Linear Logic Programming with an Ordered Context [*]

Jeff Polakow
Department of Computer Science
Carnegie Mellon University
jpolakow@cs.cmu.edu

ABSTRACT

We begin with a review of *ordered linear logic* (OLL), a refinement of intuitionistic linear logic with an inherent notion of order. We then develop a logic programming interpretation for OLL in two steps: (1) we give a system of ordered uniform derivations which is sound and complete with respect to OLL, and (2) we present a model of resource consumption which removes non-determinism from ordered resource allocation during search for uniform derivations. We also illustrate the expressive power of the resulting *ordered linear logic programming language* with several example programs.

Keywords

non-commutative logic, linear logic, logic programming, automated deduction

1. INTRODUCTION

Linear logic [6] can be considered a logic of *state*, since we can express many problems involving state in a much more natural and concise manner in linear logic than in traditional logics. It supplements the familiar notion of logical assumption with *linear hypotheses* which must be used exactly once in a derivation. Linear logic has found applications in functional programming, logic programming, logical frameworks, concurrency, and related areas. One of the critical properties of linear logic is that it extends traditional logic *conservatively*, that is, we strictly gain expressive power.

Given the many insights resulting from linear logic, we can ask if other computational phenomena besides state can be captured in a similarly logical manner. The one we consider here is the notion of *order*. There are many situations where computation is naturally subject to ordering constraints. Words in a sentence, for example, are ordered, which has given rise to the Lambek calculus [10], a weak logic with an inherent notion of order for the analysis of natural language.

To fully explore uses of an order-aware logic in computer science, we would like it to conservatively extend both traditional and linear logic. Polakow and Pfenning introduced one such system based on *ordered hypotheses* [14]. Our system of natural deduction provides a foundation for applications in functional programming, which was investigated via the Curry-Howard isomorphism and properties of an ordered λ-calculus. Subsequently, we exhibited a related sequent calculus [15] which can serve as the basis for proof search procedures. We call our calculus *ordered linear logic*[1] or *OLL*.

In this paper we investigate proof search and logic programming with ordered hypotheses. We follow the paradigm that logic programming should be understood via an abstract notion of *uniform derivation* [11] which, in a slight abuse of terminology, we take to encompass *goal-directed search* and *focussed* use of hypotheses [2]. Fortunately, the notion of uniform proof extends smoothly from linear logic to OLL. However, the techniques for reducing non-determinism during uniform proof search are significantly complicated when moving from linear logic to OLL. The principal contributions of this paper are

1. a system of ordered uniform derivations which is sound and complete with respect to OLL;
2. a model of resource consumption which removes non-determinism from resource allocation during the search for ordered uniform derivations while remaining sound and complete;
3. example programs which illustrate the expressive power of the resulting ordered logic programming language.

The soundness and completeness proofs (which are omitted due to space limitations) are not trivial, although we found the design of the resource consumption model itself to be the more difficult task. We have successfully experimented with our design and the example programs through a prototype

[*]This work has been partially supported by the National Science Foundation under grants CCR-9804014 and CCR-9619584.

[1]In previous work we called our system intuitionistic non-commutative linear logic (INCLL). However, in order to avoid confusion with other versions of intuitionistic non-commutative linear logic, we now refer to our system as ordered linear logic.

implementation in the Twelf system [12]. In future work we plan to investigate techniques for lower-level efficient implementation of ordered logic programming. We also have to gain more experience with ordered programming techniques.

The remainder of this extended abstract is organized as follows. We first review the sequent formulation of OLL in Section 2 and then introduce uniform derivations and show their soundness and completeness in Section 3. We eliminate non-determinism from resource allocation in Section 4 and show some example logic programs in Section 5. We end with some conclusions and planned work in Section 6.

All rules, proofs, and other details for the purely implicational system (which smoothly extend to the whole unifom fragment) may be found in [13].[2]

2. SEQUENT CALCULUS

We review the sequent calculus for OLL first introduced in [15]. It is shown there that cut is admissible in this system, and that there is a strong connection between cut-free sequent derivations and normal natural deduction in OLL as presented in [14]. For the sake of brevity, we restrict ourselves to the uniform fragment and omit all formal theorems.

Formulas	A	$::=$	P	atomic propositions
		$\mid$	$A_1 \to A_2$	unrestricted implication
		$\mid$	$A_1 \multimap A_2$	linear implication
		$\mid$	$A_1 \twoheadrightarrow A_2$	ordered right implication
		$\mid$	$A_1 \rightarrowtail A_2$	ordered left implication
		$\mid$	$A_1 \,\&\, A_2$	additive conjunction
		$\mid$	$\top$	additive truth
		$\mid$	$\forall x.\, A$	universal quantification

We note that in the Lambek calculus, $A_1 \rightarrowtail A_2$ is written as $A_1 \backslash A_2$ and $A_1 \twoheadrightarrow A_2$ as A_2 / A_1.

Our sequents have the form $\Gamma; \Delta; \Omega \Longrightarrow A$ where Γ, Δ, and Ω are lists of hypotheses interpreted as follows:

- Γ are *unrestricted hypotheses* (they may be used arbitrarily often in any order),
- Δ are *linear hypotheses* (each must be used exactly once, but in no particular order),
- Ω are *ordered hypotheses* (each must be used exactly once, subject to their order).

We use "$\cdot$" to stand for the empty list and juxtaposition for both list concatenation and adjoining an element to a list. Manipulating linear hypotheses requires a non-deterministic merge operation which may interleave the hypotheses in arbitrary order. We denote this operation by $\bowtie$.

We construct our sequent calculus such that the expected structural rules within each context will be admissible without being explicitly part of the system. So Γ admits exchange, weakening, and contraction, Δ admits exchange, and Ω does not admit any structural rules (except for associativity which is built in the formulation of the context as a list).

[2]Available at
`http://www.cs.cmu.edu/~jpolakow/CMU-CS-98-183.ps.gz`

We start with initial sequents, which encode that all linear and ordered hypotheses must be used, while those in Γ need not be used.

$$\frac{}{\Gamma; \cdot; A \Longrightarrow A}\,\textbf{init}$$

We have two explicit structural rules: **place** which commits a linear hypothesis to a particular place among the ordered hypotheses, and **copy** which duplicates and places an unrestricted hypothesis.

$$\frac{\Gamma_L A \Gamma_R; \Delta; \Omega_L A \Omega_R \Longrightarrow B}{\Gamma_L A \Gamma_R; \Delta; \Omega_L \Omega_R \Longrightarrow B}\,\textbf{copy} \qquad \frac{\Gamma; \Delta_L \Delta_R; \Omega_L A \Omega_R \Longrightarrow B}{\Gamma; \Delta_L A \Delta_R; \Omega_L \Omega_R \Longrightarrow B}\,\textbf{place}$$

The following rules describing unrestricted implications translate the standard sequent rules for intuitionistic linear logic into our setting. Note the restrictions on the linear and ordered contexts in the left rule which is necessary to preserve linearity and order, respectively.

$$\frac{\Gamma A; \Delta; \Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \to B}\to_R$$

$$\frac{\Gamma; \Delta; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \cdot; \cdot \Longrightarrow A}{\Gamma; \Delta; \Omega_L (A \to B) \Omega_R \Longrightarrow C}\to_L$$

The rules for linear implications are treated similarly.

