For the completness direction we need to generalize the induction hypothesis somewhat differently.

Theorem 5.2 (Completeness of I/O Resource Management)

1. *If* $\Gamma; \Delta \Longrightarrow A \Uparrow$ *then* $\Gamma; \Delta[1], \Delta_O \backslash \Delta[0], \Delta_O \Longrightarrow A \Uparrow$ *for any* Δ_O.

2. *If* $\Gamma; \Delta; A \Downarrow \Longrightarrow P$ *then* $\Gamma; \Delta[1], \Delta_O \backslash \Delta[0], \Delta_O; A \Downarrow \Longrightarrow P$ *for andy* Δ_O.

Proof: By mutual induction on the structure of the given derivation.[5] □

5.4 Some Example Programs

We start with some simple programs. Following the tradition of logic programming, we write implications in the program (Γ) in reverse so that $A \circ\!\!- B$ means $B \multimap A$. Implication in this direction is left associative, and subgoals solved (visually) from left-to-right. So,

$$P \circ\!\!- Q \circ\!\!- R$$

stands for $(P \circ\!\!- Q) \circ\!\!- R$ which is the same as $R \multimap (Q \multimap P)$. If P matches the current atomic goal, then first subgoal to be solved is Q and then R. This is consistent with the informal operational semantics explained above.

The first program is non-terminating for the simple query p.

$u_1 \;:\; p \circ\!\!- p.$
$u_0 \;:\; p.$

Then a goal $\Longrightarrow p$ under this program will diverge, since it will use u_1 first, which produces the identical subgoal of $\Longrightarrow p$. If we reorder the clauses

$u_0 \;:\; p.$
$u_1 \;:\; p \circ\!\!- p.$

the query $\Longrightarrow p$ will produce the immediate proof (u_0) first and, if further answer are requested, succeed arbitrarily often with different proofs. We can slightly complicate this example by adding an argument to p.

$u_0 \;:\; p(0).$
$u_s \;:\; \forall x.\, p(\mathsf{s}(x)) \circ\!\!- p(x).$

In a query we can now leave an existential variable, indicated by an uppercase letter, $\Longrightarrow p(X)$. this query will succeed and print the answer substitution $X = 0$. If further solutions are requested, the program will enumerate $X = \mathsf{s}(0)$, $X = \mathsf{s}(\mathsf{s}(0))$, etc. In general, most logic programming language implementation print only substitutions for existential variables in a query, but not other aspects of the proof it found.

The trivial examples above do not take advantage of the expressive power of linear logic and could equally well be written, for example, in Prolog.

For the next example we introduce lists as terms, using constructors nil and cons. For example, the list $1, 2, 3$ would be written as $\mathsf{cons}(1, \mathsf{cons}(2, \mathsf{cons}(3, \mathsf{nil})))$. A program to enumerate all permutations of a list is the following.

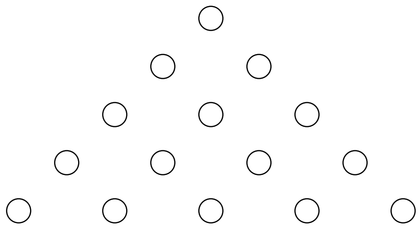
[5] [*check for lemmas and write out some cases*]

$$
\begin{array}{lcl}
p_0 & : & \mathsf{perm}(\mathsf{cons}(X, L), K) \multimapinv (\mathsf{elem}(X) \multimap \mathsf{perm}(L, K)) \\
p_1 & : & \mathsf{perm}(\mathsf{nil}, \mathsf{cons}(X, K)) \multimapinv \mathsf{elem}(X) \multimapinv \mathsf{perm}(\mathsf{nil}, K) \\
p_2 & : & \mathsf{perm}(\mathsf{nil}, \mathsf{nil})
\end{array}
$$

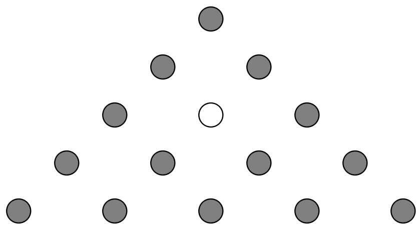
Here we have left universal quantifiers on X, L, and K implicit in each declaration in order to shorten the program. This is also supported by implementations of logic programming languages.

We assume a query of the form $\Longrightarrow \mathsf{perm}(l, K)$ where l is a list and K is a free existential variable. The program iterates over the list l with p_0, creating a linear hypothesis $\mathsf{elem}(t)$ for every element t of the list. Then it repeatedly uses clause p_1 to consume the linear hypothesis in the output list K. When there are no longer any linear hypotheses, the last clause p_2 will succeed and therefore the whole program.

