The Computational Complexity of Quantifier Raising

Ned Sanger
University of California, Los Angeles
nedsanger@ucla.edu

Abstract

Quantifier Raising (QR) has been a workhorse of the syntax-semantics interface for nearly 50 years, but its computational properties are barely known. To fill this gap, I study the time complexity of the following decision problem: given a tree whose leaves are assigned semantic types, can repeated applications of QR lead to a well-typed logical form (LF)? The main result is that this problem is NP-complete, meaning that there exists no general and efficient algorithm for building interpretable LFs using QR (assuming $P \neq NP$). The problem remains NP-complete even if one constrains QR by limiting the type of traces, limiting the number of atomic semantic types, banning parasitic scope, and banning cyclic QR.

1 Introduction

Quantifier Raising (QR) is *the* tool for repairing type mismatches and analyzing scope in LF-based approaches to semantic interpretation. But despite almost fifty years of widespread use since May (1977) introduced it, little is known about its computational properties. In order to improve our formal understanding of QR, this paper therefore answers some basic but unresolved questions about its complexity:

- Given a logical form that is *not* well-typed, how difficult is it to determine whether repeated applications of QR can turn it into one that *is*?
- Does the difficulty depend on the number of atomic types one uses? Does it depend on the possibility of higher-order traces, or the possibility of parasitic scope, or the possibility of QR applying to an element multiple times?
- Do we need any purely formal constraints on quantifier raising?

My answer to the first question above is: *very* difficult (NP-complete). My answer to the second is: *no*, none of those constraints affects the difficulty (as measured by asymptotic, worst-case time complexity). My answer to the third question is: *yes*, we need a formal constraint that prevents QR from targeting λ-abstractions that were themselves created by QR.

To date, the only formal study of QR is Barker (2020). This paper looks at QR in the same spirit that Barker did and builds on the foundation he laid.

The rest of this paper is organized as follows. Section 2 provides a formal definition of logical forms, of Quantifier Raising, and of the decision problem LF REPAIR: given a possibly ill-typed logical form, is there a way to make it well-typed by repeatedly applying QR? The section also contains proofs of a few useful facts about logical forms. Section 3 proves that LF REPAIR belongs to the complexity class NP, following Barker (2020) closely. Section 4 proves that LF REPAIR is NP-hard by a reduction from the directed Hamiltonian cycle problem. Section 5 makes some concluding observations.

I assume that the reader is familiar with basic notions from computational complexity theory, in particular the theory of NP-completeness. For a general introduction, see Papadimitriou (1994) and Arora and Barak (2009). For overviews focused on linguistic issues, see Pratt-Hartmann (2010) and Barton et al. (1987).

2 Preliminaries and Notation

2.1 Types, Logical Forms, Contexts

Given a finite set $\mathbf{A}$ of atomic types, the set of *types* over it is

$$\mathbf{T} := \mathbf{A} \mid \mathbf{T} \to \mathbf{T}.$$

In the rest of the paper, lowercase greek letters ($\alpha, \beta, \gamma, \ldots$) will range over types. As usual, '$\to$'

associates to the right. I often drop the '$\to$', so that $\alpha\beta\gamma$, for example, is a shorthand for the type $\alpha \to (\beta \to \gamma)$. I use $\alpha \xrightarrow{m} \beta$ as a shorthand for

$$\underbrace{\alpha \to \cdots \to \alpha}_{m \text{ times}} \to \beta$$

A useful way to stratify the set of types is by assigning each type an *order*. If we think of types as describing functions, then the order quantifies how complex that function's arguments are. Formally, the order of an atomic type p is $\mathrm{ord}(p) = 1$, and the order of a complex type $\alpha \to \beta$ is

$$\mathrm{ord}(\alpha \to \beta) = \max(1 + \mathrm{ord}(\alpha), \mathrm{ord}(\beta)).$$

Types of order n describe functions whose arguments have types of order less than n. Some examples: if $e, t \in \mathbf{A}$, then $\mathrm{ord}(e) = 1$, $\mathrm{ord}(eet) = 2$, and $\mathrm{ord}\big((et)et\big) = 3$.

Let $\{\mathbf{V}_\alpha\}_{\alpha \in \mathbf{T}}$ be a disjoint family of sets indexed by $\mathbf{T}$, and let

$$\mathbf{V} = \bigcup_{\alpha \in \mathbf{T}} \mathbf{V}_\alpha.$$

The members of each set $\mathbf{V}_\alpha = \{x, y, \ldots\}$ are *variables of type* α. When the type of a variable x is relevant, I will write it as x^α to indicate that $x \in \mathbf{V}_\alpha$.

The set $\mathbf{L}$ of *logical forms* is

$$\mathbf{L} := \mathbf{T} \,|\, \mathbf{V} \,|\, \mathbf{L} \cdot \mathbf{L} \,|\, \lambda \mathbf{V}\, \mathbf{L}.$$

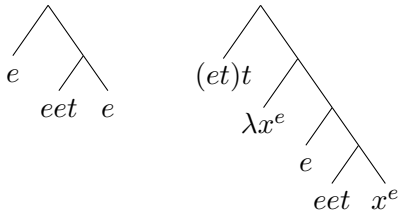
In words, an LF is either a type, a variable, the concatenation of two LFs, or a λ-abstraction over an LF. An *abstraction-free* LF is one built from only the first three cases, i.e., an LF that contains no λ-abstractions. It's usual in semantics to place typed lexical items rather than types themselves at the leaves of logical forms. But since this paper is only concerned with type coherence, the definition above is simpler.

The constructor '$\cdot$' associates to the left. It is also commutative, so there is no difference between the LFs $\alpha \cdot \beta$ and $\beta \cdot \alpha$. Nodding at linguistic practice, I sometimes call variables *traces*.

If $e, t \in \mathbf{A}$, then

$$e \cdot (eet \cdot e) \quad \text{and} \quad (et)t \cdot \lambda x^e\, (e \cdot (eet \cdot x^e))$$

are each examples of logical forms. I sometimes represent LFs as unordered binary trees in a way familiar to linguists, e.g.



Later on I will need to talk about LFs with a "hole" that can be plugged by another LF. I call these *contexts*. Formally, the set of contexts is

$$\mathbf{C} = [\,] \,|\, \lambda \mathbf{V}\, \mathbf{C} \,|\, \mathbf{L} \cdot \mathbf{C}.$$

If Γ is a context and Δ is an LF (or context), then $\Gamma[\Delta]$ is the LF (or context) that results from substituting Δ for $[\,]$ in Γ. In section 4, it will be useful to have notation for repeatedly nesting a context inside itself: if Γ is a context, let $\Gamma^0 = [\,]$ and $\Gamma^n = \Gamma[\Gamma^{n-1}]$. In the rest of the paper, capital Greek letters (Γ, Δ, etc.) will denote either LFs or contexts.