The right rule for ordered right implication, $A \twoheadrightarrow B$, adds A at the right end of the ordered context. For cut-elimination to hold, the left rule must then take hypotheses immediately to the right of the right implication for deriving the antecedent A. The remaining hypotheses are joined with B (in order) to derive C. We must also be careful that each linear hypothesis comes from exactly one premise, although their order does not matter (hence the merge operation $\Delta_B \bowtie \Delta_A$).

$$\frac{\Gamma; \Delta; \Omega A \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \twoheadrightarrow B}\twoheadrightarrow_R$$

$$\frac{\Gamma; \Delta_B; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \Delta_A; \Omega_A \Longrightarrow A}{\Gamma; \Delta_B \bowtie \Delta_A; \Omega_L (A \twoheadrightarrow B) \Omega_A \Omega_R \Longrightarrow C}\twoheadrightarrow_L$$

The rules for left implication are symmetric.

Finally, the rules for &, $\top$, and $\forall$ are as expected. Figure 1 contains the complete uniform fragment.

We conclude this section with some brief remarks on the relation between OLL and some other non-commutative logics. The full system for OLL includes additive disjunction $\oplus$, additive falsehood 0, ordered multiplicative conjunction $\bullet$, multiplicative truth 1, an exponential operator !, and a mobility operator ¡.[3] In fact OLL may be viewed as an intuitionistic version of Yetter's cyclic linear logic [17].

The non-commutative logic (NL) of Abrusci and Ruet [1] is a classical (and therefore cyclic) logic which incorporates both commutative and non-commutative logical connectives. These correspond to two different context constructors in the sequent calculus presentation. The logic is essentially all of linear logic with the addition of a non-commutative tensor and

[3]For a complete description of OLL's proof theory, see [15]

$$\frac{\Gamma A;\Delta;\Omega \implies B}{\Gamma;\Delta;\Omega \implies A \to B}\,{\to_R} \qquad \frac{\Gamma;\Delta A;\Omega \implies B}{\Gamma;\Delta;\Omega \implies A \multimap B}\,{\multimap_R} \qquad \frac{\Gamma;\Delta;\Omega A \implies B}{\Gamma;\Delta;\Omega \implies A \twoheadrightarrow B}\,{\twoheadrightarrow_R} \qquad \frac{\Gamma;\Delta;A\Omega \implies B}{\Gamma;\Delta;\Omega \implies A \rightarrowtail B}\,{\rightarrowtail_R}$$

$$\frac{}{\Gamma;\Delta;\Omega \implies \top}\,\top_R \qquad \frac{\Gamma;\Delta;\Omega \implies A \qquad \Gamma;\Delta;\Omega \implies B}{\Gamma;\Delta;\Omega \implies A \,\&\, B}\,\&_R \qquad \frac{\Gamma;\Delta;\Omega \implies [a/x]A}{\Gamma;\Delta;\Omega \implies \forall x.\, A}\,\forall_R^a \quad (a \text{ not free in conclusion})$$

$$\frac{}{\Gamma;\cdot;A \implies A}\,\textbf{init} \qquad \frac{\Gamma_L A \Gamma_R;\Delta;\Omega_L A \Omega_R \implies B}{\Gamma_L A \Gamma_R;\Delta;\Omega_L \Omega_R \implies B}\,\textbf{copy} \qquad \frac{\Gamma;\Delta_L \Delta_R;\Omega_L A \Omega_R \implies B}{\Gamma;\Delta_L A \Delta_R;\Omega_L \Omega_R \implies B}\,\textbf{place}$$

$$\frac{\Gamma;\Delta;\Omega_L B \Omega_R \implies C \qquad \Gamma;\cdot;\cdot \implies A}{\Gamma;\Delta;\Omega_L(A \to B)\Omega_R \implies C}\,{\to_L} \qquad \frac{\Gamma;\Delta_B;\Omega_L B \Omega_R \implies C \qquad \Gamma;\Delta_A;\cdot \implies A}{\Gamma;\Delta_B \bowtie \Delta_A;\Omega_L(A \multimap B)\Omega_R \implies C}\,{\multimap_L}$$

$$\frac{\Gamma;\Delta_B;\Omega_L B \Omega_R \implies C \qquad \Gamma;\Delta_A;\Omega_A \implies A}{\Gamma;\Delta_B \bowtie \Delta_A;\Omega_L(A \twoheadrightarrow B)\Omega_A \Omega_R \implies C}\,{\twoheadrightarrow_L} \qquad \frac{\Gamma;\Delta_B;\Omega_L B \Omega_R \implies C \qquad \Gamma;\Delta_A;\Omega_A \implies A}{\Gamma;\Delta_B \bowtie \Delta_A;\Omega_L \Omega_A(A \rightarrowtail B)\Omega_R \implies C}\,{\rightarrowtail_L}$$

$$\frac{\Gamma;\Delta;\Omega_L A \Omega_R \implies C}{\Gamma;\Delta;\Omega_L(A \,\&\, B)\Omega_R \implies C}\,\&_{L1} \qquad \frac{\Gamma;\Delta;\Omega_L B \Omega_R \implies C}{\Gamma;\Delta;\Omega_L(A \,\&\, B)\Omega_R \implies C}\,\&_{L2} \qquad \frac{\Gamma;\Delta;\Omega_L([t/x]A)\Omega_R \implies C}{\Gamma;\Delta;\Omega_L(\forall x.\, A)\Omega_R \implies C}\,\forall_L$$

Figure 1: Sequent rules for the uniform fragment of OLL.

and its dual. There is no explicit mobility modality as in our system. The intuitionistic version of non-commutative logic presented in [16, 7] is an acyclic, single-conclusion restriction of NL. Thus our system may be seen as the fragment of intuitionistic NL without the commutative connectives, extended with a mobility modality.

3. UNIFORM DERIVATIONS

Now that we have a suitable sequent system for OLL, we begin analyzing proof structure with an eye towards achieving a logic programming language, where we view computation as the bottom-up construction of a derivation. We refer to the succedent of a given sequent as the *goal*. The difficulty with the sequent system is that in any situation, many left or right rules could be applied, leading to unacceptable non-determinism. To solve this problem, we design an alternative, more restricted system with the following properties (which are enforced *syntactically*):

- Derivations are *goal-directed* in that a sequent with a non-atomic goal always ends in a right rule. This allows us to view logical connectives in goals as search instructions.
- Derivations are *focussed* in that when deriving a sequent with an atomic goal we single out a particular hypothesis and apply a sequence of left rules until it is also atomic and immediately implies the goal. This allows us to view atomic goals as procedure calls.

In a minor departure from [11] we call derivations which are both goal-directed and focussed *uniform*. Our system employs the following two judgements:

$$\Gamma;\Delta;\Omega \longrightarrow A$$

which denotes that goal A is uniformly derivable; and

$$\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P$$

which denotes that hypothesis A immediately entails atomic goal P. In both judgements Γ, Δ and Ω are unrestricted, linear, and ordered hypotheses, respectively. In the latter judgment the ordered hypotheses are syntactically divided into a left part Ω_L and a right part Ω_R. It corresponds to the sequent

$$\Gamma;\Delta;(\Omega_L A \Omega_R) \implies P$$

so that the split in the ordered context tracks the location of the hypothesis we have focused on. This correspondence is stated formally in the soundness and completeness theorems for uniform derivations below.

All of the right rules are exactly the same as in the sequent calculus.

