_1 \Rightarrow F_2$ and the **modus-ponens**. We can simply deduce F_1 and then directly perform the reasoning of the body of the hypothetical deduction. Thus we arrive at the following algorithm:

```
$\mathfrak{A}_2(D) = T(D, \emptyset)$
where
$T(D, \Phi) =$ *match* $D$
                **assume** $F\ D_b \rightarrow$
```

```
  $F \in \Phi \rightarrow$ $let$ $D_b' = T(D_b, \Phi)$
                 $in$
                   $D_b'$; **cond** $F, \mathcal{C}(D_b')$ ◇
                 **assume** $F$ $T(D_b, \Phi \cup \{F\})$
**suppose-absurd** $F$ $D_b \rightarrow$
  $F \in \Phi \rightarrow$ $let$ $D_b' = T(D_b, \Phi)$
                 $in$
                   $D_b'$; **neg** $F$ ◇
                 **suppose-absurd** $F$ $T(D_b, \Phi \cup \{F\})$
**pick-any** $x$ $D_b \rightarrow$ **pick-any** $x$ $T(D_b, \Phi)$
**pick-witness** $w$ **for** $\exists\, x \,.\, F$ $D_b \rightarrow$ **pick-witness** $w$ **for** $\exists\, x \,.\, F$ $T(D_b, \Phi \cup \{\{x \mapsto w\}\, F\})$
$D_1; D_2 \rightarrow$ $let$ $D_1' = T(D_1, \Phi)$
             $in$
               $D_1'; T(D_2, \Phi \cup \{\mathcal{C}(D_1)\})$
$D \rightarrow D$
```

Applying this algorithm to the preceding example would yield:

```
**left-and** $\neg\neg F \wedge H$;
**begin**
  **begin**
    **dn** $\neg\neg F$;
    **both** $F, G$
  **end**;
  **cond** $\neg\neg F, F \wedge G$
**end**;
**modus-ponens** $\neg\neg F \Rightarrow F \wedge G, \neg\neg F$
```

Passing this on to the scope-maximization procedure and then to the contraction algorithm will produce the final result:

```
**left-and** $\neg\neg F \wedge H$;
**dn** $\neg\neg F$;
**both** $F, G$
```

We can establish the soundness of this algorithm in two steps. First, we can prove by induction on D that if $\beta \vdash T(D, \Phi) \rightsquigarrow G$ then $\beta \cup \{F\} \vdash T(D, \Phi \cup \{F\}) \rightsquigarrow G$. Then, using this lemma, an induction on D will show that $D \rightarrowtail T(D, \emptyset)$, which will prove that $D \rightarrowtail \mathfrak{A}_2(D)$ for all D. However, it is readily observed that $\mathfrak{A}_1$ and $\mathfrak{A}_2$ can be combined in one pass simply by calling $T(D, OA(D))$. In other words, applying the composition of $\mathfrak{A}_1$ with $\mathfrak{A}_2$ to some D produces the same result as $T(D, OA(D))$:

$$\mathfrak{A}_1 \cdot \mathfrak{A}_2 = \lambda\, D \,.\, T(D, OA(D))$$

Accordingly, we define an algorithm $\mathfrak{A}$ as $\mathfrak{A}(D) = T(D, OA(D))$. In our implementation, instead of first calling $\mathfrak{A}_2$, then $\mathfrak{MS}$, and then $\mathfrak{A}_1$, as prescribed by (1.4), we simply call $\mathfrak{A}$ once. (For exposition purposes, we choose to keep the presentations of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ distinct.) The following lemma will prove useful in showing the soundness of $\mathfrak{A}$.