The *typing function* $\tau : \mathbf{L} \to \mathbf{T}$ is the following partial map:

$$\tau(\Gamma) = \begin{cases} \alpha & \text{if } \Gamma = \alpha, \\ \beta & \text{if } \Gamma = \Delta \cdot \Theta, \tau(\Delta) = \alpha\beta \\ & \quad \text{and } \tau(\Theta) = \alpha, \\ \alpha\beta & \text{if } \Gamma = \lambda x^\alpha\, \Delta[x^\alpha] \text{ and} \\ & \quad \tau(\Delta[\alpha]) = \beta, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The second and third cases correspond respectively to the semantic rules of *function application* and *predicate abstraction* in Heim and Kratzer (1998). When $\tau(\Gamma) = \alpha$ holds, I usually write $\Gamma : \alpha$ or say that Γ has type α. If $x \in \mathbf{V}_\alpha$, then $\tau(x^\alpha)$ is technically undefined, but I will abuse terminology below and still say that x has type α.

An LF Γ is *well typed* iff $\tau(\Gamma)$ is defined. The two LFs mentioned above, $e \cdot (eet \cdot e)$ and $(et)t \cdot \lambda x^e\, (e \cdot (eet \cdot x^e))$, are well typed because they each have type t.

Notice that if Γ has an unbound trace, i.e., a variable that is not in the scope of a λ-abstraction, then it cannot be well typed. The definition of the typing function therefore enforces a form of the empirically motivated proposal that remnant movement must reconstruct (Huang, 1993; Bhatt and Dayal, 2007).

2.2 Quantifier Raising

The surface syntax of a sentence is often ill-typed and therefore not suitable for semantic interpreta-

tion. The classic example comes from transitive sentences with a quantifier in object position ("the critic panned every film"), which correspond to the ill-typed LF $e \bullet (eet \bullet (et)t)$. Function application can't combine the type of a transitive verb, eet, with the type of a generalized quantifier, $(et)t$.

In this case, we know what LF we would *like* to interpret (Heim and Kratzer, 1998, 178–79). The proper semantic argument for the quantifier in object position is an abstraction over its context, also known as its continuation, so the LF we want is

$$\underbrace{(et)t}_{\text{quantifier}} \bullet \underbrace{\lambda x^e\,(e \bullet (eet \bullet x^e))}_{\text{continuation}}.$$

Quantifier Raising is just an operation for moving from ill-typed LFs like $e \bullet (eet \bullet (et)t)$ to the well-typed one above. Given an LF $\Gamma[\Delta[\Theta]]$, QR "raises" Θ so that it becomes Δ's sister, puts a variable in Θ's old position, and adds a binder for that variable between Θ and Δ. The result is $\Gamma[\Theta \bullet \lambda x\,\Delta[x]]$. In the case above,

$$\Gamma = [\,],\quad \Delta = e \bullet (eet \bullet [\,]),\quad \Theta = (et)t.$$

My goal is to study QR in its most general form. Accordingly, I won't constrain it by—just to name a few possibilities—imposing scope islands, scope economy (Fox, 2000), or limits on the types of traces (Poole, 2024). Whatever empirical merits they might have, these restrictions would only reduce the flexibility of the mathematical results in this paper. More importantly, to understand the effect (if any) that specific constraints have on complexity, it's necessary to first understand the complexity of the general case.

That said, *one* formal constraint is necessary to keep QR from overgenerating: the operation should be disallowed from targeting abstractions, that is, LFs of the form $\lambda x\,\Gamma$. To see what can go wrong without this constraint, look at figure 1. The leftmost tree shows an innocent-looking analysis of the sentence *Mary gave John the book*. It might not seem like QR can do anything interesting to this tree, because no subtree has a complex enough type to take scope. But the ability to raise abstractions lets us create a scope-taker.

The middle tree is the result of applying QR to *gave*, moving it to a position right below *Mary* and leaving a trace of type eet. The rightmost tree is the result of quantifier raising the *abstraction* created by the previous step to the root of the tree and leaving a trace of type e. Not only is the resulting LF well typed, but (assuming natural denotations for the lexical items) it means *gave (j) (m) (b)*, that is, *the book gave John Mary*.

Completely unrestricted QR therefore allows us to create scope-takers on the fly, even in sentences where nothing ought to be taking scope, and the semantic effect is disastrous. It does not behave in the way linguists expect QR to behave and it lacks good theoretical properties. The definition below therefore explicitly prevents QR from targeting abstractions.[1]

Definition 2.1. *Quantifier Raising* is the smallest binary relation $\to_{\text{QR}} \subseteq \mathbf{L} \times \mathbf{L}$ such that for all contexts Γ and Δ, all LFs Θ that are *not* abstractions, and all fresh variables x,

$$\Gamma[\Delta[\Theta]] \to_{\text{QR}} \Gamma[\Theta \bullet \lambda x\,\Delta[x]].$$

The reflexive, transitive closure of $\to_{\text{QR}}$ is $\to^*_{\text{QR}}$.

Because this operation targets not just quantifiers but any part of an LF at all, the name "Quantifier Raising" is misleading. "Scope Taking," as Barker (2020) says, is more accurate. But like him, I've stuck with the familiar name because it is entrenched.

Now we can define the central decision problem of this paper:

> LF REPAIR
> Given a set of atomic types $\mathbf{A}$, a type α, and a variable- and abstraction-free LF Γ, is there an LF Δ of type α such that $\Gamma \to^*_{\text{QR}} \Delta$?

The name comes from the idea that ill-typed LFs are "broken," and that repairing them involves raising the scope-takers and finding appropriate trace types in such a way that the result is well-typed.

Here is an example. Let $\mathbf{A} = \{e, t\}$ and let Γ be the ill-typed LF

$$\big[(et)e \bullet \big[(((et)et)et)et \bullet et\big]\big] \bullet \big[eet \bullet (et)t\big].$$

[1] A more elegant way to ban QR of abstractions is to change the structure of LFs so that no constituent corresponds to an abstraction in the first place. Variable binding would instead be handled by an index on moved items, so that QR would relate an LF $\Gamma[\Delta[\Theta]]$ to something like $\Gamma[\Theta_x \bullet \Delta[x]]$ (Heim, 1997; Heim and Kratzer, 1998, 187–88). But this approach can't easily accomodate *parasitic scope* (Barker, 2007). Since I want a formalization of quantifier raising that covers all of its uses in linguistics, I've therefore stuck with tradition and used λ-abstraction to bind variables.

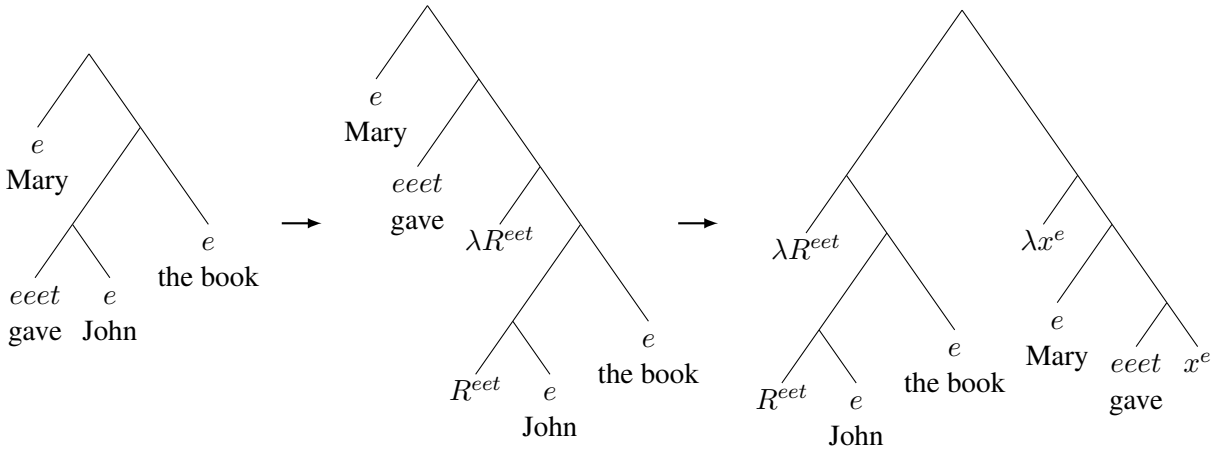


Figure 1: Raising abstractions is dangerous.

