Deforesting Logical Form

Zhong Chen and John T. Hale

Department of Linguistics
Cornell University
Ithaca, NY 14853-4701, USA
{zc77,jthale}@cornell.edu

Abstract. This paper argues against Logical Form (LF) as an intermediate level of representation in language processing. We apply a program transformation technique called *deforestation* to demonstrate the inessentiality of LF in a parsing system that builds semantic interpretations. We consider two phenomena, Quantifier Raising in English and *Wh*-movement in Chinese, which have played key roles in the broader argument for LF. Deforestation derives LF-free versions of these parsing systems. This casts doubt on LF's relevance for processing models, contrary to suggestions in the literature.

1 Introduction

It is the business of the computational linguist, in his role as a cognitive scientist, to explain how a physically-realizable system could ever give rise to the diversity of language behaviors that ordinary people exhibit. In pursuing this grand aspiration, it makes sense to leverage whatever is known about language itself in the pursuit of computational models of language use. This idea, known as the Competence Hypothesis [5], dates back to the earliest days of generative grammar.

Hypothesis 1 (Competence Hypothesis). *A reasonable model of language use will incorporate, as a basic component, the generative grammar that expresses the speaker-hearer's knowledge of the language.*

The Competence Hypothesis is the point of departure for the results reported in this paper. Section 4 and 5 show how a syntactic theory that incorporates a level of *Logical Form* (LF) can be applied fairly directly in a physically-realizable parser. These demonstrations are positive results about the viability of certain kinds of transformational grammars in models of language use. While not strictly novel, such positive results are important for cognitive scientists and others who wish to maintain the Competence Hypothesis about the grammar-parser relationship. In particular, the parser described in Section 4.2 handles a variety of English quantifier-scope examples that served to motivate LF when that level was first introduced. Section 5.2 extends the same technique to Chinese *wh*-questions, a case that has been widely taken as evidence in favor of such a level.

Sections 4.3 and 5.3 then apply a general program transformation technique called *deforestation* to each parser. Deforestation, as championed by Wadler [25] and described in more detail in Section 3, is a general method for getting rid of intermediate data structures in functional programs. In the current application, it is the LF representations that are eliminated. The resultant parsing programs do not construct any such representations, although they do compute the same input-output function. The outcome of deforestation is a witness to the fact that the parser need not construct LFs in order to do the job the grammar specifies. This inessentiality suggests that representations at the LF level should not be viewed as causally implicated in human sentence comprehension, despite suggestions to the contrary in the literature. For instance, Berwick and Weinberg suggest that:

> There is good evidence that the data structures or units of representation posited by theories of transformational grammar are actually implicated causally in online language processing. [1, 197]

Consider this claim in relation to the positive and negative results. If one were to observe a characteristic pattern of errors or response times across two language understanding tasks that vary only in their LF-handling requirements, then a good theory of these results might involve some kind of LF parser whose computational actions causally depend on LF representations. The positive results of Sections 4.2 and 5.2 show how this could be done. However, the negative results to be presented in Sections 4.3 and 5.3 imply that the very same kind of model can, through deforestation, be reformulated to avoid calculating with precisely the LF representations that Berwick and Weinberg suggest are causally-implicated in language understanding. This is a paradox. To escape it, such evidence, if it exists, must instead be viewed as evidence for cognitive processes that calculate the language relationships that LF specifies, rather than as evidence for LF per se.

The conjunction of these negative and positive results holds special significance for linguistics because it casts doubt on two widely held hypotheses.

Hypothesis 2 (No levels are irrelevant). *To understand a sentence it is first necessary to reconstruct its analysis on each linguistic level. [4, 87]*

Hypothesis 3 (LF Hypothesis). *LF is the level of syntactic representation that is interpreted by semantic rules. [22, 248]*

We present counter-examples where understanding does not require reconstruction of the LF level, and where semantic rules would not apply at LF.

With the broader significance of the question in mind, Section 2 identifies the sense of the term "Logical Form" at issue. A brief introduction to deforestation follows in Section 3. Sections 4.2 and 5.2 then define two parsers whose program text closely mirrors the grammar. Sections 4.3 and 5.3 discuss how deforestation applies to both programs. Section 6 makes some concluding remarks, speculating on relationships between this deforestation result and recent developments in linguistic theory.

2 What Is Meant by LF

2.1 LF Is a Level of Representation

Outside of the transformational generative grammar community, the words "Logical Form" refer to a symbolization of a sentence's meaning in some agreed-upon logic. However, within this community, LF is a technical term that refers to a specific level of representation — an obligatory subpart of well-formed structural descriptions. Hornstein writes,

> LF is the level of representation at which all grammatical structure relevant to semantic interpretation is provided. [15, 3]

Hornstein describes LF as being introduced in response to a realization that "surface structure cannot adequately bear the interpretive load expected of it" [15, 2]. Thus, in the transition from Chomsky's Revised Extended Standard Theory to theories based on Government and Binding, an interpretive semantics is retained, while the precise level of representation being most directly interpreted is altered.