Lemma 1.46 *If $\beta \vdash D \rightsquigarrow G$ then $\beta \vdash T(D, \beta) \rightsquigarrow G$.*

Proof: By induction on the structure of D. When D atomic the result is immediate. When D is a conditional deduction **assume** $F\ \ D_b$, the assumption $\beta \vdash D \leadsto G$ means that

$$\beta \cup \{F\} \vdash D_b \leadsto H \tag{1.99}$$

where $G = F \Rightarrow H$. Inductively, (1.99) gives

$$\beta \cup \{F\} \vdash T(D_b, \beta \cup \{F\}) \leadsto H \tag{1.100}$$

We now distinguish two cases:

1. $F \notin \Phi$: Then $T(D, \beta) =$ **assume** $F\ \ T(D_b, \beta \cup \{F\})$, and (1.100) yields the desired

$$\beta \vdash \textbf{assume } F\ \ T(D_b, \beta \cup \{F\}) \leadsto F \Rightarrow H = G$$

2. $F \in \Phi$: Then

$$T(D, \beta) = T(D_b, \beta); \textbf{cond } F, \mathcal{C}(T(D_b, \beta)) \tag{1.101}$$

   Since $\beta \cup \{F\} = \beta$, (1.100) becomes $\beta \vdash T(D_b, \beta) \leadsto H$ and hence, by (1.101), we get

$$\beta \vdash T(D, \beta) \leadsto F \Rightarrow H = G$$

When D is of the form **suppose-absurd** $F\ \ D_b$, we have $\beta \cup \{F\} \vdash D_b \leadsto$ **false**, where $G = \neg F$. By the inductive hypothesis,

$$\beta \cup \{F\} \vdash T(D_b, \beta \cup \{F\}) \leadsto \textbf{false} \tag{1.102}$$

and we again distinguish two cases:

1. $F \notin \beta$: In that case $T(D, \beta) =$ **suppose-absurd** $F\ \ T(D_b, \beta \cup \{F\})$ and (1.102) yields the desired

$$\beta \vdash \textbf{suppose-absurd } F\ \ T(D_b, \beta \cup \{F\}) \leadsto \neg F = G$$

2. $F \in \beta$: In that case $\beta \cup \{F\} = \beta$, so (1.102) becomes $\beta \vdash T(D_b, \beta) \leadsto$ **false**, which implies $\beta \vdash T(D_b, \beta); \textbf{neg } F \leadsto \neg F$, i.e., $\beta \vdash T(D, \beta) \leadsto G$.

Next, suppose that D is a composition $D_1; D_2$. The assumption $\beta \vdash D \leadsto G$ then means that $\beta \vdash D_1 \leadsto F$ and $\beta \cup \{F\} \vdash D_2 \leadsto G$, where $F = \mathcal{C}(D_1)$. Inductively, $\beta \vdash T(D_1, \beta) \leadsto F$ and

$$\beta \cup \{F\} \vdash T(D_2, \beta \cup \{F\}) \leadsto G$$

Therefore, $\beta \vdash T(D_1, \beta); T(D_2, \beta \cup \{F\}) \leadsto G$, i.e., $\beta \vdash T(D_1; D_2, \beta) \leadsto G$.

When D is a u.g. **pick-any** $x\ \ D_b$, we have $T(D, \beta) =$ **pick-any** $x\ \ T(D_b, \beta)$. The assumption $\beta \vdash D \leadsto G$ means that

$$\beta \vdash \{x \mapsto z\}\, D_b \leadsto F \tag{1.103}$$

for some fresh z, where $G = \forall\, z\, .\, F$. Now $\{z \mapsto x\}$ is safe for $\{x \mapsto z\}\, D_b$ (we can always rename D to ensure this), hence (1.103) and Theorem 1.6 give

$$\{z \mapsto x\}\, \beta \vdash \{z \mapsto x\}\, \{x \mapsto z\}\, D_b \leadsto \{z \mapsto x\}\, F \tag{1.104}$$