Is there a Δ of type t such that $\Gamma \to^*_{\text{QR}} \Delta$? The answer, not immediately obvious, is yes:

$$\begin{aligned}\Gamma &\to_{\text{QR}} (et)t \bullet \lambda x^e \big[\big[(et)e \bullet \big[(((et)et)et)et \bullet et\big]\big] \\ &\qquad \bullet \big[eet \bullet x^e\big]\big] \\ &\to_{\text{QR}} (et)t \bullet \big[(((et)et)et)et \bullet \lambda Q^{(et)et} \lambda x^e \\ &\qquad \big[\big[(et)e \bullet \big[Q^{(et)et} \bullet et\big]\big] \bullet \big[eet \bullet x^e\big]\big]\big],\end{aligned}$$

and this last LF has type t.

2.3 Normalization

QR's purpose is to let a function take some portion of its semantic context as an argument. But it's also possible for the raised element to become an argument rather than a function of its context. For instance: $e \bullet et \to_{\text{QR}} e \bullet \lambda x^e (x^e \bullet et)$. Such steps are licit but pointless—they will never help turn an ill-typed LF into a well-typed one. In general, if an LF has the form $\Delta[\Pi \bullet \lambda x\, \Sigma[x]]$, where Π has type α and $\lambda x\, \Sigma[x]$ has type $\alpha\beta$, then I call the abstraction $\lambda x\, \Sigma[x]$ *vacuous*. A well-typed LF is *normalized* iff it contains no vacuous abstractions.

The next lemma justifies the label "vacuous."

Lemma 2.1. *Let Γ be a variable- and abstraction-free LF. If $\Gamma \to^*_{\text{QR}} \Delta$ and $\Delta : \alpha$, then there exists a normalized LF $\Delta' : \alpha$ such that $\Gamma \to^*_{\text{QR}} \Delta'$.*

Proof. If Δ isn't already normalized, then it contains a vacuous abstraction and therefore has the form $\Theta[\Pi \bullet \lambda x^\beta\, \Sigma[x^\beta]]$, where $\lambda x^\beta\, \Sigma[x^\beta]$ has type $\beta\gamma$ and Π has type β. Like Δ, the LF $\Theta[\Sigma[\Pi]]$ also has type α, but it contains one fewer vacuous abstraction.

By definition, QR cannot target an abstraction. And since Δ is well-typed, no step in the derivation $\Gamma \to^*_{\text{QR}} \Delta$ raises x^β outside the scope of its binder λx^β. The derivation $\Gamma \to^*_{\text{QR}} \Delta$ therefore has the following form:

$$\begin{aligned}\Gamma &\to^*_{\text{QR}} \Theta_1[\Sigma_1[\Pi_1]] \\ &\to_{\text{QR}} \Theta_1[\Pi_1 \bullet \lambda x^\beta\, \Sigma_1[x^\beta]] \\ &\to_{\text{QR}} \Theta_2[\Pi_2 \bullet \lambda x^\beta\, \Sigma_2[x^\beta]] \\ &\to_{\text{QR}} \cdots \\ &\to_{\text{QR}} \Theta_n[\Pi_n \bullet \lambda x^\beta\, \Sigma_n[x^\beta]],\end{aligned}$$

where the Θ_i and Σ_i are contexts; the Π_i are LFs; and $\Theta_n = \Theta$, $\Sigma_n = \Sigma$, and $\Pi_n = \Pi$. Then

$$\begin{aligned}\Gamma &\to^*_{\text{QR}} \Theta_1[\Sigma_1[\Pi_1]] \\ &\to_{\text{QR}} \Theta_2[\Sigma_2[\Pi_2]] \\ &\to_{\text{QR}} \cdots \\ &\to_{\text{QR}} \Theta_n[\Sigma_n[\Pi_n]]\end{aligned}$$

is a valid derivation of $\Theta[\Sigma[\Pi]]$ from Γ.

By repeating this process, we can eliminate vacuous abstrations one by one until we obtain a normalized LF $\Delta' : \alpha$ such that $\Gamma \to^*_{\text{QR}} \Delta'$. □

The following lemma, needed in section 4, limits the trace types that need to be considered when repairing an LF.

Lemma 2.2. *Suppose Γ is a variable- and abstraction-free LF, Δ is a normalized LF, and $\Gamma \to^*_{\text{QR}} \Delta$. Let m be the maximum order among Γ's leaves. If x^α is a trace occurring in Δ, then* $\text{ord}(\alpha) < m - 1$.

Proof. Let n be the maximum order of all of Δ's traces, and suppose $n \geq m - 1$. Consider a trace x^α of order n. The λ-abstractor that binds x^α must have a sister Π, and we can therefore write $\Delta =$

$\Theta[\Pi \bullet \lambda x^{\alpha}\, \Sigma[x]]$.[2] Because Δ is normalized, Π's type has the form $(\alpha \to \tau(\Sigma[\alpha])) \to \beta$ for some β, and so

$$\mathrm{ord}(\tau(\Pi)) > \mathrm{ord}(\alpha) + 1 = n + 1. \qquad (1)$$

Every node within the subtree Π is either (i) a leaf of order at most $m \leq n+1$, (ii) function application of two elements of order at most $n+1$, or (iii) an abstraction over a variable of order at most n. It follows (by an induction from the leaves to the root) that every node of Π has order at most $n+1$, so $\mathrm{ord}(\tau(\Pi)) \leq n+1$. This contradicts (1). Therefore $n < m - 1$. □

The two lemmas above make it easier to determine if an LF Γ can be repaired using QR. It's sufficient to search for normalized LFs where the order of the trace types is below an upper bound coming from Γ.

2.4 Directed Graphs and Hamiltonian Cycles

The proof in section 4 uses some terminology from graph theory. As a reminder, a directed graph (or digraph) is a pair $\langle V, E\rangle$, where V is a finite set of *vertices* and $E \subseteq V \times V$ a set of *edges*. The first component of an edge is its *tail*, the second its *head*. Given a vertex v, its *indegree* $\deg^-(v)$ and *outdegree* $\deg^+(v)$ are the number of edges of which v is a head and tail, respectively.

A *Hamiltonian cycle* in a digraph is a sequence of edges $e_1, e_2, \ldots, e_n$ such that for $0 < i < n$, the head of e_i is the tail of e_{i+1}, the head of e_n is the tail of e_1, and every vertex occurs exactly once as a head and a tail of an edge in the sequence. The following decision problem is NP-complete (see, e.g., Garey and Johnson 1979):

> DIRECTED HAMILTONIAN CYCLE
> Given a digraph, does it have a Hamiltonian cycle?