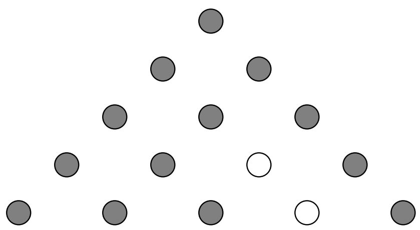
As a second example we consider a simple solitaire game with a board of the form



Each circle represents a hole which is filled if that hole contains a peg. The initial position



has one empty hole, while all other holes are filled by pegs (14 all together). We move by jumping one peg over another if the hole behind it is empty. For example, in the given initial position there are only two legal moves, one of which leads to the following situation:



The goal is to achieve a configuration in which only one peg is left. Alternatively, we can say the goal is to make 13 consecutive moves starting from the original position.

We use two digits to represent the address of each hole on the board in the following manner:

```
                00
              10  11
            20  21  22
          30  31  32  33
        40  41  42  43  44
```

To represent the current situation, we have two predicates

$\mathsf{empty}(x, y)$	Hole xy is empty
$\mathsf{peg}(x, y)$	Hold xy contains a peg

Then the initial situation is represent by the linear hypotheses

$$\begin{array}{l}
\mathsf{peg}(0,0), \\
\mathsf{peg}(1,0), \mathsf{peg}(1,1), \\
\mathsf{peg}(2,0), \mathsf{empty}(2,1), \mathsf{peg}(2,2), \\
\mathsf{peg}(3,0), \mathsf{peg}(3,1), \mathsf{peg}(3,2), \mathsf{peg}(3,4), \\
\mathsf{peg}(4,0), \mathsf{peg}(4,1), \mathsf{peg}(4,2), \mathsf{peg}(4,3), \mathsf{peg}(4,4).
\end{array}$$

Now each possible move can be represented in the style of our earlier encodings as a linear implication. Because there are six possible directions for jumping (although many are impossible for any given peg), we have six different rules. We name each rule with its compass direction

$$\begin{array}{rcl}
sw & : & \mathsf{peg}(x,y) \otimes \mathsf{peg}(x+1,y) \otimes \mathsf{empty}(x+2,y) \\
& & \multimap \mathsf{empty}(x,y) \otimes \mathsf{empty}(x+1,y) \otimes \mathsf{peg}(x+2,y) \\
ne & : & \mathsf{peg}(x+2,y) \otimes \mathsf{peg}(x+1,y) \otimes \mathsf{empty}(x,y) \\
& & \multimap \mathsf{empty}(x+2,y) \otimes \mathsf{empty}(x+1,y) \otimes \mathsf{peg}(x,y) \\
e & : & \mathsf{peg}(x,y) \otimes \mathsf{peg}(x,y+1) \otimes \mathsf{empty}(x,y+2) \\
& & \multimap \mathsf{empty}(x,y) \otimes \mathsf{empty}(x,y+1) \otimes \mathsf{peg}(x,y+2) \\
w & : & \mathsf{peg}(x,y+2) \otimes \mathsf{peg}(x,y+1) \otimes \mathsf{empty}(x,y) \\
& & \multimap \mathsf{empty}(x,y+2) \otimes \mathsf{empty}(x,y+1) \otimes \mathsf{peg}(x,y) \\
se & : & \mathsf{peg}(x,y) \otimes \mathsf{peg}(x+1,y+1) \otimes \mathsf{empty}(x+2,y+2) \\
& & \multimap \mathsf{empty}(x,y) \otimes \mathsf{empty}(x+1,y+1) \otimes \mathsf{peg}(x+2,y+2) \\
nw & : & \mathsf{peg}(x+2,y+2) \otimes \mathsf{peg}(x+1,y+1) \otimes \mathsf{empty}(x,y) \\
& & \multimap \mathsf{empty}(x+2,y+2) \otimes \mathsf{empty}(x+1,y+1) \otimes \mathsf{peg}(x,y)
\end{array}$$

In order to specify the goal, we can specify the precise desired configuration. In this example we see if we won by counting the number of moves, so we can add $\mathsf{count}(0)$ to the initial state, $\mathsf{count}(n)$ to the left-hand side of every implication and $\mathsf{count}(n+1)$ to the right-hand side. To solve the puzzle we then have to prove

$$\Gamma_0; \Delta_0, \mathsf{count}(0) \Longrightarrow \mathsf{count}(13) \otimes \top$$

where Γ_0 is the program above and Δ_0 is the representation of the initial situation.

This representation is adequate, but unfortunately it does not fall within the class of linear hereditary Harrop formulas because of its use of $\otimes$. In fact, if we look at the seqent we have to prove above to solve the puzzle, the proof will not proceed in a goal-directed fashion. Instead, we have to continually focus on propositions in Γ_0 until the goal happens to be provable in the end.