Since z does not occur in β, we have $\{z \mapsto x\}\, \beta = \beta$; and since z does not occur in D_b, Lemma 1.4 gives $\{z \mapsto x\}\, \{x \mapsto z\}\, D_b = D_b$, so (1.104) becomes

$$\beta \vdash D_b \rightsquigarrow \{z \mapsto x\}\, F$$

Inductively,

$$\beta \vdash T(D_b, \beta) \rightsquigarrow \{z \mapsto x\}\, F \tag{1.105}$$

The substitution $\{x \mapsto z\}$ is safe for $T(D_b, \beta)$ (owing to z's freshness), hence, by (1.105) and Theorem 1.6, we get

$$\{x \mapsto z\}\, \beta \vdash \{x \mapsto z\}\, T(D_b, \beta) \rightsquigarrow \{x \mapsto z\}\, \{z \mapsto x\}\, F \tag{1.106}$$

Without loss of generality, we may assume that x does not occur in β (again, this can be ensured by renaming D), hence $\{x \mapsto z\}\, \beta = \beta$. Moreover, x does not occur in $\{x \mapsto z\}\, D_b$, hence Lemma 1.8 and Theorem 1.9 entail that x does not occur in F. Accordingly, Lemma 1.3 yields

$$\{x \mapsto z\}\, \{z \mapsto x\}\, F = F$$

Therefore, (1.106) becomes $\beta \vdash \{x \mapsto z\}\, T(D_b, \beta) \rightsquigarrow F$ and thus

$$\beta \vdash \textbf{pick-any}\ x\ \ T(D_b, \beta) \rightsquigarrow \forall\, z \,.\, F = G$$

which is to say $\beta \vdash T(D, \beta) \rightsquigarrow G$.

Finally, suppose that D is of the form **pick-witness** w **for** $\exists\, x \,.\, F\ \ D_b$. The assumption $\beta \vdash D \rightsquigarrow G$ entails

$$\exists\, x \,.\, F \in \beta \tag{1.107}$$

and that

$$\beta \cup \{\{x \mapsto z\}\, F\} \vdash \{w \mapsto z\}\, D_b \rightsquigarrow G \tag{1.108}$$

for some fresh z such that $z \notin FV(G)$. Without loss of generality, we may assume that w does not occur in $\{w \mapsto z\}\, D_b$, which means that $\{z \mapsto w\}$ is safe for $\{w \mapsto z\}\, D_b$, so that, by Theorem 1.6 and (1.108),

$$\{z \mapsto w\}\, \beta \cup \{\{z \mapsto w\}\, \{x \mapsto z\}\, F\} \vdash \{z \mapsto w\}\, \{w \mapsto z\}\, D_b \rightsquigarrow \{z \mapsto w\}\, G$$

Since z does not occur in β, F, D_b, or G, the above becomes (by Lemma 1.3 and Lemma 1.4):

$$\beta \cup \{\{x \mapsto w\}\, F\} \vdash D_b \rightsquigarrow G \tag{1.109}$$

The inductive hypothesis transforms (1.109) into

$$\beta \cup \{\{x \mapsto w\}\, F\} \vdash T(D_b, \beta \cup \{\{x \mapsto w\}\, F\}) \rightsquigarrow G \tag{1.110}$$

Now $\{w \mapsto z\}$ is safe for $T(D_b, \beta \cup \{\{x \mapsto w\}\, F\})$, hence (1.110) and Theorem 1.6 imply

$$\{w \mapsto z\}\, \beta \cup \{\{w \mapsto z\}\, \{x \mapsto w\}\, F\} \vdash \{w \mapsto z\}\, T(D_b, \beta \cup \{\{x \mapsto w\}\, F\}) \rightsquigarrow \{w \mapsto z\}\, G \tag{1.111}$$

Without loss of generality, we may assume that w does not occur in β or in F, so that $\{w \mapsto z\}\, \beta = \beta$ and, by Lemma 1.3, $\{w \mapsto z\}\, \{x \mapsto w\}\, F = F$. Moreover, w does not occur in $\{w \mapsto z\}\, D_b$, hence by (1.108), Lemma 1.8 and Theorem 1.9, we conclude that w does not occur in G, and hence $\{w \mapsto z\}\, G = G$. Accordingly, (1.111) becomes

$$\beta \cup \{\{x \mapsto z\}\, F\} \vdash \{w \mapsto z\}\, T(D_b, \beta \cup \{\{x \mapsto w\}\, F\}) \rightsquigarrow G$$

which is to say, by virtue of (1.107),

$$\beta \vdash \textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \;\; T(D_b, \beta \cup \{\{x \mapsto w\}\, F\}) \leadsto G$$

i.e., $\beta \vdash T(D, \beta) \leadsto G$. This completes the case analysis and the inductive argument. ■