3 LF REPAIR is in NP

This goal in this section is to prove that LF REPAIR is in NP. This result would be a simple corollary of the proof in Barker (2020) that LF REPAIR is decidable, but that proof contains an error (explained below).[3] The rest of this section therefore provides a corrected proof, and uses it to show that LF REPAIR is in NP. Because many of the details from Barker's paper don't need to change, the proofs in this section move quickly; see his paper for further information.

The high-level idea, following Barker's lead, is to introduce a type-logical grammar, QRT (Quantifier Raising with Types), such that (i) provability in QRT is in NP, and (ii) LF REPAIR is reducible in polynomial time to provability in QRT.

The axiom and inference rules of QRT are shown in figure 2. Γ, Δ and Θ represent LFs or contexts, and A and B represent types. $\lambda^{\uparrow}$ and $\lambda^{\downarrow}$ are subject to the restriction that the LF Θ not be a λ-abstraction (or equivalently, that Θ be either a type or of the form $\Sigma \bullet \Omega$). This restriction, which is the only difference between the version of QRT in this paper and the one in Barker (2020), will crucially guarantee that theorem 3.2 holds. Unrestricted raising—which is what caused the error in Barker's proof—wreaks havoc when combined with the commutativity of '•'. For instance, it lets us freely swap an inner and outer context above a '•'-node, which results in massive overgeneration:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma[\Delta[A \bullet B]] \vdash C}{\Gamma[A \bullet \lambda x\, \Delta[x \bullet B]] \vdash C}\,\lambda^{\downarrow}}{(\lambda x\, \Delta[x \bullet B]) \bullet \lambda y\, \Gamma[A \bullet y] \vdash C}\,\lambda^{\downarrow}}{\Delta[(\lambda y\, \Gamma[A \bullet y]) \bullet B]) \vdash C}\,\lambda^{\uparrow}}{\Delta[\Gamma[A \bullet B]] \vdash C}\,\lambda^{\uparrow}$$

This issue is not really different from the one in figure 1, and my solution is similar: forbidding $\lambda^{\uparrow}$ and $\lambda^{\downarrow}$ from targeting abstractions.

Figure 3 shows a small QRT proof.

The next theorem shows that a cut-elimination theorem holds for QRT sequents that are *abstraction-free*, that is, sequents $\Gamma \vdash A$ where Γ contains no λ-abstractions.

Theorem 3.1. *Every abstraction-free theorem of QRT has a* CUT*-free proof.*

Proof. By the normal method of permuting cuts upward. When the cut formula is not principal in both premises of CUT, the permutations are straightforward.

The subtle case is when the cut formula *is* principal in both premises:

$$\dfrac{\dfrac{\Gamma[A] \vdash B}{\lambda x\, \Gamma[x] \vdash AB}\,{\to}\mathrm{R} \quad \dfrac{\Delta \vdash A \quad \Theta[B] \vdash C}{\Theta[AB \bullet \Delta] \vdash C}\,{\to}\mathrm{L}}{\Theta[\lambda x\, \Gamma[x] \bullet \Delta] \vdash AB}\,\textsc{Cut}$$

[2] Technically, there could be more abstractions intervening, so that I should write $\Theta[\Pi \bullet \lambda y_1^{\beta_1} \ldots \lambda y_k^{\beta_k} \lambda x^{\alpha}\, \Sigma[x^{\alpha}]]$. But this doesn't affect the rest of the proof.

[3] Barker points out the mistake himself in a footnote added after the early-access version of the paper was published.

$$\frac{}{A \vdash A}\,\textsc{Ax} \qquad \frac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[AB \bullet \Gamma] \vdash C}\,{\to}\mathrm{L} \qquad \frac{\Gamma[A] \vdash B}{\lambda x\,\Gamma[x] \vdash AB}\,{\to}\mathrm{R}$$

$$\frac{\Gamma[\Theta \bullet \lambda x\,\Delta[x]] \vdash A}{\Gamma[\Delta[\Theta]] \vdash A}\,\lambda^{\uparrow} \qquad \frac{\Gamma[\Delta[\Theta]] \vdash A}{\Gamma[\Theta \bullet \lambda x\,\Delta[x]] \vdash A}\,\lambda^{\downarrow} \qquad \frac{\Gamma \vdash A \qquad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\,\textsc{Cut}$$

Figure 2: Axiom and Rules of QRT.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{e \vdash e}\,\textsc{Ax} \quad \dfrac{\dfrac{}{et \vdash et}\,\textsc{Ax} \quad \dfrac{}{t \vdash t}\,\textsc{Ax}}{(et)t \bullet et \vdash t}\,{\to}\mathrm{L}}{(et)t \bullet (eet \bullet e) \vdash t}\,{\to}\mathrm{L}}{\lambda x\,((et)t \bullet (eet \bullet x)) \vdash et}\,{\to}\mathrm{R} \qquad \dfrac{}{t \vdash t}\,\textsc{Ax}}{(et)t \bullet \lambda x\,((et)t \bullet (eet \bullet x)) \vdash t}\,{\to}\mathrm{L}}{(et)t \bullet (eet \bullet (et)t) \vdash t}\,\lambda^{\uparrow}$$

Figure 3: An example QRT derivation.

If Δ is not an abstraction, then we can replace the cut with two smaller cuts:

$$\dfrac{\dfrac{\Delta \vdash A \qquad \Gamma[A] \vdash B}{\Gamma[\Delta] \vdash B}\,\textsc{Cut} \qquad \Theta[B] \vdash C}{\dfrac{\Theta[\Gamma[\Delta]] \vdash C}{\Theta[\lambda x\,\Gamma[x] \bullet \Delta]}\,\lambda^{\downarrow}}\,\textsc{Cut}$$

If Δ *is* an abstraction, then it turns out there is no need to eliminate the cut, because it couldn't have occurred in the proof of an abstraction-free theorem. To see why, first define a sequent to be *stuck* if its left-hand side has the form $\Sigma[\Phi \bullet \Psi]$ where Φ and Ψ are both abstractions. A straightforward induction shows that if a sequent in a QRT proof is stuck, then every subsequent sequent—and in particular the conclusion—is stuck too. Since $\Theta[\lambda x\,\Gamma[x] \bullet \Delta]$ is stuck whenever Δ is an abstraction, the proof's conclusion cannot be abstraction-free. □

Theorem 3.2. *Every abstraction-free theorem of QRT has a* Cut- *and* $\lambda^{\downarrow}$*-free proof.*

Proof. Take a Cut-free proof of an abstraction-free theorem, and consider an occurrence of $\lambda^{\downarrow}$ of maximal depth. Because the proof's conclusion is abstraction-free, there must be a lower occurrence of $\lambda^{\uparrow}$ eliminating the abstraction introduced by $\lambda^{\downarrow}$. If the occurrence of $\lambda^{\uparrow}$ is immediately below, then the application of $\lambda^{\downarrow}$ is eliminable:

$$\dfrac{\dfrac{\Gamma[\Delta[\Theta]] \vdash A}{\Gamma[\Theta \bullet \lambda x\,\Delta[x]] \vdash A}\,\lambda^{\downarrow}}{\Gamma[\Delta[\Theta]] \vdash A}\,\lambda^{\uparrow} \quad \rightsquigarrow \quad \Gamma[\Delta[\Theta]] \vdash A$$

Otherwise we can permute the occurrence of $\lambda^{\downarrow}$ downward across any occurrence of ${\to}$L, ${\to}$R, and $\lambda^{\uparrow}$ until it meets up with a suitable lower occurrence of $\lambda^{\uparrow}$ and can be removed. (The requirement that $\lambda^{\uparrow}$ and $\lambda^{\downarrow}$ not target abstractions guarantees that it is possible to permute $\lambda^{\downarrow}$ across $\lambda^{\uparrow}$ whenever needed.)