2.2 LF Is an Interface Level

In the course of applying LF to problems of quantifier scope, May [23] makes clear that LF is a kind of interface level by writing,

> We understand Logical Form to be the interface between a highly restricted theory of linguistic form and a more general theory of natural language semantics and pragmatics. [23, 2]

The heavy lifting will be done by this more general theory. May emphasizes the mismatch between the limited capabilities of the rules affecting LF, compared to those that will be required to turn LFs into bona fide semantic representations. The latter occupies a different level, LF$'$, pronounced "LF prime".

> Representations at LF$'$ are derived by rules applying to the output of sentence grammar ... Since the rules mapping Logical Form to LF$'$ are not rules of core grammar, they are not constrained by the restrictions limiting the expressive power of rules of core grammar. [23, 27]

Representations at LF$'$ are subject to "the recursive clauses of a Tarskian truth-condition theory" [23, 26]. This is to be contrasted with representations at LF that are not. These representations are phrase markers, just like the immediate constituency trees at surface structure. This sense of LF has been repeatedly invoked in Chomsky's recent work [7, fn 20][8, fn 11].

With this LF–terminology clarified, subsequent sections go on to apply functional programming techniques to define a surface structure parser and extend it with a transformational rule, Move-α, to build LF representations. We investigate two cases of this rule, Quantifier Raising in Section 4 and *Wh*-movement in Section 5. The relationship between these levels is depicted in Figure 1.

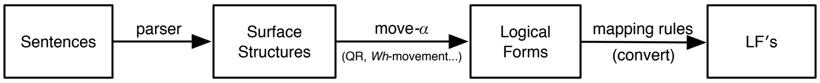


Fig. 1. The interlevel relationships from a sentence to LF′s

3 Deforestation

Deforestation is a source-to-source program transformation used to derive programs that are more efficient in the sense that they allocate fewer data structures that only exist ephemerally during a program's execution. Wadler provides a small deforestation example [25] which we repeat below as Listing 1 in the interest of self-contained presentation.

In this example, the main function sumSquares calculates the value $\sum_{x=1}^{n} x^2$.

```
let rec upto m n = match (m>n) with
    true -> []
  | false -> m::(upto (m+1) n)
let square x = x*x
let rec map f xs = match xs with
    [] -> []
  | y::ys -> (f y)::(map f ys)
let sum xs =
  let rec sumAux a = function
      [] -> a
    | x::rest -> sumAux (a+x) rest
  in
    sumAux 0 xs
let sumSquares x = sum (map square (upto 1 x))
```

Listing 1. Classic deforestation example

The manner in which sumSquares actually calculates the sum of squares is typical of functional programs. Intuitively, it seems necessary that the upto function must first create the list $[1, 2, \ldots, n]$. Then the map function maps square over this list, yielding a new list: $[1, 4, \ldots, n^2]$. These lists are both kinds of intermediate data. They are artifacts of the manner in which the function is calculated. Indeed, the sum function transmutes its input list into a single integer, $\frac{1}{6}\left(2n^3 + 3n^2 + n\right)$. This makes clear that the intermediate lists have a limited lifetime during the execution of the function. The idea of deforestation is to translate programs like the one in Listing 1 into programs like the one in Listing 2.

```
let sumSquaresDeforested x =
  let rec h a m n =
    if m > n
    then a
    else h (a + (square m)) (m+1) n
  in
    h 0 1 x
```

Listing 2. Deforested program does not allocate intermediate lists

The program in Listing 2 recurs on an increasing integer m and scrupulously avoids calling the Caml list constructor (::). The name "deforestation" is evocative of the fact that intermediate trees can be eliminated in the same way as lists. Over the years, many deforestation results have been obtained in the programming languages community. The typical paper presents a system of transformation rules that converts classes of program fragments in an input language into related fragments in the output language. An example rule from Wadler's transformation scheme T is given is given below.

$$\begin{aligned} T\,[\![f\,t_1\ldots t_k]\!] &= T\,[\![t[t_1/v_1,\ldots,t_k/v_k]]\!] \\ &\quad \text{where } f \text{ is defined by } f\,v_1\ldots v_k = t \end{aligned} \tag{4}$$

The deforestation rule above essentially says that if the definition of a function f is a term t in the input language, then the deforestation of f applied to some argument terms t_1 through t_k is simply the replacement of the function by its definition, subject to a substitution of the formal parameters $v_1,\ldots,v_k$ by the actual parameters $t_1,\ldots,t_k$. This kind of program transformation is also known as the unfold transformation [3]. Wadler provides six other rules that handle other features of his input language; the original paper [25] should be consulted for full details.

In the present application, neither automatic application nor heightened efficiency is the goal. Rather, at issue is the linguistic question whether or not a parser that uses LF strictly needs to do so. Sections 4.3 and 5.3 argue for negative answers to this question.

4 Quantifier Raising

Quantifier Raising (QR) is an adjunction transformation[1] proposed by May [23] as part of a syntactic account of quantifier scope ambiguities. May discusses the following sentence (5) with two quantifiers *every* and *some*. It has only one reading; the LF′ representation is given in (6).