Since no left rules apply when the goal is non-atomic, the derivation is completely determined by the structure of the goal, as desired. When the goal has become atomic, we need to single out a hypothesis and determine if it immediately entails the goal. This is achieved by the three **choice** rules which apply to unrestricted, linear, or ordered hypotheses.

$$\frac{\Gamma_L A \Gamma_R;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma_L A \Gamma_R;\Delta;\Omega_L \Omega_R \longrightarrow P}\,\textbf{choice}_\Gamma$$

$$\frac{\Gamma;\Delta_L \Delta_R;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma;\Delta_L A \Delta_R;\Omega_L \Omega_R \longrightarrow P}\,\textbf{choice}_\Delta$$

$$\frac{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma;\Delta;\Omega_L A \Omega_R \longrightarrow P}\,\textbf{choice}_\Omega$$

choice$_\Gamma$ is justified by **copy** in the sequent calculus, and **choice**$_\Delta$ by **place**. The premise and conclusion of **choice**$_\Omega$ correspond to identical sequents. An initial sequent corresponds to an immediate entailment between identical atomic formulas.

$$\frac{}{\Gamma; \cdot; (\cdot;\cdot) \longrightarrow P \gg P}\,\textbf{init}$$

The remaining left rules for immediate entailment directly correspond to the left sequent rules, keeping in mind that we have to consider the focussing formula as being between the left and right parts of the ordered context. Figure 2 contains the complete uniform derivation system.

Note that in $\twoheadrightarrow_L$ and $\rightarrowtail_L$, Ω_A is some initial or final segment of the right or left part of the ordered context, respectively.

Uniform derivations are sound and complete with respect to the sequent calculus. The soundness result is easy to show following the intuition given above.

Theorem 1. *Soundness of Uniform Derivations*

1. *If* $\Gamma; \Delta; \Omega \longrightarrow A$ *then* $\Gamma; \Delta; \Omega \Longrightarrow A$.
2. *If* $\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow A \gg P$ *then* $\Gamma; \Delta; \Omega_L\, A\, \Omega_R \Longrightarrow P$.

Proof. By mutual structural induction on the sequent derivations of the given judgements. □

The completeness result is harder, but largely follows techniques of [2] and [11], adapted to the ordered case.

Theorem 2. *Completeness of Uniform Derivations* *If* $\Gamma; \Delta; \Omega \Longrightarrow A$ *then* $\Gamma; \Delta; \Omega \longrightarrow A$.

Proof. By induction on the structure of the given judgement appealing to the inversion and permutation properties of the sequent calculus (whose formulation and proof are omitted from this paper). □

We have now shown that OLL qualifies as an abstract logic programming language in the sense of [11]. However, uniform derivations as given above are not yet suitable for a logic programming interpreter, since there is an enormous amount of non-determinism in the system, much of which arises from the need to split the linear and ordered contexts.

4. ORDERED RESOURCE MANAGEMENT

There are several sources of non-determinism in uniform derivations which must be resolved in order to obtain a predictable operational behavior. Fortunately, standard solutions suffice for most of them. The selection of hypothesis implicit in the **choice** rules is resolved by scanning the context in a fixed order and backtracking. The selection of subgoals in the rules with two premises proceeds from left to right. Instantiation of universal quantifiers is postponed and the **init** rule performs unification.

What remains are issues of resource management. Unrestricted hypotheses are propagated to all subgoals without difficulty. Linear hypotheses can be treated as in the so-called IO system of Hodas and Miller [9]. The rules $\multimap_L$, $\twoheadrightarrow_L$, and $\rightarrowtail_L$ propagate all linear hypotheses to the first premise which returns the list of unused hypotheses when it has been solved successfully. These are then passed on to the second premise. The hypotheses used in neither premise are then returned as unused in the conclusion.

This model of deterministic *resource consumption* is intuitively attractive and easy to reason about for the programmer, but its extension to the ordered context is complicated by the need to preserve the order of the hypotheses.[4] For the main judgment of uniform derivability, adding input and output contexts is straightforward.

$$\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \longrightarrow A$$

During the search, the *input contexts* Γ, Δ_I, and Ω_I and the goal A are given, while the *output contexts* Δ_O and Ω_O are returned. In the interest of economy (both for the presentation of the rules and the implementation) we do not actually delete formulas from Δ_I and Ω_I but replace them with a placeholder $\Box$. For the remainder of this paper, contexts may contain formulas and placeholders. In order to state the invariants relating input and output contexts we define the context difference $\Omega_I - \Omega_O$:

$$\begin{array}{rcl} \cdot - \cdot & = & \cdot \\ \Omega_I A - \Omega_O A & = & \Omega_I - \Omega_O \\ \Omega_I \Box - \Omega_O \Box & = & \Omega_I - \Omega_O \\ \Omega_I A - \Omega_O \Box & = & (\Omega_I - \Omega_O) A \\ \Omega_I \Box - \Omega_O A & = & \text{undefined} \end{array}$$

$\Omega_I - \Omega_O$ is also undefined when $\|\Omega_I\| \neq \|\Omega_O\|$ where $\|\Omega\|$ denotes the length of list Ω. In valid derivations the context difference $\Omega_I - \Omega_O$ and $\Delta_I - \Delta_O$ is always defined—we elide the corresponding lemma in this extended abstract.

The right rules for the ordered resource management judgment are constructed by expanding the contexts from the uniform derivation system into pairs of input/output contexts. Thus the $\twoheadrightarrow_R$ rule is just:

$$\frac{\Gamma\,;\ \Delta_I \backslash \Delta_O\ ;\ \Omega_I A \backslash \Omega_O \Box \longrightarrow B}{\Gamma\,;\ \Delta_I \backslash \Delta_O\ ;\ \Omega_I \backslash \Omega_O \longrightarrow A \twoheadrightarrow B}\,\twoheadrightarrow_R$$

We require the $\Box$ in the output context to ensure the hypothesis has actually been used. The other right rules are constructed similarly (see figure 3). Note that this lazy splitting of resources actually introduces non-determinism into the $\top_R$ rule.

Next we come to the **choice**$_\Omega$ rule, that is, we chose to focus on an ordered assumption. This determines the division of the remaining ordered hypotheses unambiguously. We therefore divide the input contexts and join the output contexts at the chosen assumption. The new judgment reads

$$\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \longrightarrow A \gg P$$

where Ω_{LI} and Ω_{RI} are the parts to the left and right of the focussed formula A, and Ω_{LO} and Ω_{RO} are the correspond-

[4]$\cdot; \cdot; AB(B \rightarrowtail A \rightarrowtail C) \longrightarrow C$ is a derivable sequent while $\cdot; \cdot; BA(B \rightarrowtail A \rightarrowtail C) \longrightarrow C$ is not.