By repeating this procedure, we can eliminate each occurrence of $\lambda^{\downarrow}$ one by one. Since the procedure doesn't introduce any new applications of Cut, the end result is a Cut- and $\lambda^{\downarrow}$-free proof. □

Theorem 3.3. *Deciding whether an abstraction-free sequent is a theorem of QRT is in NP.*

Proof. Theorem 3.2 shows that every abstraction-free theorem of QRT has a proof that uses only the rules Ax, ${\to}$L, ${\to}$R, and $\lambda^{\uparrow}$. Reading the rules from bottom to top, ${\to}$L and ${\to}$R each eliminate at least one '$\bullet$' or '$\to$', and no rule introduces new ones. The number of occurrences of ${\to}$L and ${\to}$R in a Cut- and $\lambda^{\downarrow}$-free proof is therefore bounded from above by the total number of '$\bullet$'s and '$\to$'s in the conclusion. $\lambda^{\uparrow}$ can only apply if a λ-abstraction was introduced by a higher occurrence of ${\to}$R, and so the number of times it is used is likewise bounded from above by the number of '$\bullet$'s and '$\to$'s in the conclusion.

Every abstraction-free theorem of QRT therefore has a proof whose size is polynomial in the theorem's length and which can serve as an efficient witness of theoremhood. □

Given an abstraction-free LF Γ and a type A, there is an LF Δ of type A such that $\Gamma \to^*_{\text{QR}} \Delta$ if and only if QRT proves $\Gamma \vdash A$. See the appendix to Barker (2020) for a proof, which applies without modifications to the version of QRT used in this paper. LF REPAIR therefore has the same time complexity as provability (of abstraction-free sequents) in QRT, which is what we set out to show:

Theorem 3.4. LF REPAIR *is in NP.*

4 LF REPAIR is NP-Hard

4.1 Encoding Hamiltonian Cycles in LFs

This section describes a polynomial-time reduction from DIRECTED HAMILTONIAN CYCLE to LF REPAIR. Let $G = \langle V, E\rangle$ be a connected digraph with n vertices and m edges such that at least one vertex has indegree greater than one and at least one vertex has outdegree greater than one. It's convenient to assume that the vertices of G are some initial segment of the natural numbers, i.e. $V = \{1, \ldots, n\}$. I construct an instance of LF REPAIR whose answer is "yes" iff G has a Hamiltonian cycle. The method resembles the one in Krantz and Mobile (2001), who encode DIRECTED HAMILTONIAN CYCLE in the multiplicative fragment of linear logic.

The set of atomic types for the instance consists of a type v for each $v \in V$, plus the additional types a, b, c, d, e, and t. That is,

$$\mathbf{A} = \{1, \ldots, n\} \cup \{a, b, c, d, e, t\}.$$

Below, I'll use the following abbreviations for types:

$$\begin{aligned}
\mathrm{E}^v_u &:= (a \to u) \to b \to v,\\
\mathrm{V}_v &:= (a \to v) \to c \to v,\\
\mathrm{G}_v &:= (a \to t) \to d \to v,\\
\mathrm{G}^v &:= (a \to v) \to e \to t.
\end{aligned}$$

I'll now construct an LF Γ such that $\Gamma \to^*_{\text{QR}} \Delta$ for some $\Delta : t$ iff G has a Hamiltonian cycle. Without lost of generality, I assume the cycle begins at the vertex 1. To make the construction more concrete, I will illustrate it as I go using the digraph in figure 4. Notice that the graph has a Hamiltonian cycle: $(1,3), (3,2), (2,4), (4,1)$.

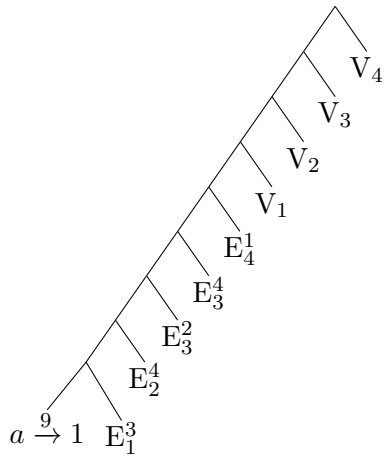
I'll build up Γ in four layers, constructing an LF Γ_1 and three contexts Γ_2, Γ_3, Γ_4, such that $\Gamma = \Gamma_4[\Gamma_3[\Gamma_2[\Gamma_1]]]$. The plan is as follows. Γ_1 will contain a leaf E^v_u for each edge (u, v) of G, as well as a leaf V_v for every vertex. Each one of these leaves will need to undergo QR to have a chance of making Γ into a well-typed LF. The context Γ_2 will have n positions to which n of the leaves E^v_u from Γ_1 can undergo QR (as well as n positions for the V_v); by design, the types will properly compose if and only if the n corresponding edges in G form a Hamiltonian cycle.

The $m - n$ leaves E^v_u corresponding to edges not occurring in the cycle will also need a place that they can QR to. The context Γ_4 will provide $m - n$ positions for them. Since a Hamiltonian cycle enters and exits every vertex exactly once, for each vertex v there are $\deg^-(v) - 1$ edges with head v and $\deg^+(v) - 1$ edges with tail v that do not occur in the cycle. Γ_3 will accordingly contain $\deg^+(v) - 1$ leaves G_v and $\deg^-(v) - 1$ leaves G^v. For each E^v_u that QRs into Γ_4, G_u and G^v will QR right below and above it in Γ_4. This last step is needed to make composition run smoothly.

Now for the technical details. Let $(u_1, v_1), \ldots, (u_m, v_m)$ be G's edges, and let

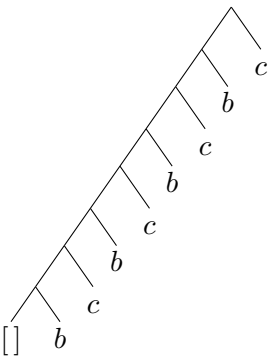
$$\Gamma_1 = (a \xrightarrow{m+n} 1) \bullet \mathrm{E}^{v_1}_{u_1} \bullet \ldots \bullet \mathrm{E}^{v_m}_{u_m} \bullet \mathrm{V}_1 \bullet \ldots \bullet \mathrm{V}_n.$$

For the digraph in figure 4, Γ_1 would be



The intention is that every E^v_u and V_v will take scope and leave a trace of type a. The leaf of type $a \xrightarrow{m+n} 1$ will consume these traces one by one, producing a subtree of type 1.