(5) Every body in some Italian city met John.
'There is an Italian city, such that all the people in it met John.'

(6) $\exists x\,(\forall y\,((\text{Italian-city}\,(x)\,\&\,(\text{body-in}\,(y,x) \rightarrow \text{met-John}\,(y)))))$

4.1 Proper Binding

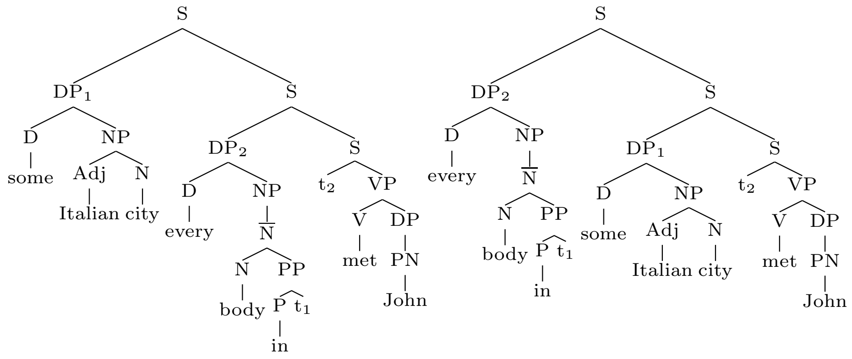
When May's QR rule freely applies to a sentence with multiple quantifiers, it derives multiple LFs that intuitively correspond to different quantifier scopes.

[1] The word "transformation" here is intended solely to mean a function from trees to trees. Adjunction is a particular kind of transformation such that, in the result, there is a branch whose parent label is the same as the label of the sister of the re-arranged subtree.

Sometimes this derivational ambiguity correctly reflects scope ambiguity. However, in semantically unambiguous cases like (5), the QR rule overgenerates. A representational constraint, the Proper Binding Condition (PBC) [12] helps to rule out certain bad cases that would otherwise be generated by unfettered application of QR.

Principle 1 (The Proper Binding Condition on QR). *Every raised quantified phrase must c-command*[2] *its trace.*

Perhaps the most direct rendering of May's idea would be a program in which LFs are repeatedly generated but then immediately filtered for Proper Binding. One of the LFs for example (5), shown in Figure 2(b) below, would be ruled out by the PBC because DP_1 does not c-command its trace t_1.



(a) LF_1 (b) $^*\mathrm{LF}_2$

Fig. 2. The logical form of the Quantifier Raising example (5)

This kind of "Generate-and-Test" application of Proper Binding is obviously wasteful. There is no need to actually create LF trees that are doomed to failure. Consider the origins of possible Proper Binding violations. Because QR stacks up subtrees at the front of the sentence, the main threat is posed by inopportune choice of subtree to QR. A precedence-ordered list of quantified nodes is destined to fail the PBC just in case that list orders a super-constituent before one of its sub-constituents. This observation paves the way for a change of representation from concrete hierarchical trees to lists of tree-addresses[3]. Principle 2 below reformulates the PBC in terms of this alternative representation.

[2] C-command is a relationship defined on tree nodes in a syntactic structure. A node α c-commands another node β if the first node above α contains β.

[3] The "Gorn address" is used here. It is a method of addressing an interior node within a tree [14].) Here we illustrate the Gorn address as an integer list. The Gorn address of the tree root is an empty list [] with the first child [0] and the second child [1]. The j-th child of the node with the Gorn address $[i]$ has an address $[i, j-1]$.

Principle 2 (Linear Proper Binding Condition on quantified phrases) *Let $L = n_1, \ldots, n_m$ be a list of tree-node addresses of quantified phrases. L will violate the Proper Binding Condition if any address n_i is a prefix of n_j $(i < j)$.*

The tree-addresses of *every body in some Italian city* and *some Italian city* in the surface structure tree are $[0]$ and $[0, 1, 0, 1, 1]$ respectively. The first address is the prefix of the latter. A list of quantified nodes $[[0], [0, 1, 0, 1, 1]]$, indicating that *every body* outscopes *some Italian city*, will violate Principle 2. Recognizing this allows a parser to avoid constructing uninterpretable LFs like the one in Figure 2(b). Note that the effect of this linear PBC is exactly the same as that of its hierarchical cousin, Principle 1.

The following sections take up alternative implementations of the function from sentences to LF′ representations. One of these implementations can be derived from the other. The first implementation obtains LF′s after building LFs. The second one does not allocate any intermediate LF but instead calculates LF′s directly based on surface structure analyses.

4.2 The Quantifier Raising Implementation

The Appendix presents a small Context Free Grammar used in a standard combinator parser[4] that analyzes example (5).