$$\frac{\Gamma A;\Delta;\Omega \longrightarrow B}{\Gamma;\Delta;\Omega \longrightarrow A \rightarrow B}\rightarrow_R \qquad \frac{\Gamma;\Delta A;\Omega \longrightarrow B}{\Gamma;\Delta;\Omega \longrightarrow A \multimap B}\multimap_R \qquad \frac{\Gamma;\Delta;A\Omega \longrightarrow B}{\Gamma;\Delta;\Omega \longrightarrow A \rightarrowtail B}\rightarrowtail_R \qquad \frac{\Gamma;\Delta;\Omega A \longrightarrow B}{\Gamma;\Delta;\Omega \longrightarrow A \twoheadrightarrow B}\twoheadrightarrow_R$$

$$\frac{}{\Gamma;\Delta;\Omega \Longrightarrow \top}\top_R \qquad \frac{\Gamma;\Delta;\Omega \Longrightarrow A \qquad \Gamma;\Delta;\Omega \Longrightarrow B}{\Gamma;\Delta;\Omega \Longrightarrow A \,\&\, B}\&_R \qquad \frac{\Gamma;\Delta;\Omega \Longrightarrow [a/x]A}{\Gamma;\Delta;\Omega \Longrightarrow \forall x.\, A}\forall_R^a \quad (a \text{ not free in conclusion})$$

$$\frac{}{\Gamma;\cdot;(\cdot;\cdot) \longrightarrow P \gg P}\textbf{init} \qquad \frac{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma;\Delta;\Omega_L A \Omega_R \longrightarrow P}\textbf{choice}_\Omega$$

$$\frac{\Gamma_L A \Gamma_R;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma_L A \Gamma_R;\Delta;\Omega_L\Omega_R \longrightarrow P}\textbf{choice}_\Gamma \qquad \frac{\Gamma;\Delta_L\Delta_R;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma;\Delta_L A \Delta_R;\Omega_L\Omega_R \longrightarrow P}\textbf{choice}_\Delta$$

$$\frac{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow B \gg P \qquad \Gamma;\cdot;\cdot \longrightarrow A}{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \rightarrow B \gg P}\rightarrow_L \qquad \frac{\Gamma;\Delta_B;(\Omega_L;\Omega_R) \longrightarrow B \gg P \qquad \Gamma;\Delta_A;\cdot \longrightarrow A}{\Gamma;\Delta_A \bowtie \Delta_B;(\Omega_L;\Omega_R) \longrightarrow A \multimap B \gg P}\multimap_L$$

$$\frac{\Gamma;\Delta_B;(\Omega_L;\Omega_R) \longrightarrow B \gg P \qquad \Gamma;\Delta_A;\Omega_A \longrightarrow A}{\Gamma;\Delta_A \bowtie \Delta_B;(\Omega_L\Omega_A;\Omega_R) \longrightarrow A \rightarrowtail B \gg P}\rightarrowtail_L \qquad \frac{\Gamma;\Delta_B;(\Omega_L;\Omega_R) \longrightarrow B \gg P \qquad \Gamma;\Delta_A;\Omega_A \longrightarrow A}{\Gamma;\Delta_A \bowtie \Delta_B;(\Omega_L;\Omega_A\Omega_R) \longrightarrow A \twoheadrightarrow B \gg P}\twoheadrightarrow_L$$

$$\frac{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P}{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \,\&\, B \gg P}\&_{L1} \qquad \frac{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow B \gg P}{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \,\&\, B \gg P}\&_{L2} \qquad \frac{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow [t/x]A \gg P}{\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow \forall x.\, A \gg P}\forall_L$$

Figure 2: Uniform derivation system for OLL.

ing output contexts. The **choice**$_\Omega$ rule for this system then looks as follows:

$$\frac{\Gamma\ ;\ \Delta_I\backslash\Delta_O\ ;\ (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \longrightarrow A \gg P}{\Gamma\ ;\ \Delta_I\backslash\Delta_O\ ;\ \Omega_{LI}A\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO} \longrightarrow P}\textbf{choice}_\Omega$$

Replacing A from the input context with $\Box$ in the output context indicates that A was consumed. We postpone dealing with the other choice rules.

The **init** rule does not consume any resources except for the focus formula. Therefore all input resources are passed on.

$$\frac{}{\Gamma;\Delta\backslash\Delta;(\Omega_L\backslash\Omega_L;\Omega_R\backslash\Omega_R) \longrightarrow P \gg P}\textbf{init}$$

This effectively states that the linear and ordered contexts of the initial sequent should be empty.

The $\rightarrow_L$, $\multimap_L$, $\&_L$, and $\forall_L$ rules for this judgment introduce no new ideas.

We now consider the left rule for right implication. We are trying to derive a judgment of the form

$$\Gamma\ ;\ \Delta_I\backslash\Delta_O\ ;\ (\Omega_{LI}\backslash ?\ ;\ \Omega_{RI}\backslash ?) \longrightarrow A \twoheadrightarrow B \gg P$$

where the ordered output contexts denoted by **?** have yet to be computed. Because $A \twoheadrightarrow B$ is a right implication situated between Ω_{LI} and Ω_{RI}, the derivation of A must consume some initial segment of Ω_{RI}. Before that, we need to see if B immediately entails P (the left premise of the $\twoheadrightarrow_L$ rule)[5] which we obtain from

$$\Gamma\ ;\ \Delta_I\backslash\Delta_M\ ;\ (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega) \longrightarrow B \gg P$$

Then we need to take the unconsumed parts at the left end of Ω, denoted by Ω_{AI}, and allow them as the ordered input context for the solution to A.

$$\Gamma\ ;\ \Delta_M\backslash\Delta_O\ ;\ \Omega_{AI}\backslash\Omega_{AO} \longrightarrow A$$

Now we can fill the holes in the conclusion with Ω_{LO} and $\Omega_{AO}\Omega_{RO}$, respectively. In summary, the rule reads

$$\frac{\begin{array}{c}\Gamma\ ;\ \Delta_I\backslash\Delta_M\ ;\ (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega) \longrightarrow B \gg P\\ \Gamma\ ;\ \Delta_M\backslash\Delta_O\ ;\ \Omega_{AI}\backslash\Omega_{AO} \longrightarrow A\end{array}}{\Gamma\ ;\ \Delta_I\backslash\Delta_O\ ;\ (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{AO}\Omega_{RO}) \longrightarrow A \twoheadrightarrow B \gg P}\twoheadrightarrow_{L1}$$

where Ω_{AI} is the longest leftmost segment of Ω not containing $\Box$, and Ω_{RO} the remainder (so $\Omega = \Omega_{AI}\Omega_{RO}$). The left rule for right ordered implication is symmetric. Figure 4 summarizes the left rules where the contrext split is known.

A difficulty remains, however, in that when a formula is directly chosen from the unrestricted or linear context, its exact position in the ordered context is undetermined. As before, we would like the rule applications in the derivation of the premise to implicitly determine where the formula might have been placed. This is captured in the judgment

$$\Gamma\ ;\ \Delta_I\backslash\Delta_O\ ;\ \Omega_I\backslash(\Omega_L|\Omega_M|\Omega_R) \longrightarrow A \gg P$$

where $\Omega_I - \Omega_L\Omega_M\Omega_R$ is defined and Ω_M does not contain any $\Box$. Since no formula in Ω_M is actually consumed in

[5]In Prolog terminology: we need to unify the clause head with P before solving any subgoals.

the derivation of $A \gg P$, the whole subcontext marks the place where A occurs in the ordered context in the sequent calculus.

The unrestricted choice rule

$$\frac{\Gamma_L A \Gamma_R \ ; \ \Delta_I \backslash \Delta_O \ ; \ \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \ \longrightarrow \ A \gg P}{\Gamma_L A \Gamma_R \ ; \ \Delta_I \backslash \Delta_O \ ; \ \Omega_I \backslash \Omega_L \Omega_M \Omega_R \ \longrightarrow \ P} \textbf{choice}_\Gamma$$

then just passes the whole input context Ω_I. The $\textbf{choice}_\Delta$ rule is similar.

The rest of the left rules are constructed using similar reasoning (see figure 5).