Theorem 1.47 *$\mathfrak{A}$ terminates; $D \rightarrowtail \mathfrak{A}(D)$; and $SZ(\mathfrak{A}(D)) \leq SZ(D)$.*

Proof: Termination is obvious. That the size of $\mathfrak{A}(D)$ is never more than the size of D also follows by a straightforward induction on D. Finally, to prove $D \rightarrowtail \mathfrak{A}(D)$, suppose that $\beta \vdash D \leadsto F$ for some β. By Theorem 1.11, we must have

$$\beta \supseteq OA(D). \tag{1.112}$$

By the same result, $OA(D) \vdash D \leadsto F$, hence, by Lemma 1.46, $OA(D) \vdash T(D, OA(D)) \leadsto F$, i.e.,

$$OA(D) \vdash \mathfrak{A}(D) \leadsto F.$$

Therefore, by (1.112) and dilution we get $\beta \vdash \mathfrak{A}(D) \leadsto F$, which shows that $D \rightarrowtail \mathfrak{A}(D)$. ■

The final transformation, $\mathfrak{A}_3$, determines whether the hypothesis F of a conditional deduction $D' = \textbf{assume } F \;\; D_b$ is deduced at a later point, or, more precisely, whether it is deduced somewhere within a deduction dominated by D', as in the following picture:

$$\begin{array}{cl} \vdots & \\ D' = \textbf{assume } F \;\; D_b; & \\ \vdots & \\ D''; & \qquad \text{(Deduces } F\text{)} \\ \vdots & \end{array}$$

This can lead to the following variant of the detour we discussed earlier:

```
(1) assume $\neg\neg F$
      begin
        dn $\neg\neg F$;
        both $F, G$
      end;
(2) left-and $\neg\neg F \wedge H$;
(3) modus-ponens $\neg\neg F \Rightarrow F \wedge G, \neg\neg F$
```

However, unlike the cases discussed in connection with $\mathfrak{A}_2$ and $\mathfrak{A}_1$, here we cannot hoist the body of (1) above the **assume** (and replace the **assume** by an application of **cond**), because the said body strictly uses the hypothesis $\neg\neg F$, which is neither an open assumption of the overall deduction nor deduced prior to its hypothetical postulation in (1). Rather, $\neg\neg F$ is deduced *after* the conditional deduction where it appears as a hypothesis. What we will do instead is reduce this case to one that can be handled by the simple hoisting method of algorithm $\mathfrak{A}$. We can do that by "bubbling up" the deduction which derives the hypothesis in question until it precedes the hypothetical deduction, at which point $\mathfrak{A}$ will be able to perform as usual. Specifically, we define:

```
1. $\mathfrak{A}_3(\textbf{assume } F \ D_b) = \textbf{assume } F \ \mathfrak{A}_3(D_b)$
2. $\mathfrak{A}_3(\textbf{suppose-absurd } F \ D_b) = \textbf{suppose-absurd } F \ \mathfrak{A}_3(D_b)$
3. $\mathfrak{A}_3((\textbf{assume } F \ D_b); D) =$
4.     $let \ (D'_b, D') = (\mathfrak{A}_3(D_b), \mathfrak{A}_3(D))$
5.         $(D'', \_, \Delta) = H(D', \{F \Rightarrow \mathcal{C}(D'_b)\}, \emptyset)$
6.     $in$
7.         $\overline{\Delta}; \textbf{assume } F \ D'_b; D''$
8. $\mathfrak{A}_3((\textbf{suppose-absurd } F \ D_b); D) =$
9.     $let \ (D'_b, D') = (\mathfrak{A}_3(D_b), \mathfrak{A}_3(D))$
10.        $(D'', \_, \Delta) = H(D', \{\neg F\}, \emptyset)$
11.    $in$
12.        $\overline{\Delta}; \textbf{suppose-absurd } F \ D'_b; D''$
13. $\mathfrak{A}_3(\textbf{pick-any } x \ D_b) = \textbf{pick-any } x \ \mathfrak{A}_3(D_b)$
14. $\mathfrak{A}_3(\textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \ D_b) = \textbf{pick-witness } w \textbf{ for } \exists\, x \,.\, F \ \mathfrak{A}_3(D_b)$
15. $\mathfrak{A}_3(D_1; D_2) = \mathfrak{A}_3(D_1); \mathfrak{A}_3(D_2)$
16. $\mathfrak{A}_3(D) = D$
```