The LF-using implementation takes each PBC-respecting quantifier ordering, applies QR, then converts the resulting LF into an LF′ formula. Illicit LFs, such as the one in Figure 2(b), are never built due to the linear PBC. Raised quantified phrases are translated into pieces of formulas using two key rules:

$$\begin{array}{lll} \textit{some Italian city} \ldots & \Rightarrow \exists x\ (\text{Italian-city}(x)\ \&\ \ldots) \\ \textit{every body in x} \quad \ldots & \Rightarrow \forall y\ (\text{body-in}(y, x)\ \rightarrow \ldots) \end{array}$$

All these procedures can be combined in a LF parser as in Listing 3.

```
let withLF ss = Seq.map convert (Seq.map qr (candidate_quantifier_orderings ss))
```

Listing 3. QR analyzer with plug-in that uses LF

The higher-order function `Seq.map` applies its first argument to every answer in a stream of alternative answers. Our program faithfully calculates interactions between the constraints that May's competence theory specifies. Its output corresponds to the LF′ representation in (6).

```
# Seq.iter formula (analyze withLF "every body in some italian city met john");;
'exists v31[forall v32[(italian-city(v31) & (body-in(v32,v31) -> met-john(v32)))]]'
- : unit = ()
```

[4] A *combinator* is a function that takes other functions as arguments, yielding more interesting and complicated functions just through function-application rather than any kind of variable-binding [10][11]. Parser combinators are higher-order functions that put together more complex parsers from simpler ones. The parsing method is called combinatory parsing [2][13].

4.3 Deforesting LF in Quantifier Raising

Executing analyze withLF allocates an intermediate list of quantifier-raised logical forms. These phrase markers will be used as the basis for LF′ representations by convert. The existence of an equivalent program, that does not construct them, would demonstrate the inessentiality of LF.

```
let f_may (ss,places) = ⟦ convert (qr (ss,places)) ⟧^deforest
let withoutLF ss = Seq.map f_may (candidate_quantifier_orderings ss)
```

Listing 4. LF-free quantifier analyzer

The key idea is to replace the composition of the two functions convert and qr into one deforested function f_{may} which does the same thing without constructing any intermediate LFs. Algorithm 1 provides pseudocode for this function, which can be obtained via a sequence of deforestation steps[5], as shown in Table 1. It refers to rules defined by Wadler [25], including the unfold program transformation symbolized in (4). The implementation takes advantage of the fact that the list of quantified nodes, QDPplaces, can be filtered for compliance with the linear PBC ahead of time.

Table 1. Steps in the *deforestation* of QR example (5)

Wadler's Number	Action in the program
3	unfold a function application convert with its definition
6	unfold a function application qr with its definition
7	broadcast the inner case statement of a QDP outwards
5	simplify the matching on a constructor. Use fact: NP is the second daughter of DP
5	get rid of the matching on a Some constructor
Not relevant	knot-tie in the outermost match. Realize we started by translating (convert (qr (ss,places))) == f_{may} (ss,places)

Since the PBC applies equally well to lists of "places", it is no longer necessary to actually build tree structures. The deforested procedure recurs down the list of QDPplaces (Line 5); applies QR rules according to their quantifier type (Line 10 or 16) and turns phrases into pieces of formulas using predicateify (Line 7 or 13). No phrase markers are moved and no LF representations are built.

5 *Wh*-Movement

Apart from quantifier scope, perhaps the most seminal application of LF in syntax is motivated by *wh*-questions. The same deforestation techniques are equally applicable to this case. They similarly demonstrate that a LF-free semantic

[5] The full derivation of the deforestation is omitted in the paper due to space limits. The rule numbers used in Table 1 is consistent with Wadler [25, 238, Figure 4].

Algorithm 1. Pseudocode for the deforested f_{may}

```
function f_may(ss, QDPplaces)
    if no more QDPplaces then
        predicateify(ss)
    else
        examine the next QDPplace qdp
        if qdp =    DP    then
                   /  \
                  D    NP
                  |
                every
            let restrictor = predicateify(NP)
            let v be a fresh variable name
            let body = ss with qdp replaced by an indexed variable
            ∀ v restrictor(v) → f_may (body, remaining QDPplaces)
        end if
        if qdp =    DP    then
                   /  \
                  D    NP
                  |
                some
            let restrictor = predicateify(NP)
            let v be a fresh variable name
            let body = ss with qdp replaced by an indexed variable
            ∃ v restrictor(v) ∧ f_may (body, remaining QDPplaces)
        end if
    end if
end function
```

interpreter may be obtained by deforesting a straightforward implementation that does use LF. To understand this implementation, a bit of background on *wh*-questions is in order.

The standard treatment of *wh*-questions reflects languages like English where a *wh*-word overtly moves to a clause-initial position when forming an interrogative. However, it is well-known that in Chinese, the same sorts of interrogatives leave their *wh*-elements in clause-internal positions. This has come to be known as "*wh*-in-situ" [18].

Although the *wh*-word remains in its surface position, some syntacticians have argued that movement to a clause-initial landing site does occur, just as in English, but that movement is not visible on the surface because it occurs at LF [16][17]. This analysis thus makes crucial use of LF as a level of analysis.

5.1 The ECP and the Argument/Adjunct Asymmetry

At the heart of the argument for LF is the idea that this "covert" movement in Chinese interrogatives creates ambiguity. Example (7) below is a case in point: if Move-α were allowed to apply freely, then both readings, (7a) and (7b) should be acceptable interpretations of the sentence. In actuality only (7a) is acceptable.