To summarize, our context management system is based on three judgments.

$$\begin{array}{l}\Gamma \ ; \ \Delta_I \backslash \Delta_O \ ; \ \Omega_I \backslash \Omega_O \ \longrightarrow \ A \\ \Gamma \ ; \ \Delta_I \backslash \Delta_O \ ; \ (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{RO}) \ \longrightarrow \ A \gg P \\ \Gamma \ ; \ \Delta_I \backslash \Delta_O \ ; \ \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \ \longrightarrow \ A \gg P\end{array}$$

where contexts Δ and Ω may contain $\Box$ as placeholder for a consumed formula. The first two judgments mirror the behavior of the uniform sequents where the ordered context split is always known. The third sequent is used when a focus formula is chosen from the unrestricted or linear contexts and the splitting of the ordered context is to be determined lazily.

We prove the correctness of the resource management system with respect to the uniform system by separately proving a completeness and a soundness result. Complete proofs for the implicational fragment may be found in [13].

THEOREM 3. *Soundness of Ordered Resource Management*

1. *If* $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \longrightarrow \ A$ *then* $\Gamma; \Delta_I - \Delta_O; \Omega_I - \Omega_O \ \longrightarrow \ A$,
2. *if* $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}; \ \Omega_{RI} \backslash \Omega_{RO}) \ \longrightarrow \ B \gg P$ *then* $\Gamma; \Delta_I - \Delta_O; (\Omega_{LI} - \Omega_{LO}; \Omega_{RI} - \Omega_{RO}) \longrightarrow B \gg P$, *and*
3. *if* $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \ \longrightarrow \ A \gg P$ *then* $\Gamma; \Delta_I - \Delta_O; (\Omega_{IL} - \Omega_L; \Omega_{IR} - \Omega_R) \ \longrightarrow \ A \gg P$,

where $\Omega_I, \Omega_{LI}, \Omega_{RI}$ *do not contain* $\Box$ *and* $\Omega_I = \Omega_{IL}\Omega_M\Omega_{IR}$ *and* $\|\Omega_{IL}\| = \|\Omega_L\|$ *and* $\|\Omega_{IR}\| = \|\Omega_R\|$.

PROOF. By mutual induction on the structure of the given derivations. $\Box$

THEOREM 4. *Completeness of Ordered Resource Management*

1. *For all* $\Delta_I, \Delta_O, \Omega_I, \Omega_O$ *such that* $\Delta_I - \Delta_O = \Delta$ *and* $\Omega_I - \Omega_O = \Omega$ *we have that* $\Gamma; \Delta; \Omega \ \longrightarrow \ A$ *implies* $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \longrightarrow \ A$
2. *For all* $\Delta_I, \Delta_O, \Omega_{LI}, \Omega_{LO}, \Omega_{RI}, \Omega_{RO}$ *such that* $\Delta_I - \Delta_O = \Delta$, $\Omega_{LI} - \Omega_{LO} = \Omega_L$ *and* $\Omega_{RI} - \Omega_{RO} = \Omega_R$ *we have that* $\Gamma; \Delta; (\Omega_L; \Omega_R) \ \longrightarrow \ A \gg P$ *implies*
   (a) $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}; \Omega_{RI} \backslash \Omega_{RO}) \ \longrightarrow \ A \gg P$ *and*
   (b) $\Gamma; \Delta_I \backslash \Delta_O; \Omega_{LI}\Omega_M\Omega_{RI} \backslash (\Omega_{LO} | \Omega_M | \Omega_{RO}) \longrightarrow A \gg P$,

where $\Omega_I, \Omega_M, \Omega_{LI}, \Omega_{RI}$ *do not contain* $\Box$.

PROOF. By mutual induction on the structure of the given derivations. $\Box$

Using the strategy for choosing formulas described at the beginning of the preceding section, together with the system of ordered resource management described above, we come very close to a usable logic programming language. The usability of the preceding system is mainly hampered by the non-determinism introduced in the $\top$ rule. Removing this non-determinism is the subject of current research. Although we believe it is possible, the task has been surprisingly difficult– once again the techniques used for linear logic [4, 8] can not be directly applied.

5. LOGIC PROGRAMMING

In this section we show some example logic programs which may be written in uniform OLL. We note that all three of our system's contexts are important for logic programming. The program clauses will typically reside in the unrestricted context since they should be arbitrarily usable. The linear context is useful to hold information on which no access restrictions are placed—the final example in this section shows how the linear context can be put to use. The ordered context can act as a logical data structure and hold information which must be accessed in a constrained fashion.

It remains to clarify the order in which the clauses in a context are considered when the goal has become atomic. We employ here the same strategy as the Lolli language [9] and linear LF [5] by dividing the hypotheses into a static program and dynamic assumptions made during search. We first scan all dynamic assumptions from right to left, where $\rightarrow_R$, $\multimap_R$, $\twoheadrightarrow_R$ add assumptions on the right, and $\rightarrowtail_R$ adds assumptions on the left. Then we attempt each formula in the static program from first to last. Note that we have to be aware of the status of each hypothesis (unrestricted, linear, or ordered) as we consider it so we can apply the correct **choice** rule.

In the examples below, we assume a typed, higher-order term language and implicit universal quantification over free variables (tokens beginning with an uppercase letter). We also write $B \twoheadleftarrow A$ for $A \twoheadrightarrow B$ and $B \leftarrowtail A$ for $A \rightarrowtail B$ in the manner of Prolog where backwards arrows bind looser than forwards arrows and &. We use *italics* for meta-variables which stand for ground terms and `typewriter` for program code, including logic variables.

$$\frac{\Gamma A\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; B}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; A \to B}\to_R \qquad \frac{\Gamma\,;\,\Delta_I A\backslash\Delta_O\Box\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; B}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; A \multimap B}\multimap_R$$

$$\frac{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,A\Omega_I\backslash\Box\Omega_O \;\longrightarrow\; B}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; A \rightarrowtail B}\rightarrowtail_R \qquad \frac{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I A\backslash\Omega_O\Box \;\longrightarrow\; B}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; A \twoheadrightarrow B}\twoheadrightarrow_R$$

$$\frac{\Delta_I - \Delta_O \text{ and } \Omega_I - \Omega_O \text{ both defined}}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; \top}\top_R \qquad \frac{\begin{array}{c}\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; A\\ \Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; B\end{array}}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; A \,\&\, B}\&_R$$

$$\frac{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; [a/x]A}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_I\backslash\Omega_O \;\longrightarrow\; \forall x.\, A}\forall^a_R \quad (a \text{ not free in conclusion})$$

Figure 3: Resource management system for OLL– Right rules.