Applying this algorithm to the deduction above yields:

left-and $\neg\neg F \wedge H$;
assume $\neg\neg F$
  begin
    dn $\neg\neg F$;
    both F, G
  end;
claim $\neg\neg F$;
modus-ponens $\neg\neg F \Rightarrow F \wedge G, \neg\neg F$

which will be readily handled by $\mathfrak{A}$. In particular, after applying $\mathfrak{A}$ to the above deduction, followed by $\mathfrak{MS}$ and *contract*, we obtain the final result:

left-and $\neg\neg F \wedge H$;
dn $\neg\neg F$;
both F, G

We can prove:

Theorem 1.48 *$\mathfrak{A}_3$ terminates. Moreover, if D is right-linear then $D \rightarrowtail \mathfrak{A}_3(D)$.*

Proof: Termination is straightforward. We will prove $D \rightarrowtail \mathfrak{A}_3(D)$ by induction on the structure of D. When D is an atomic deduction, the result is immediate. When D is a hypothetical deduction, the result follows by straightforward applications of the inductive hypothesis and Lemma 1.17. Next, suppose that D is of the form $D_1; D_2$. We distinguish three subscases:

(a) D_1 is of the form **assume** $F \ D_b$**:** In that case, letting $D'_b = \mathfrak{A}_3(D_b)$ and $D'_2 = \mathfrak{A}_3(D_2)$, we have

$$\mathfrak{A}_3(D) = \overline{\Delta}; \textbf{assume } F \ D'_b; D''_2 \tag{1.113}$$

where $(D''_2, \_, \Delta) = H(D'_2, \{F \Rightarrow \mathcal{C}(D'_b)\}, \emptyset)$. Inductively, $D_b \rightarrowtail D'_b$ and $D_2 \rightarrowtail D'_2$, so, by Lemma 1.17,

$$(\textbf{assume } F \ D_b); D_2 \rightarrowtail (\textbf{assume } F \ D'_b); D'_2. \tag{1.114}$$

Further, Lemma 1.40 implies

$$D'_2 \rightarrowtail \overline{\Delta}; D''_2. \tag{1.115}$$

From (1.115) and (1.114) we get

$$(\textbf{assume } F \;\; D_b); D_2 \rightarrowtail (\textbf{assume } F \;\; D'_b); \overline{\Delta}; D''_2. \tag{1.116}$$

By Lemma 1.35, we have $OA(D_x) \cap \{F \Rightarrow \mathcal{C}(D'_b)\} = \emptyset$ for all $D_x \in \Delta$, hence, by Lemma 1.38,

$$(\textbf{assume } F \;\; D'_b); \overline{\Delta}; D''_2 \rightarrowtail \overline{\Delta}; (\textbf{assume } F \;\; D'_b); D''_2. \tag{1.117}$$

Finally, from (1.116), (1.117), and the transitivity of $\rightarrowtail$, and in view of (1.113), we conclude $D \rightarrowtail \mathfrak{A}_3(D)$.

(b) D_1 is of the form **suppose-absurd** $F \;\; D_b$**:** The reasoning here is as in (a).

(c) None of the above**:** In this case the result follows directly from the inductive hypotheses.