(7)

ni	xiangzhidao	wo	weishenme	mai	shenme?
you	wonder	I	why	buy	what

a. 'What is the x such that you wonder why I bought x?'

b. *'What is the reason x such that you wonder what I bought for x?'

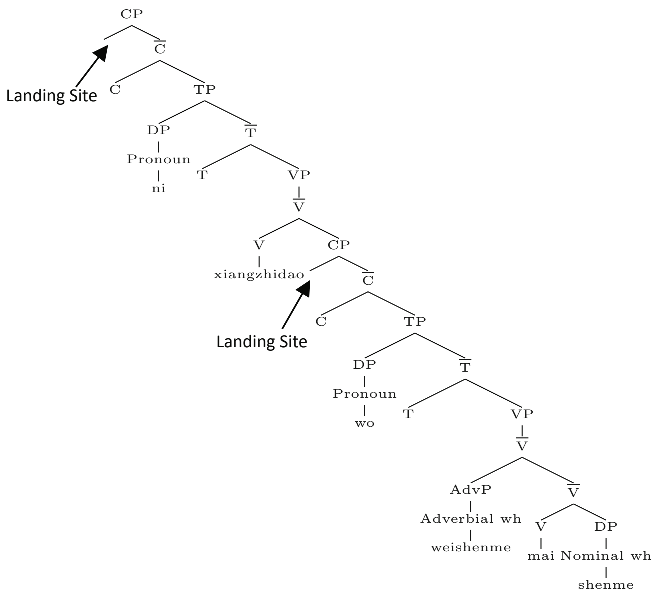


Fig. 3. The surface structure of the Chinese *Wh*-movement example (7)

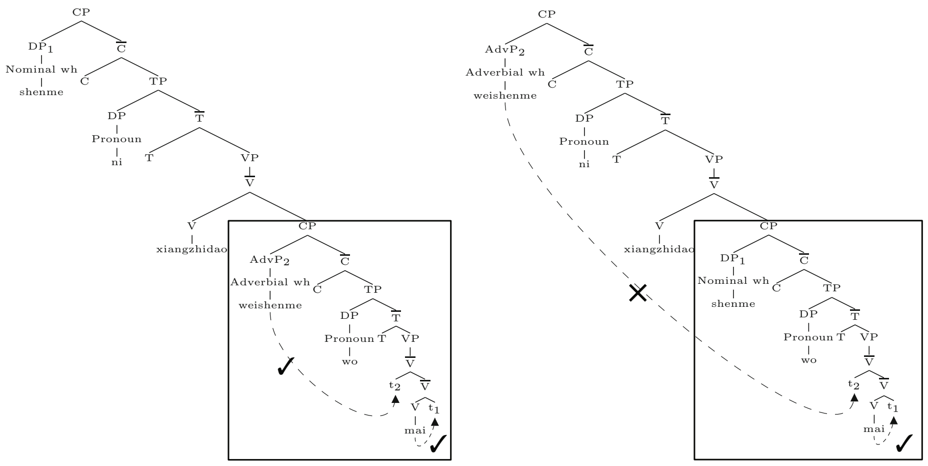
The verb *xiangzhidao* 'wonder' in (7) forms a subordinate question. The presence of two alternative landing sites, shown in Figure 3, allows *Wh*-movement to derive two different LF possibilities. These two LFs, illustrated in Figure 4, correspond to alternative readings. However, only reading (7a) is acknowledged by Chinese speakers. Sentence (7) does not question the reason for buying as in (7b). Rather, it is a direct question about the object of *mai* 'buy'.

Huang argues that Empty Category Principle (ECP) [6] correctly excludes the unattested interpretation as shown in Figure 4(b).

Principle 3 (The Empty Category Principle). *A non-pronominal empty category (i.e., trace) is properly governed by either a lexical head or its antecedent.*[6]

Figure 4 illustrates the ECP's filtering action on example (7). In this Figure, dotted arrows indicate government. In the ECP-respecting LF shown in Figure 4(a), the trace t_1 of the moved *wh*-word *shenme* is an empty category lexically governed by the verb *mai* 'buy'. Trace t_2 is antecedent governed. In the ECP-failing LF of Figure 4(b), the trace of *shenme* t_1, is also lexically governed. However, *weishenme*'s trace, t_2 is left ungoverned. As an adjunct, it is not lexically governed. Nor is it antecedent governed — the nearest binder *weishenme* lies beyond a clause-boundary.

[6] The antecedent can be a moved category (i.e., *wh*-phrase). We follow Huang et al [18] in using the classical, "disjunctive" version of the ECP.



(a) LF_1 (b) $*LF_2$

Fig. 4. The logical form of the Chinese *Wh*-movement example (7)

Like the Proper Binding Condition, the ECP can also be reformulated as a constraint on lists; in this case lists of *wh* elements. The key requirement is that only *wh*-arguments should outscope other *wh*-phrases.