Let $\Delta = [\,] \bullet b \bullet c$ and let $\Gamma_2 = \Delta^n$. For the digraph in figure 4, $\Gamma_2 = \Delta^4$, which looks like



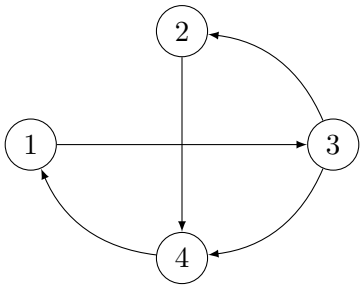


Figure 4: A directed graph.

If E_u^v and V_v each QR directly below the occurrences of b and c respectively in $\Delta[u]$, then the resulting subtree will have type v. In other words, if we feed u as an "input" to the context Δ, and then apply the "resources" E_u^v and V_v, the "output" will have type v. Δ is therefore a gadget for simulating the traversal of an edge from u to v. The left side of figure 5 illustrates the simulation with E_1^3 and V_3.

By nesting n copies of Δ, Γ_2 can simulate a path of length n in G. Notice that simulating the traversal of an edge with head v requires raising an occurrence of V_v. Since Γ will contain exactly one leaf V_v for each vertex v, the paths Γ_2 can simulate are therefore those that visit each vertex exactly once, that is, Hamiltonian paths.

Γ_3 is most complicated. For each $v \in V$, let

$$\Phi_v = [\,] \bullet \underbrace{\mathrm{G}_v \bullet \ldots \bullet \mathrm{G}_v}_{\deg^+(v) - 1 \text{ times}}, \quad \Psi_v = [\,] \bullet \underbrace{\mathrm{G}^v \bullet \ldots \bullet \mathrm{G}^v}_{\deg^-(v) - 1 \text{ times}}$$

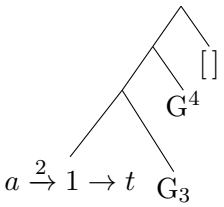
and let

$$\Phi = \Phi_1[\Phi_2[\ldots \Phi_n]] \quad \text{and} \quad \Psi = \Psi_1[\Psi_2[\ldots \Psi_n]].$$

Furthermore, let

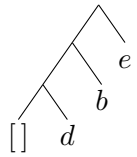
$$\begin{aligned} k &= \sum_{v=1}^{n} (\deg^+(v) - 1) + (\deg^-(v) - 1) \\ &= 2m - 2n. \end{aligned}$$

Then $\Gamma_3 = \Psi[\Phi[a \xrightarrow{k} 1 \to t]] \bullet [\,]$. In the case of the digraph in figure 4 (where only Φ_3 and Ψ_4 are nontrivial), Γ_3 would be



To make Γ well typed, each G_u and G^v will need to undergo QR and leave behind a trace of type a. The leaf of type $a \xrightarrow{k} 1 \to t$ will consume these traces, resulting in a subtree of type $1 \to t$.

Finally, let $\Sigma = [\,] \bullet d \bullet b \bullet e$, and let $\Gamma_4 = \Sigma^{m-n}$. In the case of the digraph in figure 4, Γ_4 would be



The context Σ is a gadget that works similar to Δ. If G_u, E_u^v, and G^v each QR directly below the occurrences of d, b, and e respectively in $\Sigma[t]$, the resulting subtree will have type t. The right side of figure 5 illustrates the procedure with G_3, E_3^4, and G^4. Together $\Sigma[t]$, G_u, and G^v work as a kind of disposal mechanism for the leaf E_u^v, providing it a place to which it can QR and returning a tree of type t.

Γ_4 consists of $m - n$ nested copies of Σ. When G has a Hamiltonian cycle, Γ_1 will contain $m - n$ leaves E_u^v coming from edges not used in the cycle. Γ_4's purpose is to act as a waste bin of sorts for these unused edges. Notice that "disposing" of a leaf E_u^v requires raising both a leaf G_u and a leaf G^v from Γ_3. As mentioned earlier, for each vertex v there are $\deg^-(v) - 1$ edges with head v and $\deg^+(v) - 1$ edges with tail v that do not occur in the cycle. Γ_3 therefore provides exactly the right number of leaves G_u and G^v to dispose of all the unused edges.

Figure 6 shows the final LF $\Gamma = \Gamma_4[\Gamma_3[\Gamma_2[\Gamma_1]]]$. Notice that Γ will always be linear in the size of the input, and can be constructed in polynomial time.

4.2 Correctness

It remains to show that the reduction is correct, i.e., that G has a Hamiltonian cycle iff the answer to the LF REPAIR-instance constructed in the previous

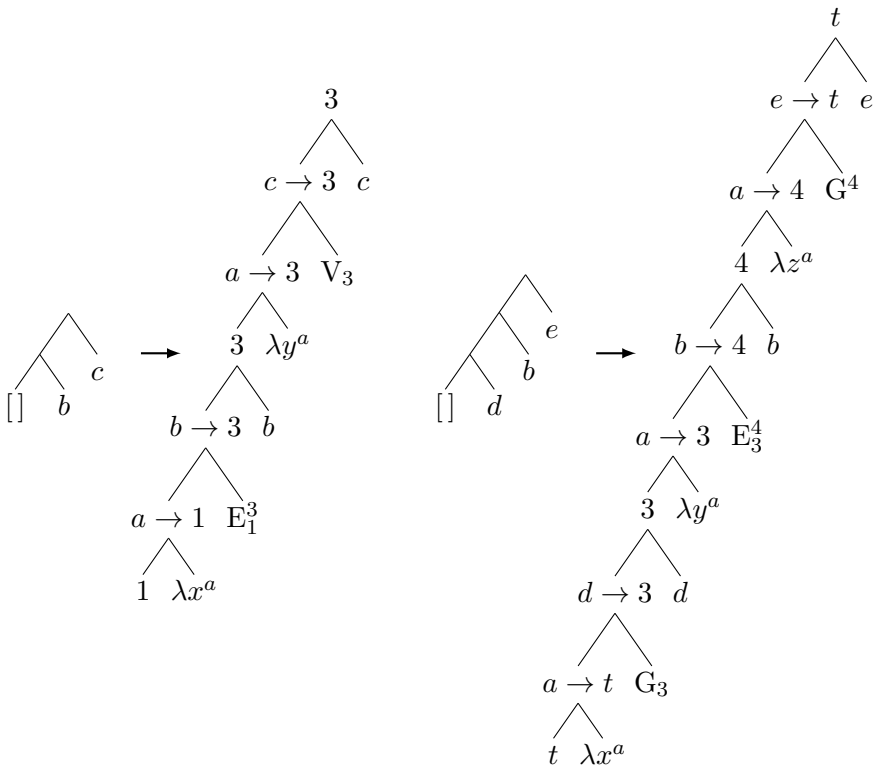


Figure 5: Simulating the traversal of an edge (left) and disposing of an unused edge (right). Interior nodes are annotated with the type of the subtree they dominate.

subsection is "yes".

First the easy direction.