Fortunately, we can transfer it into the class LHHF by taking advantage of two ideas. The first is a logical law called *uncurrying*:

$$(A \otimes B) \multimap C \dashv\vdash A \multimap (B \multimap C)$$

This allows us to eliminate simultanous conjunction on the left-hand side of linear implications. But what about the right-hand side? In this case no similar local transformation exists. Instead we transform the whole program by using a so-called A-translation or continuation-passing transform.[6] Normally, a move $A \multimap B$ is a function that transforms the resource A into the goal B. However, instead of returning B, we will now add a second argument that consumes the result B and eventually returns a final answer. By the substitution principle (or cut, in the sequent calculus), this is sound and complete. So $A \multimap B$ is transformed into $A \multimap (B \multimap p) \multimap p$ where p is a new atomic predicate. Note that this shifts B to the left-hand side of an implication, so we can now apply uncurrying in case it is a tensor. In general, all clauses of the program need to be transformed with the same new propositional parameter or constant p.

If we read the transformed proposition operationally (also changing the order of subgoals),

$$p \circ\!\!- A \circ\!\!- (B \multimap p),$$

it means that in order to prove p we have to prove A (which may consume some resources), then assume B and make a recursive call. In our example, we call the new predicate *jump* and the first clause

$$\begin{array}{rcl} sw & : & \mathsf{peg}(x,y) \otimes \mathsf{peg}(x+1,y) \otimes \mathsf{empty}(x+2,y) \\ & & \multimap \mathsf{empty}(x,y) \otimes \mathsf{empty}(x+1,y) \otimes \mathsf{peg}(x+2,y) \end{array}$$

is transformed into

$$\begin{array}{rcl} sw & : & \mathsf{jump} \circ\!\!- (\mathsf{peg}(x,y) \otimes \mathsf{peg}(x+1,y) \otimes \mathsf{empty}(x+2,y)) \\ & & \circ\!\!- (\mathsf{empty}(x,y) \otimes \mathsf{empty}(x+1,y) \otimes \mathsf{peg}(x+2,y) \multimap \mathsf{jump}) \end{array}$$

and similarly for the other clauses. Now we can eliminate the simultaneous conjunction by uncurrying to obtain an equivalent proposition in the LHHF

[6] [*add citations*]

fragment.

$$\begin{array}{rcl} sw & : & \mathsf{jump} \circ\!\!- \mathsf{peg}(x,y) \circ\!\!- \mathsf{peg}(x+1,y) \circ\!\!- \mathsf{empty}(x+2,y) \\ & & \circ\!\!- (\mathsf{empty}(x,y) \multimap \mathsf{empty}(x+1,y) \multimap \mathsf{peg}(x+2,y) \multimap \mathsf{jump}) \end{array}$$

In order to count the number of moves we just add an argument to the jump predicate that we count down to zero.

$$\begin{array}{rcl} sw & : & \mathsf{jump}(n+1) \circ\!\!- \mathsf{peg}(x,y) \circ\!\!- \mathsf{peg}(x+1,y) \circ\!\!- \mathsf{empty}(x+2,y) \\ & & \circ\!\!- (\mathsf{empty}(x,y) \multimap \mathsf{empty}(x+1,y) \multimap \mathsf{peg}(x+2,y) \multimap \mathsf{jump}(n)) \\ ne & : & \mathsf{jump}(n+1) \circ\!\!- \mathsf{peg}(x+2,y) \circ\!\!- \mathsf{peg}(x+1,y) \circ\!\!- \mathsf{empty}(x,y) \\ & & \circ\!\!- (\mathsf{empty}(x+2,y) \multimap \mathsf{empty}(x+1,y) \multimap \mathsf{peg}(x,y) \multimap \mathsf{jump}(n)) \\ e & : & \mathsf{jump}(n+1) \circ\!\!- \mathsf{peg}(x,y) \circ\!\!- \mathsf{peg}(x,y+1) \circ\!\!- \mathsf{empty}(x,y+2) \\ & & \circ\!\!- (\mathsf{empty}(x,y) \multimap \mathsf{empty}(x,y+1) \multimap \mathsf{peg}(x,y+2) \multimap \mathsf{jump}(n)) \\ w & : & \mathsf{jump}(n+1) \circ\!\!- \mathsf{peg}(x,y+2) \circ\!\!- \mathsf{peg}(x,y+1) \circ\!\!- \mathsf{empty}(x,y) \\ & & \circ\!\!- (\mathsf{empty}(x,y+2) \multimap \mathsf{empty}(x,y+1) \multimap \mathsf{peg}(x,y) \multimap \mathsf{jump}(n)) \\ se & : & \mathsf{jump}(n+1) \circ\!\!- \mathsf{peg}(x,y) \circ\!\!- \mathsf{peg}(x+1,y+1) \circ\!\!- \mathsf{empty}(x+2,y+2) \\ & & \circ\!\!- (\mathsf{empty}(x,y) \multimap \mathsf{empty}(x+1,y+1) \multimap \mathsf{peg}(x+2,y+2) \multimap \mathsf{jump}(n)) \\ nw & : & \mathsf{jump}(n+1) \circ\!\!- \mathsf{peg}(x+2,y+2) \circ\!\!- \mathsf{peg}(x+1,y+1) \circ\!\!- \mathsf{empty}(x,y) \\ & & \circ\!\!- (\mathsf{empty}(x+2,y+2) \multimap \mathsf{empty}(x+1,y+1) \multimap \mathsf{peg}(x,y) \multimap \mathsf{jump}(n)) \end{array}$$