Principle 4 (Linear Empty Category Principle on *wh*-questions). *Let $L = n_1, \ldots, n_m$ be a scope-ordered list of wh-elements where n_1 has the widest scope and n_m has the narrowest. L will violate the Empty Category Principle if any wh-adjunct is at the position n_i and $i < m$.*

The correctness of Principle 4 derives from the fact that moved *wh*-elements always end up at the edge of a clause. If a *wh*-adjunct scopes over another *wh*-phrase, it must have crossed a clause boundary in violation of the ECP.

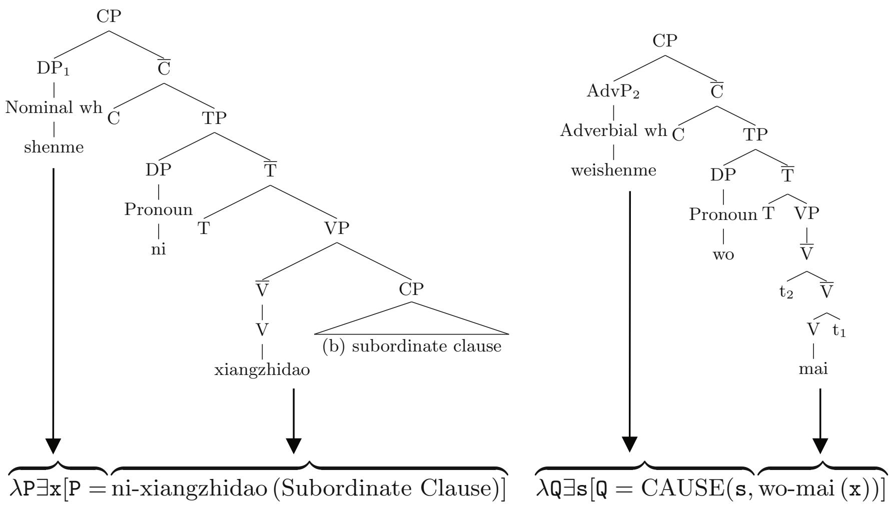
We postulate LF′ representations as in (8) in which *wh*-phrases denote sets of propositions [21]. The *wh*-argument *shenme* scopes over the *wh*-adjunct *weishenme*. The answer to *weishenme* is a set of propositions which CAUSEs the action *wo-mai* to happen as part of the state *s*.

(8) $\lambda P \exists x\, (P = \text{ni-xiangzhidao}\, (\lambda Q \exists s\, (Q = \text{CAUSE}\, (s, \text{wo-mai}\, (x)))))$

5.2 The *Wh*-Movement Implementation

The implementation of *Wh*-movement is mostly analogous to QR in Section 4.2. We define a combinator parser for the Chinese fragment in the Appendix. A *Wh*-movement function fills a landing site with a *wh*-phrase and creates a trace. A filter applies the ECP to the derived LFs.

This Chinese parser differs from the QR example in its ability to handle subordinate clauses. It obtains LF′ representations for the main clause and the



(a) Main clause (b) Subordinate clause

Fig. 5. cpconvert translates LF into LF′ of the example (7)

subordinate clause separately and then encapsulates them together, as shown in Figure 5.

The cpconvert function applies two mapping rules:

$$\begin{aligned} &\textit{shenme} &&\Rightarrow \lambda P \exists x\ (P = \ldots x \ldots) \\ &\textit{weishenme} &&\Rightarrow \lambda Q \exists s\ (Q = \text{CAUSE}\,(s, \ldots)) \end{aligned}$$

This function plugs-in to yield an LF-using analyzer as shown in Listing 5.

```
let withCHSLF ss = Seq.map cpconvert (Seq.map mvwh (candidate_whps_orderings ss))
```

Listing 5. *wh*-question analyzer with plug-in that uses LF

This analyzer correctly finds the LF′ shown in (8).

```
# Seq.iter formula (analyzeCHS withCHSLF "ni xiangzhidao wo weishenme mai shenme");;
‘lambda h[exists v6[h=[ni-xiangzhidao(lambda i[exists s[i=[CAUSE(s,wo-mai(v6))]]])]]]’
- : unit = ()
```

5.3 Deforesting LF in *Wh*-Movement

Executing the program in Section 5.2 causes intermediate *wh*-moved LF trees to be created. This is similar to the QR program discussed in Section 4.2 which allocates quantifier-raised structures. Using the same deforestation techniques, an equivalent program that does not allocate these intermediate data structures can be obtained.

```
let $f_{\text{huang}}$ (ss, places) = $[\![$ cpconvert (mvwh (ss,places)) $]\!]^{deforest}$
let withoutCHSLF ss = Seq.map $f_{\text{huang}}$ (candidate_whp_orderings ss)
```

Listing 6. LF-free analyzer for Chinese *wh*-questions

The *Wh*-movement function mvwh takes a pair consisting of a surface structure and an ECP-compliant list of *wh*-element addresses. After deforestation, the resultant function f_{huang} in Algorithm 2 is similar to f_{may} but also has the ability to handle complex embedded sentences. Apart from deciding the type of *wh*-phrase (Line 6 and 20), an additional condition detects whether the current structure is complex (Line 10 and 24). If it is, the program obtains the LF′ representation for the first clause then recursively works on the subordinate clause and the remaining *wh*-phrases (Line 17 and 31). Table 2 lists the deforestation steps used in the derivation of this deforested f_{huang}.