When D is an u.g. or an e.i., the result follows from the inductive hypothesis and Lemma 1.17. ■

Finally, we address the question of size—whether $\mathfrak{A}_3(D)$ is always smaller than D. This will usually be the case, but there is an exception similar to that which we discussed in connection with *Hoist*: when algorithm H inserts a tail-position claim in D'' (lines 5 and 10). This will increase the size of the resulting deduction by one. (Any other claims generated by H will be eliminated later by the claim-removal algorithm, $\mathfrak{C}$, as guaranteed by Lemma 1.27). However, it is easy to avoid this special case, since H is defined so that whenever a tail-position claim is appended to D'', the last deduction of the list Δ has the same conclusion as the proposition asserted by the said claim. But if this is the case we can do away with D'' altogether, as well with the **assume** $F \;\; D'_b$, and simply output $\overline{\Delta}$ (and likewise for the **suppose-absurd**), in which case the size of the resulting deduction will be *strictly* smaller than that of the original. Accordingly, we modify lines 7 and 12 to be as follows, respectively:

$$\Delta \neq [] \;\; \textit{and} \;\; \mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'')\,? \rightarrow \overline{\Delta} \diamondsuit \overline{\Delta}; \textbf{assume } F \;\; D'_b; D''$$

and

$$\Delta \neq [] \;\; \textit{and} \;\; \mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'')\,? \rightarrow \overline{\Delta} \diamondsuit \overline{\Delta}; \textbf{suppose-absurd } F \;\; D'_b; D''.$$

This affects neither termination nor the property $D \rightarrowtail \mathfrak{A}_3(D)$ (on the assumption that D is right-linear), since the reduction is performed only if $\mathcal{C}(\overline{\Delta}) = \mathcal{C}(D'')$, so Theorem 1.48 continues to hold. Further, the modification guarantees that every claim inserted by H will eventually be removed by $\mathfrak{C}$, which ensures that the ultimate result of the simplification procedure will never be of greater size than the original.[9]

In conclusion, we define

$$\textit{restructure} = \mathfrak{MS} \cdot \mathfrak{A} \cdot \mathfrak{MS} \cdot \mathfrak{A}_3 \cdot \mathfrak{MS}$$

and

$$\textit{simplify} = \textit{contract} \cdot \textit{restructure}.$$

[9]Moreover, to avoid gratuitous hoistings, in practice we perform these restructurings only if the hypothesis F is in fact derived within D (lines 3 and 8).

Putting together the various preceding results will show that *simplify* always terminates and that $D \rightarrowtail \mathit{simplify}(D)$. Size is always either strictly decreased or preserved, except by *Hoist*, during the application of $\mathfrak{MS}$, and by $\mathfrak{A}_3$. Both of these transformations may introduce some additional trivial claims. However, we have taken care to define $\mathfrak{MS}$ and $\mathfrak{A}_3$ so that all such claims will be in non-tail positions and will thus be eventually eliminated by the claim-removal algorithm, $\mathfrak{C}$. Therefore, we conclude:

Theorem 1.49 *simplify always terminates;* $D \rightarrowtail \mathit{simplify}(D)$*;* $\mathit{SZ}(\mathit{simplify}(D)) \leq \mathit{SZ}(D)$.

1.6 Examples

In this section we illustrate *simplify* with a few examples of detours. For brevity, we write **mp** and **dn** for **modus-ponens** and **double-negation**, respectively. We begin with a couple of examples of conditional detours.

$$\mathrm{D} = \begin{array}{l} \mathbf{dn}\ \neg\neg F; \\ \mathbf{assume}\ F\ \mathbf{in} \\ \quad \mathbf{both}\ F, G; \\ \mathbf{mp}\ F \Rightarrow F \wedge G, F \end{array} \xrightarrow{\textit{restructure}} \begin{array}{l} \mathbf{dn}\ \neg\neg F; \\ \mathbf{both}\ F, G; \\ \mathbf{cond}\ F, F \wedge G; \\ \mathbf{mp}\ F \Rightarrow F \wedge G, F \end{array} \xrightarrow{\textit{contract}} \begin{array}{l} \mathbf{dn}\ \neg\neg F; \\ \mathbf{both}\ F, G; \end{array}$$