Theorem 4.1. *If G has a Hamiltonian cycle, then there exists an LF $\Delta : t$ such that $\Gamma \to^*_{\mathrm{QR}} \Delta$.*

Proof. Let

$$(u_1, u_2), (u_2, u_3), \ldots, (u_{n-1}, u_n), (u_n, u_1)$$

be a Hamiltonian cycle in G. Without loss of generality, assume that $u_1 = 1$. We can derive an appropriate $\Delta : t$ from Γ as follows. Note that every QR step in the derivation below will leave a trace from $\mathbf{V}_a$.

First QR $\mathrm{E}_{u_1}^{u_2}$ and V_{u_2} directly below the lowest occurrence of b and c respectively in Γ_2 (compare the left side of figure 5). Then QR $\mathrm{E}_{u_2}^{u_3}$ and V_{u_3} directly below the *second*-lowest b and c respectively in Γ_2. Continue this process for the rest of the cycle, finishing with $\mathrm{E}_{u_n}^{u_1}$ and V_{u_1}.

Exactly $m - n$ leaves of the form E_u^v in Γ_1 are still waiting to QR, each one of them corresponding to an edge not used in the Hamiltonian cycle. For each vertex v, exactly $\deg^-(v) - 1$ of these leaves correspond to an edge with head v and $\deg^+(v) - 1$ to an edge with tail v.

For each b in Γ_4, QR one of the $m - n$ remaining leaves E_u^v's directly below it. Then QR an instance of G_u and G^v from Γ_3 directly below the adjacent occurrences of d and e, respectively (compare the right side of figure 5). Since the number of leaves G_u and G^v is $\deg^+(v) - 1$ and $\deg^-(u) - 1$ respectively, by the end of this process every G_u and G^v from Γ_3 will have undergone QR.

The LF resulting from these $3m - n$ applications of QR is the Δ we wanted. An easy check shows that it has type t. □

Now for the harder direction, which comes down to showing that the procedure in the previous proof is the only way to make Γ well typed.

Theorem 4.2. *Suppose that $\Gamma \to^*_{\mathrm{QR}} \Delta$ for some Δ of type t. Then G has a Hamiltonian cycle.*

Proof. By lemma 2.1, we can assume that Δ is normalized. We now make a series of claims about Δ. Below, I refer to a leaf that occurs in Γ as a "Γ-leaf."

(i) Every trace in Δ is of order one and therefore of atomic type. The claim follows from lemma 2.2 and the fact that all of Γ's leaves have order at most three. A consequence is that nothing in the derivation $\Gamma \to^*_{\mathrm{QR}} \Delta$, undergoes QR twice,

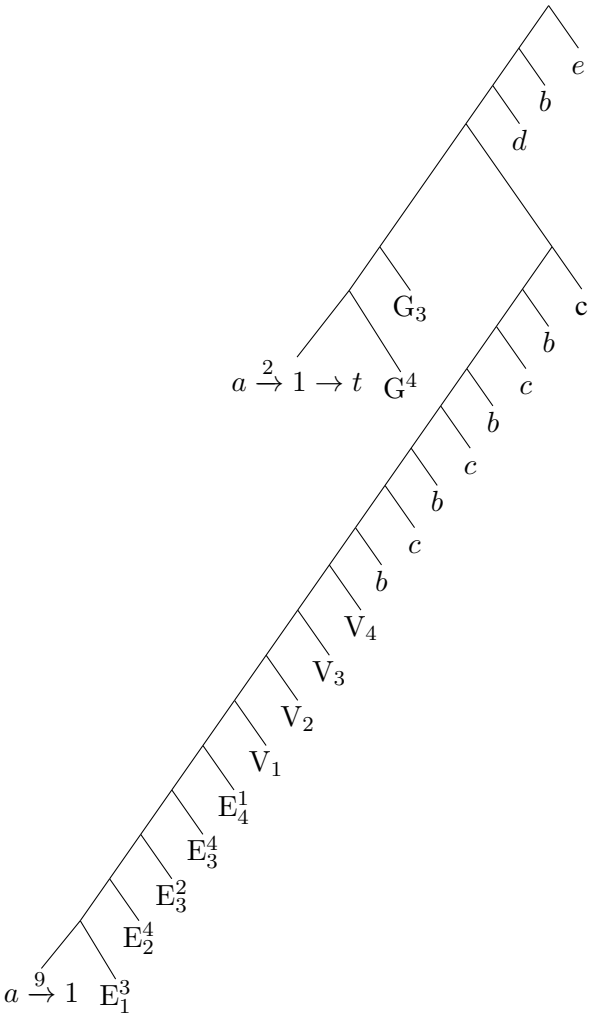


Figure 6: The LF REPAIR-instance for the digraph in figure 4.

since this would create a trace of order one next to an abstraction, contradicting the fact that Δ is normalized.

(ii) Every internal node of Δ has a type of order at most two. To prove the claim, suppose it were false and consider a maximally deep internal node of order greater than two. By (i), all abstractions are over variables of order one, so the node is not an abstraction (otherwise there would be a deeper internal node of the same order). It therefore has the form $\Theta \bullet \Lambda$, where Θ has type $\alpha\beta$, Λ has type α, and $\mathrm{ord}(\beta) > 2$. Since $\mathrm{ord}(\tau(\Theta)) = \mathrm{ord}(\alpha\beta) > 2$, Θ cannot be an internal node, so it must be a leaf. But each leaf of Δ is either a trace of order one or a Γ-leaf, and no Γ-leaf has the form $\alpha\beta$ where $\mathrm{ord}(\beta) > 2$. So no such internal node can exist.

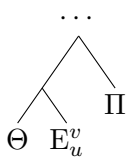
(iii) Δ contains no "double abstractions," i.e., it is impossible to write $\Delta = \Theta[\lambda x\,(\lambda y\,\Lambda)]$. Suppose otherwise, and consider a "double abstraction" of *minimal* depth, so that we can write Δ as $\Theta[\Pi \bullet \lambda x^{\alpha}\,(\lambda y^{\beta}\,\Lambda)]$. Because Δ is normalized, Π must have a type of the form $(\alpha\beta\gamma)\delta$. This means that Π's type has order at least three, and so by (ii) Π must be a leaf. But no leaf of Δ has the form $(\alpha\beta\gamma)\delta$, a contradiction.

(iv) If an abstraction $\lambda x^{\alpha}\,\Pi$ occurs in Δ, then $\Delta = \Theta[l \bullet \lambda x^{\alpha}\,\Pi]$ for some context Θ and leaf l of type E^{v}_{u}, V_{v}, G_{v}, or G^{v}. This follows from (i), (ii), (iii), and the fact that Δ is normalized. A consequence is that in a derivation $\Gamma \to^{*}_{\mathrm{QR}} \Delta$, the only elements that undergo QR are leaves of type E^{v}_{u}, V_{v}, G_{v}, or G^{v}.

(v) Every trace in Δ has type a. If x^{α} is a trace in Δ, then it must be bound by some abstraction, so by (iv) we can write Δ as $\Theta[l \bullet \lambda x^{\alpha}\,\Pi[x^{\alpha}]]$, where l is a Γ-leaf. Since Δ is normalized, $l = (\alpha\beta)\gamma$ for some types β, γ. Inspecting the leaves of Γ shows that α can only be a.