$$\frac{}{\Gamma\,;\,\Delta\backslash\Delta\,;\,(\Omega_L\backslash\Omega_L;\Omega_R\backslash\Omega_R) \;\longrightarrow\; P \gg P}\,\textbf{init}$$

$$\frac{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; A \gg P}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,\Omega_{LI}A\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO} \;\longrightarrow\; P}\,\textbf{choice}_\Omega$$

$$\frac{\begin{array}{l}\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; B \gg P\\ \Gamma\,;\,\cdot\backslash\cdot\,;\,\cdot\backslash\cdot \;\longrightarrow\; A\end{array}}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; A \to B \gg P}\to_L \qquad \frac{\begin{array}{l}\Gamma\,;\,\Delta_I\backslash\Delta_M\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; B \gg P\\ \Gamma\,;\,\Delta_M\backslash\Delta_O\,;\,\cdot\backslash\cdot \;\longrightarrow\; A\end{array}}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; A \multimap B \gg P}\multimap_L$$

$$\frac{\begin{array}{l}\Gamma\,;\,\Delta_I\backslash\Delta_M\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{AI}\Omega_{RO}) \;\longrightarrow\; B \gg P\\ \Gamma\,;\,\Delta_M\backslash\Delta_O\,;\,\Omega_{AI}\backslash\Omega_{AO} \;\longrightarrow\; A\end{array}}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,(\Omega_{LI}\backslash\Omega_{LO}\,;\,\Omega_{RI}\backslash\Omega_{AO}\Omega_{RO}) \;\longrightarrow\; A \twoheadrightarrow B \gg P}\twoheadrightarrow_L \quad (\Box \notin \Omega_{AI} \text{ and } \Omega_{RO} = \Box\Omega' \text{ or } \cdot)$$

$$\frac{\begin{array}{l}\Gamma\,;\,\Delta_I\backslash\Delta_M\,;\,(\Omega_{LI}\backslash\Omega_{LO}\Omega_{AI}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; B \gg P\\ \Gamma\,;\,\Delta_M\backslash\Delta_O\,;\,\Omega_{AI}\backslash\Omega_{AO} \;\longrightarrow\; A\end{array}}{\Gamma\,;\,\Delta_I\backslash\Delta_O\,;\,(\Omega_{LI}\backslash\Omega_{LO}\Omega_{AO}\,;\,\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; A \rightarrowtail B \gg P}\rightarrowtail_L \quad (\Box \notin \Omega_{AI} \text{ and } \Omega_{LO} = \Omega'\Box \text{ or } \cdot)$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{RO}) \longrightarrow A \gg P}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{RO}) \longrightarrow A \,\&\, B \gg P}\&_{L1} \qquad \frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{RO}) \longrightarrow A \gg P}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{RO}) \longrightarrow A \,\&\, B \gg P}\&_{L2}$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{RO}) \longrightarrow [t/x]A \gg P}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{RO}) \longrightarrow \forall x.\, A \gg P}\forall_L$$

Figure 4: Resource management system for OLL– Left rules w/ known split.

$$\frac{}{\Gamma \,;\, \Delta \backslash \Delta \,;\, \Omega \backslash (\cdot | \Omega | \cdot) \;\longrightarrow\; P \gg P}\,\mathbf{init}'$$

$$\frac{\Gamma_L A \Gamma_R \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; A \gg P}{\Gamma_L A \Gamma_R \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash \Omega_L \Omega_M \Omega_R \;\longrightarrow\; P}\,\mathbf{choice}_\Gamma$$

$$\frac{\Gamma \,;\, \Delta_{LI} \Delta_{RI} \backslash \Delta_{LO} \Delta_{RO} \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; A \gg P}{\Gamma \,;\, \Delta_{LI} A \Delta_{RI} \backslash \Delta_{LO} \Box \Delta_{RO} \,;\, \Omega_I \backslash \Omega_L \Omega_M \Omega_R \;\longrightarrow\; P}\,\mathbf{choice}_\Delta \;\; (\|\Delta_{LI}\| = \|\Delta_{LO}\|)$$

$$\frac{\begin{array}{l}\Gamma \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; B \gg P \\ \Gamma \,;\, \cdot \backslash \cdot \,;\, \cdot \backslash \cdot \;\longrightarrow\; A\end{array}}{\Gamma \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; A \rightarrow B \gg P}\,{\rightarrow}'_L \qquad \frac{\begin{array}{l}\Gamma \,;\, \Delta_I \backslash \Delta_M \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; B \gg P \\ \Gamma \,;\, \Delta_M \backslash \Delta_O \,;\, \cdot \backslash \cdot \;\longrightarrow\; A\end{array}}{\Gamma \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; A \multimap B \gg P}\,{\multimap}'_L$$

$$\frac{\begin{array}{l}\Gamma \,;\, \Delta_I \backslash \Delta_M \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; B \gg P \\ \Gamma \,;\, \Delta_M \backslash \Delta_O \,;\, \Omega_M \backslash \Omega_{ML} \Omega_{MR} \;\longrightarrow\; A\end{array}}{\Gamma \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash (\Omega_L | \Omega_{ML} | \Omega_{MR} \Omega_R) \;\longrightarrow\; A \twoheadrightarrow B \gg P}\,{\twoheadrightarrow}'_L \qquad (\Box \notin \Omega_{ML} \text{ and } \Omega_{MR} = \Box\Omega' \text{ or } \cdot)$$

$$\frac{\begin{array}{l}\Gamma \,;\, \Delta_I \backslash \Delta_M \,;\, \Omega_I \backslash (\Omega_L | \Omega_M | \Omega_R) \;\longrightarrow\; B \gg P \\ \Gamma \,;\, \Delta_M \backslash \Delta_O \,;\, \Omega_M \backslash \Omega_{ML} \Omega_{MR} \;\longrightarrow\; A\end{array}}{\Gamma \,;\, \Delta_I \backslash \Delta_O \,;\, \Omega_I \backslash (\Omega_L \Omega_{ML} | \Omega_{MR} | \Omega_R) \;\longrightarrow\; A \rightarrowtail B \gg P}\,{\rightarrowtail}'_L \qquad (\Box \notin \Omega_{MR} \text{ and } \Omega_{ML} = \Omega'\Box \text{ or } \cdot)$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash (\Omega_L | \Omega_M | \Omega_R) \longrightarrow A \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash (\Omega_L | \Omega_M | \Omega_R) \longrightarrow A \,\&\, B \gg P}\,\&'_{L1} \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash (\Omega_L | \Omega_M | \Omega_R) \longrightarrow B \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash (\Omega_L | \Omega_M | \Omega_R) \longrightarrow A \,\&\, B \gg P}\,\&'_{L2}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash (\Omega_L | \Omega_M | \Omega_R) \longrightarrow [t/x]A \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash (\Omega_L | \Omega_M | \Omega_R) \longrightarrow \forall x.\, A \gg P}\,\forall'_L$$

Figure 5: Resource management system for OLL– Left rules w/ unknown split.

5.1 Translating to deBruijn Notation

Our first example is a translation between regular lambda terms and deBruijn style terms, where `tr` e `E'` expects a lambda term, e, and computes an equivalent deBruijn style term, `E'`. This example shows how the ordered context can be used as a stack. We assume the following term constants (where `i` is an arbitrary base type for terms):

```
lam :  (i -> i) -> i.      lam' :  i -> i.
app :  i -> i -> i.        app' :  i -> i -> i.
                           shift :  i -> i
                           one :  i.
```

where `lam`, and `app` represent regular lambda terms (variables will be implicitly represented by meta-variables); while `lam'`, `app'`, `shift`, and `one` represent deBruijn terms.

Given a lambda term, we can construct a deBruijn term by recursively descending through the given term's structure and maintaining a stack of lambda-bound variables. Upon reaching a variable, its depth in the stack corresponds to the index in a deBruijn term.