We continue with a detour based on negation:

$$D = \begin{array}{l} \mathbf{left\text{-}and}\ F \wedge G; \\ \mathbf{suppose\text{-}absurd}\ \neg F\ \mathbf{in} \\ \quad \mathbf{absurd}\ F, \neg F; \\ \mathbf{dn}\ \neg\neg F \end{array} \xrightarrow{\textit{restructure}} \begin{array}{l} \mathbf{left\text{-}and}\ F \wedge G; \\ \mathbf{suppose\text{-}absurd}\ \neg F\ \mathbf{in} \\ \quad \mathbf{absurd}\ F, \neg F; \\ \mathbf{dn}\ \neg\neg F \end{array} \xrightarrow{\textit{contract}} \mathbf{left\text{-}and}\ F \wedge G$$

Next we illustrate a disjunction detour. Let D be the following deduction:

$$\begin{array}{l} \mathbf{dn}\ \neg\neg(F_1 \wedge G); \\ \mathbf{left\text{-}either}\ F_1 \wedge G, F_2 \wedge G; \\ \mathbf{assume}\ F_1 \wedge G\ \mathbf{in} \\ \quad \mathbf{right\text{-}and}\ F_1 \wedge G; \\ \mathbf{assume}\ F_2 \wedge G\ \mathbf{in} \\ \quad \mathbf{right\text{-}and}\ F_2 \wedge G; \\ \mathbf{cases}\ (F_1 \wedge G) \vee (F_2 \wedge G), (F_1 \wedge G) \Rightarrow G, (F_2 \wedge G) \Rightarrow G \end{array}$$

We have:

$$D \xrightarrow{\textit{restructure}} \begin{array}{l} \mathbf{dn}\ \neg\neg(F_1 \wedge G); \\ \mathbf{left\text{-}either}\ F_1 \wedge G, F_2 \wedge G; \\ \mathbf{right\text{-}and}\ F_1 \wedge G; \\ \mathbf{cond}\ F_1 \wedge G, G; \\ \mathbf{assume}\ F_2 \wedge G\ \mathbf{in} \\ \quad \mathbf{right\text{-}and}\ F_2 \wedge G; \\ \mathbf{cases}\ (F_1 \wedge G) \vee (F_2 \wedge G), (F_1 \wedge G) \Rightarrow G, (F_2 \wedge G) \Rightarrow G \end{array} \xrightarrow{\textit{contract}} \begin{array}{l} \mathbf{dn}\ \neg\neg(F_1 \wedge G); \\ \mathbf{right\text{-}and}\ F_1 \wedge G \end{array}$$

We close with a biconditional detour. Let D be the following deduction:

```
**assume** $F \wedge G$ **in**
  **begin**
    **left-and** $F \wedge G$;
    **right-and** $F \wedge G$;
    **both** $G, F$
  **end**;
**assume** $G \wedge F$ **in**
  **begin**
    **right-and** $G \wedge F$;
    **left-and** $G \wedge F$;
    **both** $F, G$
  **end**;
**equivalence** $F \wedge G \Rightarrow G \wedge F, G \wedge F \Rightarrow F \wedge G$;
**left-iff** $F \wedge G \Leftrightarrow G \wedge F$
```

We have:

$$D \xrightarrow{\mathit{restructure}} D \xrightarrow{\mathit{contract}}$$

```
**assume** $F \wedge G$ **in**
  **begin**
    **left-and** $F \wedge G$;
    **right-and** $F \wedge G$;
    **both** $G, F$
  **end**
```

Bibliography

[1] K. Arkoudas. Athena. `http://www.pac.csail.mit.edu/athena`.

[2] K. Arkoudas. Denotational Proof Languages. PhD dissertation, MIT, 2000.

[3] K. Arkoudas. Type-α DPLs. MIT AI Memo 2001-25.

[4] K. Arkoudas. Type-ω DPLs. MIT AI Memo 2001-27.