(vi) Call the sister of the mother of a node in Δ its *aunt*. Then the aunt of each leaf E^{v}_{u}, V_{v}, G_{v}, and

G^v in Δ is a Γ-leaf of type b, c, d or e, respectively. Consider, for example, a leaf E^v_u occurring in Δ with sister Θ and aunt Π, as shown below:



Θ would need order at least 4 to take

$$\mathrm{E}^v_u := (a \to u) \to b \to v$$

as an argument, but no node in Δ has order higher than 3. So Θ must have type $a \to u$ and $\Theta \bullet \mathrm{E}^v_u$ must have type $b \to v$. For Π to take $b \to v$ as an argument, it would need to have order at least 3. The only such elements of Δ are the Γ-leaves, but no Γ-leaf has a type of the form $(b \to v) \to \gamma$. So Π must have type b. Since all the traces in Δ have type a, the only nodes of type b in Δ are the m Γ-leaves coming from Γ_2 and Γ_4, and therefore Π is a Γ-leaf of type b, as we wanted to show. Similar arguments apply to the leaves of type V_v, G_v, and G^v. Since none of the leaves E^v_u, V_v, G_v, and G^v in Γ has an aunt of type b, a consequence of (vi) is that they must all have undergone QR in the derivation $\Gamma \to^*_{\mathrm{QR}} \Delta$.

The upshot of all these observations is that in the derivation $\Gamma \to^*_{\mathrm{QR}} \Delta$, every leaf of type E^u_v, V_v, G_v, and G^v undergoes QR *exactly once* (by (i) and (vi)), leaving a trace of type a (by (v)), and *nothing else* undergoes QR (by (iv)).

We can now extract a Hamiltonian cycle from Δ. Order the leaves E^v_u and V_v in Δ that QRed into Γ_2 by decreasing depth:

$$\mathrm{E}^{v_1}_{u_1}, \mathrm{V}_{w_1}, \mathrm{E}^{v_2}_{u_2}, \mathrm{V}_{w_2}, \ldots, \mathrm{E}^{v_n}_{u_n}, \mathrm{V}_{w_n}.$$

In order for Δ to be well-typed, it must be the case that

$$u_1 = v_n = w_n = 1$$

and $v_i = w_i = u_{i+1}$ for $1 \leq i < n$. So we can rewrite the sequence above as

$$\mathrm{E}^{u_2}_{u_1}, \mathrm{V}_{u_2}, \mathrm{E}^{u_3}_{u_2}, \mathrm{V}_{u_3}, \ldots, \mathrm{E}^{u_1}_{u_n}, \mathrm{V}_{u_1}.$$

Since $V_{u_1}, \ldots, \mathrm{V}_{u_n}$ are pairwise distinct, so are $u_1, \ldots, u_n$, and

$$(u_1, u_2), (u_2, u_3), \ldots, (u_n, u_1),$$

is therefore a Hamiltonian cycle in G. □

Putting theorems 3.4, 4.1, and 4.2 proves the main theorem of this paper:

Theorem 4.3. LF REPAIR *is NP-complete.*

Section 3 mentioned that LF REPAIR has the same time complexity as QRT-provability of abstraction-free sequents. So a corollary of the above is:

Corollary 4.1. *Provability of abstraction-free sequences in QRT is NP-hard.*

5 Conclusion

The main result of this paper is that finding well-typed logical forms using quantifier raising is NP-complete, and therefore belongs to a well-known class of intractable problems. Where does the intractability come from? The reduction in section 4 provides some insight. Notice that it:

(i) only used traces of a single atomic type,
(ii) only used types of order ≤ 3, which is the minimum needed for nontrivial applications of QR,
(iii) did not need parasitic scope (Barker, 2007),
(iv) did not need any elements to undergo QR multiple times.

As a consequence, even if one constrained QR by restricting the type or order of traces, or banning parasitic scope, or disallowing elements from undergoing QR multiple times, or imposing any combination of these constraints simultaneously, the reduction in the previous section would still be valid and the problem of finding a well-typed LF would remain NP-complete.

The reduction *does* help itself to a large inventory of atomic types, but this isn't essential. Given a single atomic type p, it's possible to encode all of the types in **A** in "unary", representing a as p, b as $p \to p$, c as $p \to p \to p$, etc. This will preserve the reduction. (This encoding comes with a small cost: it increases the order of the types used in the reduction by one).

So as far as worst-case time complexity is concerned, none of the constraints above are consequential. It seems like the true source of complexity, the true reason why finding LFs with QR is difficult, is simply the enormous number of derivational possibilities that arise when (i) large numbers of interacting scope-takers accumulate and (ii) there are few or no syntactic constraints on what QR can do.

A natural next question is whether a modification of the reduction in this paper can prove NP-completeness results for other scope-taking for-

malisms. The type-logical grammar NL_λ, for example, is a close relative of the grammar QRT from section 3, and should have a similar time complexity (Barker, 2019; Moot, 2020). Another good candidate are continuized CCGs (Barker and Shan, 2014; White et al., 2017).

Acknowledgments

I'm thankful to Dylan Bumford, Tim Hunter, and three anonymous reviewers for their advice.

References

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, UK.

Chris Barker. 2007. Parasitic Scope. *Linguistics and Philosophy*, 30:407–444.

Chris Barker. 2019. NL_λ as the Logic of Scope and Movement. *Journal of Logic, Language and Information*, 28:217–237.

Chris Barker. 2020. The Logic of Quantifier Raising. *Semantics and Pragmatics*, 30:1–40.

Chris Barker and Chung-chieh Shan. 2014. *Continuations and Natural Language*. Oxford University Press.

G. Edward Barton, Jr., Robert C. Berwick, and Eric Sven Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge, Massachusetts.

Rajesh Bhatt and Veneeta Dayal. 2007. Rightward Scrambling as Rightward Remnant Movement. *Linguistic Inquiry*, 38(2):287–301.

Danny Fox. 2000. *Economy and Semantic Interpretation*. Linguistic Inquiry Monographs 35. MIT Press, Cambridge, MA.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman and Company, New York.

Irene Heim. 1997. Predicates or Formulas? Evidence from Ellipsis. In *Proceedings of SALT 7*, pages 197–221.

Irene Heim and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Blackwell Publishers Ltd, Oxford, UK.

C.-T. James Huang. 1993. Reconstruction and the Structure of VP: Some Theoretical Consequences. *Linguistic Inquiry*, 24(1):103–138.

Thomas Krantz and Virgile Mobile. 2001. Encoding Hamiltonian Circuits into Multiplicative Linear Logic. *Theoretical Computer Science*, 266:987–996.

Robert May. 1977. *The Grammar of Quantification*. PhD thesis, Massachusetts Institute of Technology.

Richard Moot. 2020. Proof-theoretic Aspects of NL_λ. *Preprint*, arXiv:2010.12223.

Christos H. Papadimitriou. 1994. *Computational Complexity*. Addison-Wesley, Reading, MA.

Ethan Poole. 2024. (Im)possible Traces. *Linguistic Inquiry*, 55(2):287–326.

Ian Pratt-Hartmann. 2010. Computational Complexity in Natural Language. In *The Handbook of Computational Linguistics and Natural Language Processing*, pages 43–73. Wiley-Blackwell.

Michael White, Simon Charlow, Jordan Needle, and Dylan Bumford. 2017. Parsing with Dynamic Continuized CCG. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 71–83. Association for Computational Linguistics.