To translate `app E1 E2`, we simply translate both subterms using the current stack. Thus we need to reduce proving

$$\texttt{var } x_1, \ldots, \texttt{var } x_n \longrightarrow \texttt{tr (app E1 E2) (app' E1' E2')}$$

to proving

$$\texttt{var } x_1, \ldots, \texttt{var } x_n \longrightarrow \texttt{tr E1 E1'}$$

and

$$\texttt{var } x_1, \ldots, \texttt{var } x_n \longrightarrow \texttt{tr E2 E2'}$$

where we have omitted the unrestricted context (which contains only the static program and does not change) and the linear context (which is always empty). The x_i are term-level variables already encountered during the translation. This is achieved with the following clause:

$$\begin{array}{l}\texttt{tr (app E1 E2) (app' E1' E2')} \\ \quad \twoheadleftarrow \texttt{tr E1 E1'} \ \& \ \texttt{tr E2 E2'}.\end{array}$$

which may be read `app E1 E2` translates to `app' E1' E2'` if `E1` translates to `E1'` with the current stack, and `E2` translates to `E2'` also using the current stack. Notice the & gives a copy of the stack (the ordered context) to each subterm translation. Also note that the $\twoheadleftarrow$ forces the subgoals to be solved with resources to their right; thus the placement of this clause must be to the left of the ordered data.[6]

To translate a `lam E`, we add a variable to the stack and translate the body of the lambda. Thus we must reduce solving

$$\texttt{var } x_1, \ldots, \texttt{var } x_n \longrightarrow \texttt{tr (lam E) (lam' E')}$$

to

$$\texttt{var } x_1, \ldots, \texttt{var } x_n, \texttt{ var } x \longrightarrow \texttt{tr (E } x\texttt{) E'}$$

[6]This could equivalently be written with a $\leftarrowtail$ forcing placement of the clause to the right. In general the outermost ordered arrow of an unrestricted (or linear) program clause may be either $\twoheadrightarrow$ or $\rightarrowtail$ without affecting the outcome of a program.

This is accomplished by the following clause:

$$\begin{array}{l}\texttt{tr (lam E) (lam' E')} \\ \quad \twoheadleftarrow \forall x. (\texttt{var } x \twoheadrightarrow \texttt{tr (E } x\texttt{) E'}).\end{array}$$

which may be read `lam E` translates to `lam' E'` if for any term x, `E` x translates to `E'` with x added to the current stack. The inner $\twoheadrightarrow$ forces the variable to be added to the top of the stack (the right side of the ordered context[7]).

To translate a variable, which `E` must be if it is not an application or lambda, we search through the stack, keeping track of how far we've gone, until we find the variable. If the target variable is at the top of the stack (the rightmost `var` in the context) we have the following situation

$$\texttt{var } x_1, \ldots, \texttt{var } x_n, \texttt{ var } x \longrightarrow \texttt{tr } x \texttt{ one}$$

where we are done and can throw away the rest of the stack. This is accomplished by

$$\begin{array}{l}\texttt{tr E one} \\ \quad \twoheadleftarrow \texttt{var E} \\ \quad \twoheadleftarrow \top.\end{array}$$

which may be read `E` translates to `one` if `E` is at the top of the stack; the final $\top$ simply erases the rest of the stack. For this clause to be successfully used, the `var E` must come from the top of the stack (the right of the ordered context) since the $\top$ can only consume data to the left of `var E`.

If the target variable is not at the top of the stack, we must reduce

$$\texttt{var } x_1, \ldots, \texttt{var } x_n, \texttt{ var } x \longrightarrow \texttt{tr } y \texttt{ (shift E')}$$

to

$$\texttt{var } x_1, \ldots, \texttt{var } x_n \longrightarrow \texttt{tr } y \texttt{ E'}$$

which is accomplished by the following.

$$\begin{array}{l}\texttt{tr E (shift E')} \\ \quad \twoheadleftarrow \texttt{var F} \\ \quad \twoheadleftarrow \texttt{tr E E'}.\end{array}$$

which may be read `E` translates to `shift E'` if `E` translates to `E'` with the bottom of the stack.

5.2 Mergesort

Our second example, a merge sort, shows how the ordered context can be used as a queue. This example is rather procedural and the program clauses do not have a very clear declarative reading. However, the program shows how a standard term-level representation of a queue using lists (which would be necessary for a Prolog merge sort), may be shifted up to the logical level where the ordered context can be directly used as a queue.

We elide the constant types for this example and merely note that in the proposition: `mergeSort` k `L`, k is an input list and `L` is an output list computed by mergesort. The

[7]Again this is an arbitrary choice. We could rewrite the whole program to use the left side of the context as the top. However this would require changing the program clauses that read variables off the stack.

computation proceeds in two phases. Assuming an input list $x_1 :: \ldots :: x_n :: \texttt{nil}$ we want to reduce solving

$$\cdot \longrightarrow \texttt{mergeSort}\ (x_1 :: \ldots :: x_n :: \texttt{nil})\ \texttt{L}$$

to solving

$$\texttt{srt}(x_1 :: \texttt{nil}), \ldots, \texttt{srt}(x_n :: \texttt{nil}) \longrightarrow \texttt{msort L}$$

We achieve this with the two clauses

$$\begin{array}{l} \texttt{mergeSort (H::T) L} \\ \quad \twoheadleftarrow (\texttt{srt (H::nil)} \twoheadrightarrow \texttt{mergeSort T L}). \\ \\ \texttt{mergeSort nil L} \twoheadleftarrow \texttt{msort L}. \end{array}$$

In the first clause, the embedded $\twoheadrightarrow$ causes the new `srt` hypothesis to be added to the right of all the other ordered hypotheses (the top of our queue).

In the second phase we assume a general situation of the form

$$\texttt{srt}\ l_n, \ldots, \texttt{srt}\ l_2, \texttt{srt}\ l_1 \longrightarrow \texttt{msort L}$$

where the l_i are already sorted and `L` is still to be computed. Starting from the right, we merge l_1 and l_2 and add the result to the *left* end of the ordered context, in effect using it as a work queue. The resulting situation will be

$$\texttt{srt}\ l_{12}, \texttt{srt}\ l_n, \ldots, \texttt{srt}\ l_4, \texttt{srt}\ l_3 \longrightarrow \texttt{msort L}$$

which is then treated the same way, merging l_3 and l_4. We finish when there is only one element $\texttt{srt}\ k$ in the ordered context and unify `L` with k. This is expressed by the following two clauses.

$$\begin{array}{l} \texttt{msort L} \\ \quad \twoheadleftarrow \texttt{srt L1} \\ \quad \twoheadleftarrow \texttt{srt L2} \\ \quad \leftarrow \texttt{merge L1 L2 L12} \\ \quad \twoheadleftarrow (\texttt{srt L12} \rightarrowtail \texttt{msort L}). \\ \\ \texttt{msort L} \twoheadleftarrow \texttt{srt L}. \end{array}$$

We elide the standard Prolog-style `merge` predicate which, given two sorted lists l_1 and l_2 returns a sorted merge l_{12}. In the first clause, similar to the previous example the three $\twoheadleftarrow$ require that the two `srt` hypotheses be taken from the right of the ordered clauses used to solve the rest of the body. The unrestricted implication, $\leftarrow$, is used for the call to `merge` since the merge operation does not make use of the ordered (or linear) context. The embedded $\rightarrowtail$ causes the new `srt` clause to be inserted at the left end of the ordered consequences available to the recursive computation of `msort`. Since all ordered assumptions must be used, the final clause above can succeed only if the complete list has indeed been sorted, that is, there is only one ordered hypothesis $\texttt{srt}\ l$. If we change the first `msort` clause to assume `L12` on the right, by writing ($\texttt{srt L12} \twoheadrightarrow \texttt{msort L}$) instead of ($\texttt{srt L12} \rightarrowtail \texttt{msort L}$) then we obtain an insertion sort because after one step we arrive at

$$\texttt{srt}\ l_n, \ldots, \texttt{srt}\ l_3, \texttt{srt}\ l_{12}$$

which will next merge l_3 into l_{12}, etc.