Algorithm 2. Pseudocode for the deforested f_{huang}

```
 1: function $f_{\text{huang}}(ss, \text{WHPplaces})$
 2:     if no more WHPplaces then
 3:         predicateify($ss$)
 4:     else
 5:         examine the next WHPplace $whp$
 6:         if $whp =$      DP       then
                             |
                        Nominal wh
                             |
                          shenme
 7:             let $v$ be a fresh variable name
 8:             let $P$ be a fresh variable name
 9:             let body = ss with $whp$ replaced by an indexed variable
10:             if body has no subordinate clause then
11:                 $\lambda P \exists v . P = f_{\text{huang}}$ (body, remaining WHPplace)
12:             else
13:                 let $Q$ be a fresh variable name
14:                 let $\text{cp}_1$ = body with the subordinate clause replaced by $Q$
15:                 let $\text{cp}_2$ = the subordinate clause of body
16:                 let WHPplaces′ = the WH-phrase places of the $\text{cp}_2$
17:                 $\lambda P \exists v . P = (\lambda Q . \texttt{predicateify}(\text{cp}_1))\,(f_{\text{huang}}(\text{cp}_2, \text{WHPplaces}''))$
18:             end if
19:         end if
20:         if $whp =$     AdvP      then
                             |
                        adverbial wh
                             |
                        weishenme
21:             let $s$ be a variable name for the state/event
22:             let $P$ be a fresh variable name
23:             let body = ss with $whp$ replaced by an indexed variable
24:             if body has no subordinate clause then
25:                 $\lambda P \exists s . P = \text{CAUSE}(\text{s}, f_{\text{huang}}(\text{body, remaining WHPplaces}))$
26:             else
27:                 let $Q$ be a fresh variable name
28:                 let $\text{cp}_1$ = body with the subordinate clause replaced by $Q$
29:                 let $\text{cp}_2$ = the subordinate clause of body
30:                 let WHPplaces′ = the WH-phrase places of the $\text{cp}_2$
31:                 $\lambda P \exists s . P = (\lambda Q . \text{CAUSE}(\text{s}, \texttt{predicateify}(\text{cp}_1)))\,(f_{\text{huang}}(\text{cp}_2, \text{WHPplaces}''))$
32:             end if
33:         end if
34:     end if
35: end function
```

Table 2. Steps in the *deforestation* of *Wh*-movement example (7)

Wadler's Number	Action in the program
3	unfold a function application cpconvert with its definition
6	unfold a function application mvwh with its definition
7	broadcast the inner case statement of a WHP outwards
5	simplify the matching on a constructor; set the subordinate clause as the current structure; update WHP addresses accordingly.
5	get rid of the matching on a Some constructor
Not relevant	a knot-tie in the outermost match. We started by translating (cpconvert (mvwh (ss,places))) == f_{huang} (ss,places)

6 Conclusion

Parsing systems initially defined with the help of LF can be re-formulated so as to avoid constructing any LFs, or so it turns out in the well-known cases of English quantifier-scope and Chinese *wh*-questions. These findings suggest that LF may not in fact be an essential level of representation in a parser, because it can always be deforested away. They are consistent with the alternative organization depicted in Figure 6.

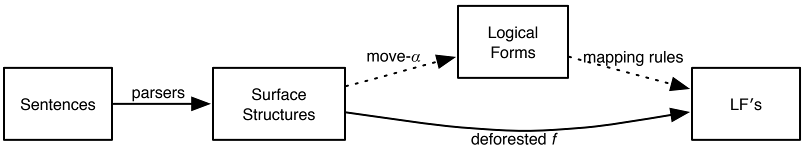


Fig. 6. The interlevel relationships from a sentence to LF′s

Of course, it remains to be seen if a facet of the LF idea will be uncovered that is in-principle un-deforestable. The positive results suggest that such an outcome is unlikely because the central elements of May and Huang's proposal are so naturally captured by the programs in Listings 3 and 5. In the end, there is no formal criterion for having adequately realized the spirit of a linguistic analysis in an automaton. The most we can ask is that the grammar somehow be recognizable within the parser (Hypothesis 1).

Nevertheless, the result harmonizes with a variety of other research. For instance, our method uses lists of quantified phrases or *wh*-phrases in a way that is reminiscent of Cooper Storage [9]. Cooper storage stores the denotation of quantified phrases or *wh*-phrases and retrieves them in a certain order. The linear versions of PBC and ECP could be construed as constraints on the order of retrieving them from the store.

In addition, our method is consistent with the work of Mark Johnson, who uses the fold/unfold transformation to, in a sense, deforest the S-structure and D-structure levels of representation out of a deductive parser [20].