[5] K. Arkoudas, S. Khurshid, D. Marinov, and M. Rinard. Integrating model checking and theorem proving for relational reasoning. In *Proceedings of the 7th International Seminar on Relational Methods in Computer Science (RelMiCS 7)*, Malente, Germany, May 2003.

[6] K. Arkoudas and M. Rinard. Deductive runtime certification. In *Proceedings of the 2004 Workshop on Runtime Verification*, Barcelona, Spain, April 2004.

[7] J. Barwise and J. Etchemendy. *Hyperproof: for Macintosh.* CSLI Publications, 1995.

[8] M. Bergmann, J. Moor, and J. Nelson. *The Logic Book.* Random House, New York, 1980.

[9] D. Bonevac. *Deduction.* Blackwell Publishing, 2003.

[10] George Boolos. Don't eliminate cut. *Journal of Philosophical Logic*, 13:373–378, 1984.

[11] I. M. Copi. *Symbolic Logic.* Macmillan Publishing Co., New York, 5th edition, 1979.

[12] D. V. Dalen. *Logic and Structure.* Springer Verlag, 1983.

[13] A. G. Dragalin. *Mathematical Intuitionism. Introduction to Proof Theory*, volume 67 of *Translations of Mathematical Monographs.* American Mathematical Society, Providence, RI, 1988.

[14] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Springer-Verlag, 2nd edition, 1994.

[15] F. B. Fitch. *Symbolic Logic: an Introduction.* The Ronald Press Co., New York, 1952.

[16] G. Gentzen. *The collected papers of Gerhard Gentzen.* North-Holland, Amsterdam, Holland, 1969. English translations of Gentzen's papers, edited and introduced by M. E. Szabo.

[17] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[18] M. J. C. Gordon and T. F. Melham. *Introduction to HOL, a theorem proving environment for higher-order logic.* Cambridge University Press, Cambridge, England, 1993.

[19] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[20] W. A. Howard. The formulae-as-types notion of construction. In J. Hindley and J. R. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms*, pages 479–490. Academic Press, 1980.

[21] G. Kahn. Natural semantics. In *Proceedings of Theoretical Aspects of Computer Science*, Passau, Germany, February 1987.

[22] D. Kalish and R. Montague. *Logic: Techniques of Formal Reasoning.* Harcourt Brace Jovanovich, Inc., New York, 1964. Second edition in 1980, with G. Mar.

[23] E. J. Lemmon. *Beginning Logic.* Hackett Publishing Company, 1978.

[24] G. Necula and P. Lee. Proof-carrying code. Computer Science Technical Report CMU-CS-96-165, CMU, September 1996.

[25] G. Necula and P. Lee. Efficient representation and validation of logical proofs. Computer Science Technical Report CMU-CS-97-172, CMU, October 1997.

[26] L. Paulson. *Isabelle, A Generic Theorem Prover.* Lecture Notes in Computer Science. Springer-Verlag, 1994.

[27] F. J. Pelletier. A Brief History of Natural Deduction. *History and Philosophy of Logic*, 20:1–31, 1999.

[28] J. Pelletier. Automated natural deduction in thinker. *Studia Logica*, 60(1):3–43, 1998.

[29] G. D. Plotkin. A structural approach to operational semantics. Research Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, September 1981.

[30] John L. Pollock. Rational cognition in OSCAR. In *Agent Theories, Architectures, and Languages*, pages 71–90, 1999.

[31] D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, Stockhol, Sweden, 1965.

[32] N. Rescher. *Introduction to Logic.* St. Martin's Press, 1964.

[33] J. C. Reynolds. *Theories of Programming Languages.* Cambridge University Press, 1998.

[34] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, Cambridge, England, 1996.

[35] A. Voronkov. The anatomy of Vampire: implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 15(2), 1995.

[36] W. Wechler. *Universal Algebra for Computer Scientists.* Springer-Verlag, 1992.

[37] C. Weidenbach. Combining superposition, sorts, and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2. North-Holland, 2001.