5.3 Parsing

The following example is a fragment of a parsing program from [13]. This example shows how OLL can be used to directly parse grammatical constructions with unbounded dependencies such as relative clauses.[8]

$$\begin{array}{rl} 1: & \texttt{snt} \twoheadleftarrow \texttt{vp} \twoheadleftarrow \texttt{np}. \\ 2: & \texttt{vp} \twoheadleftarrow \texttt{np} \twoheadleftarrow \texttt{tv}. \\ 3: & \texttt{rel} \leftarrowtail \texttt{whom} \twoheadleftarrow (\texttt{np} \multimap \texttt{snt}). \\ 4: & \texttt{np} \twoheadleftarrow \texttt{jill}. \\ 5: & \texttt{tv} \twoheadleftarrow \texttt{married}. \end{array}$$

We may intuitively read the formulas in the following manner: $\texttt{snt} \twoheadleftarrow \texttt{vp} \twoheadleftarrow \texttt{np}$ states that a sentence is a verb phrase to the right of a noun phrase. We can use these formulas to parse a phrase by putting each word of the sentence into the ordered context and trying to derive the atomic formula corresponding to the phrase type.

We may interpret clause 3 as: a relative clause is `whom` to the left of a sentence missing a noun phrase. As explained in [8] this is a standard interpretation of relative clauses. By putting a `np` into the linear context, the sentence after `whom` will only be successfully parsed if it is indeed missing a noun phrase.

Figure 6 shows a trace of the above formulas parsing a relative clause, showing at each step the resource sequent (including the current goal), the pending goals, and the the rule applied. The unrestricted context containing the above program is left implicit.

6. CONCLUSION

The succession of systems developed in this paper form a promising basis for incorporating order into linear logic programming in a logical and efficient manner. In [15] we give the full complement of connectives for OLL. Similarly to Lolli [8], our resource management system can support restricted occurrences of those connectives not in the uniform fragment. As noted at the end of section 4 we believe that the non-determinism present in the $\top$ rule can be eliminated. This would result in a proof search procedure where all non-determinism is pushed into the choice of formula to focus on. We further believe that the techniques used in [4] for efficient resource management can also be extended to the system for ordered resource management detailed in this paper, leading to a conservative extension of Lolli. That is, every legal Lolli program would remain legal and have the same operational behavior.

There are many examples where order can be exploited, such as algorithms which employ various forms of stacks or queues. Natural language parsing, the original motivation for the Lambek calculus [10] is a further rich source of examples. Using our prototype implementation we have programmed two such parsers, one sketched above and one threaded which improves on the one given in [8]. Another elegant example in [13] is an abstract machine where the stack of continuations is maintained unobtrusively in the ordered context. In many of our examples, the ordered connectives

[8]Relative clauses typically are not truly unbounded dependencies and as such this naive fragment of a parser will accept some ungrammatical sentences. For a more in depth discussion of this point see [13].

Action	Active hypotheses and goal	Goals pending
	$\cdot$; `whom jill married` $\longrightarrow$ `rel`	none
reduce by 3	$\cdot$; `whom jill married` $\longrightarrow$ `whom`	`np` $\multimap$ `snt`
solved, restore pending goal	$\cdot$; `jill married` $\longrightarrow$ `np` $\multimap$ `snt`	none
assume	`np` ; `jill married` $\longrightarrow$ `snt`	none
reduce by 1	`np` ; `jill married` $\longrightarrow$ `vp`	`np`
reduce by 2	`np` ; `jill married` $\longrightarrow$ `np`	`tv` , `np`
solved, restore pending goal	$\cdot$; `jill married` $\longrightarrow$ `tv`	`np`
reduce by 5	$\cdot$; `jill married` $\longrightarrow$ `married`	`np`
solved, restore pending goal	$\cdot$; `jill` $\longrightarrow$ `np`	none
reduce by 4	$\cdot$; `jill` $\longrightarrow$ `jill`	none
solved		

Figure 6: Trace of parsing example.

allow a more concise, logical specification of algorithms than possible in other languages.

However, there are also some difficulties. If an algorithm requires more than one work queue or stack they can interfere, since we have only one ordered context. Another problem is that we sometimes have to write "glue" code which initializes or erases the ordered context prior to a subcomputation. Despite these current limitations, we feel that ordered logic programming is an interesting paradigm which can shed light on the concept of order in computation and warrants further investigation from all angles.

Besides further work on language design, implementation, and programming methodology, we also plan to investigate the extension of our logic with "mixed contexts" (contexts with commutative and non-commutative constructors) as in the non-commutative logic of Abrusci and Ruet [1]. Recent work by Andreoli and Maieli [3] has shown that focusing proofs (and thus uniform proofs) can be extended to this system. However, little or no work has been done on reducing the non-determinism present in splitting mixed contexts. We conjecture that the techniques discussed in this paper can be extended to mixed contexts.

Finally, the existence of canonical forms for the natural deduction system of OLL [14] means that it might be possible to add it to the linear logical framework [5] in a conservative manner. However, we have not yet considered such issues as type reconstruction or unification.

7. ACKNOWLEDGEMENTS

We would like to thank Frank Pfenning for valuable contributions to this paper.

8. REFERENCES

[1] V. M. Abrusci and P. Ruet. Non-commutative logic I: The multiplicative fragment. Submitted, May 1998.

[2] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[3] J.-M. Andreoli and R. Maieli. Focusing and proof-nets in linear and non-commutative logic. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of 6th International Conference on Logic Programming and Automated Reasoning*, pages 320–336, Tbilisi, Republic of Georgia, Sept. 1999. Springer-Verlag LNAI 1705.

[4] I. Cervesato, J. S. Hodas, and F. Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232:133–163, 2000. Revised version of paper in the Proceedings of the 5th International Workshop on Extensions of Logic Programming, Leipzig, Germany, March 1996.

[5] I. Cervesato and F. Pfenning. A linear logical framework. *Information and Computation*, 1999. To appear in the special issue with invited papers from LICS'96, E. Clarke, editor.

[6] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[7] Philippe De Groote. Partially commutative linear logic: Sequent calculus and phase semantics. In V.M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories, Applications of Logic to the Analysis and Implementation of Natural Language*, pages 199–208, Roma, Italy, 1996.

[8] J. S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[9] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.

[10] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:363–386, 1958.

[11] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[12] F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the*

16th International Conference on Automated Deduction (CADE-16), pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[13] J. Polakow and F. Pfenning. Ordered linear logic programming. Technical Report CMU-CS-98-183, Department of Computer Science, Carnegie Mellon University, Dec. 1998.

[14] J. Polakow and F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In J.-Y. Girard, editor, *Proceedings of the Fourth International Conference on Typed Lambda Calculi and Applications (TLCA'99)*, pages 295–309, l'Aquila, Italy, Apr. 1999. Springer-Verlag LNCS 1581.

[15] J. Polakow and F. Pfenning. Relating natural deduction and sequent calculus for intuitionistic non-commutative linear logic. In A. Scedrov and A. Jung, editors, *Proceedings of the 15th Conference on Mathematical Foundations of Programming Semantics*, pages 311–328, New Orleans, Louisiana, Apr. 1999. Electronic Notes in Theoretical Computer Science, Volume 20.

[16] Paul Ruet. *Logique non-commutative et programmation concurrente par contraintes.* PhD thesis, Universite Denis Diderot, Paris 7, 1997.

[17] D. Yetter. Quantales and (non-commutative) linear logic. *Journal of Symbolic Logic*, 55(1), 1990.