It also harmonizes with the research program that seeks directly-compositional competence-grammars [19][24]. The present paper argues only that LF is unnecessary in the processor, leaving open the possibility that LF might be advantageous in the grammar for empirical or conceptual reasons. Proponents of directly compositional grammars have argued that the available evidence fails to motivate LF even in competence. If they are right, a fortiori there is no need for LF in performance models.

Acknowledgements

The authors would like to thank Hongyuan Dong, Scott Grimm, Tim Hunter, Graham Katz, Greg Kobele, Alan Munn and Michael Putnam and Chung-Chieh Shan for helpful feedback on this material. They are not to be held responsible for any errors, misstatements or other shortcomings that persist in the final version.

References

1. Berwick, R.C., Weinberg, A.S.: The Grammatical Basis of Linguistic Performance. MIT Press, Cambridge (1984)
2. Burge, W.H.: Recursive Programming Techniques. Addison-Wesley, Reading (1975)
3. Burstall, R., Darlington, J.: A transformation system for developing recursive programs. Journal of the Association for Computing Machinery 24(1), 44–67 (1977)
4. Chomsky, N.: Syntactic structures. Mouton de Gruyter, Berlin (1957)
5. Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, Cambridge (1965)
6. Chomsky, N.: Lectures on Government and Binding. Foris, Dordrecht (1981)
7. Chomsky, N.: Approaching UG from below. In: Sauerland, U., Gärtner, H.M. (eds.) Interfaces + recursion = language?: Chomsky's minimalism and the view from syntax-semantics, pp. 1–29. Mouton de Gruyter, Berlin (2007)
8. Chomsky, N.: On phases. In: Freidin, R., Otero, C., Zubzarreta, M.-L. (eds.) Foundational Issues in Linguistic Theory: Essays in Honor of Jean-Roger Vergnaud, pp. 133–166. MIT Press, Cambridge (2008)
9. Cooper, R.H.: Montague's Semantic Theory and Transformational Syntax. Ph.D. thesis, Umass (1975)
10. Curry, H.B., Feys, R.: Combinatory Logic, vol. 1. North-Holland, Amsterdam (1958)
11. Curry, H.B., Hindley, J.R., Seldin, J.P.: Combinatory Logic, vol. 2. North-Holland, Amsterdam (1972)
12. Fiengo, R.: Semantic Conditions on Surface Structure. Ph.D. thesis, MIT, Cambridge (1974)
13. Frost, R., Launchbury, J.: Constructing natural language interpreters in a lazy functional language. The Computer Journal 32(2), 108–121 (1989)

14. Gorn, S.: Explicit definitions and linguistic dominoes. In: Hart, J., Takasu, S. (eds.) Systems and Computer Science, University of Toronto Press (1967)
15. Hornstein, N.: Logical Form: From GB to Minimalism. Blackwell, Oxford (1995)
16. Huang, C.T.J.: Logic relations in Chinese and the theory of grammar. Ph.D. thesis, MIT, Cambridge; edited version published by Garland, New York, 1998 (1982)
17. Huang, C.T.J.: Move wh in a language without wh-movement. The linguistic review 1, 369–416 (1982)
18. Huang, C.T.J., Li, Y.H.A., Li, Y.: The Syntax of Chinese. Cambridge University Press, Cambridge (2009)
19. Jacobson, P.: Paycheck pronouns, Bach-Peters sentences, and variable-free semantics. Natural Language Semantics 8(2), 77–155 (2000)
20. Johnson, M.: Parsing as deduction: the use of knowledge of language. Journal of Psycholinguistic Research 18(1), 105–128 (1989)
21. Karttunen, L.: Syntax and semantics of questions. Linguistics and Philosophy 1, 3–44 (1977)
22. Larson, R., Segal, G.: Knowledge of meaning. MIT Press, Cambridge (1995)
23. May, R.: The grammar of quantification. Ph.D. thesis, MIT, Cambridge (1977)
24. Steedman, M.: The Syntactic Process. MIT Press, Cambridge (2000)
25. Wadler, P.: Transforming programs to eliminate trees. Theoretical Computer Science 73, 231–248 (1990)

Appendix: Grammar of Two Examples

Grammar of QR example (5)		Grammar of *Wh*-movement example (7)	
S → DP VP	D → every	CP → Spec $\overline{\text{C}}$	Pronoun → ni
DP → D NP	D → some	$\overline{\text{C}}$ → C TP	Pronoun → wo
DP → PN	PN → John	TP → DP $\overline{\text{T}}$	NOMwh → shenme
NP → Adj $\overline{\text{N}}$	N → city	TP → Spec $\overline{\text{T}}$	ADVwh → weishenme
NP → Adj NP	N → body	$\overline{\text{T}}$ → T VP	V → xiangzhidao
$\overline{\text{N}}$ → N PP	Adj → Italian	DP → Pronoun	V → mai
VP → V DP	P → in	DP → NOMwh	
VP → V DP PP	V → met	NP → NOMwh	
PP → P DP PP		AdvP → ADVwh	
		$\overline{\text{V}}$ → V CP	
		$\overline{\text{V}}$ → AdvP $\overline{\text{V}}$	
		$\overline{\text{V}}$ → V DP	
		VP → $\overline{\text{V}}$	