Finally we add a clause to succeed if we can make n moves given the query $\mathsf{jump}(n)$.

$$done \quad : \quad \mathsf{jump}(0) \circ\!\!- \top$$

Note the use of $\top$ in order to consume the state at the end of the computation.

The runnable implementation of this program in the concrete syntax of Lolli can be found in the Lolli distribution in the directory `examples/solitaire/`. In order to circumvent the problem that Lolli does not show proofs, but only instantiations for the free variables in the query, we can add another argument to the jump predicate to construct a sequence of moves as it executes.

5.5 Logical Compilation

In this section we examine means to compile program clauses, which may reside either in Γ or Δ. We do not push the ideas very far, but we want to give some intuition on how more efficient and lower level compilation strategies can be developed.

The main question is how to compile procedure call, which, as we have seen, corresponds to applying focusing on the left once the goal is atomic. Instead of working with focusing rules on the left, we would like to translate the program clause to a residual subgoal, still is logically described but which can be interpreted as providing instructions for search (and thereby for computation). More formally, we introduce three new judgments. Note that propositions A

stand for linear hereditary Harrop formulas as before, while G stands for residual formulas whose formal definition we postpone until we have considered what is needed.

$\Gamma; \Delta \Longrightarrow A \upuparrows$	A follows by resolution from $\Gamma; \Delta$
$A \gg P \setminus G$	A immediately entails goal P with residual subgoal G
$\Gamma; \Delta \Longrightarrow G \uparrow$	residual subgoal G has a resolution proof from $\Gamma; \Delta$

The resolution judgment $A \upuparrows$ arises from the uniform proof judgment $A \Uparrow$ by copying all right rules except the choice rules

$$\frac{\Gamma; \Delta; A \Downarrow \Longrightarrow P}{\Gamma; \Delta, A \Longrightarrow P \Uparrow}\,\mathsf{decideL} \qquad \frac{\Gamma, A; \Delta; A \Downarrow \Longrightarrow P}{\Gamma, A; \Delta \Longrightarrow P \Uparrow}\,\mathsf{decideL!}$$

which are replaced by

$$\frac{A \gg P \setminus G \qquad \Gamma; \Delta \Longrightarrow G \uparrow}{\Gamma; \Delta, A \Longrightarrow P \upuparrows}\,\mathsf{decideL}{\gg}$$

$$\frac{A \gg P \setminus G \qquad \Gamma, A; \Delta \Longrightarrow G \uparrow}{\Gamma, A; \Delta \Longrightarrow P \upuparrows}\,\mathsf{decideL!}{\gg}$$

We design the residuation $A \gg P \setminus G$ to be parametric in P, so the translation of A to G does not depend on the precise form of P. In other words, we compile A independently of the form of the call—relaxing this would allow some inlining optimizations. Furthermore, we must be careful that compilation always succeeds and gives a uniquely defined result. That is, for every A and P there exists a unique residual goal G such that $A \gg P \setminus G$. Therefore the natural form of definition is inductive on the structure of A.

On the other hand it turns out our propositions A are not general enough to express residual goals, so we need some additional propositions. We will see for each connective what we need and collect them at the end. Note that if $A \gg P \setminus G$ then a uniform proof with focus formula $A \Downarrow$ and goal P should correspond to a proof of the residual goal $G \uparrow$.

Atomic Propositions. If our focus formula is atomic, then we have to residuate an equality check. Operationally, this will be translated into a call to unification.

$$\frac{}{P' \gg P \setminus P' \doteq P}$$

The right rule for this new connective is clear: it just establishes the equality.

$$\frac{}{\Gamma; \cdot \Longrightarrow P \doteq P \uparrow}\doteq\mathrm{R}$$

For the left rule (which is not needed here), see Exercise 5.3.

Linear Implication. If the focus formula is $A_1 \multimap A_2$, then A_1 is must be solved as a subgoal. In addition, the residuation of A_2 yields another subgoal G. These must be joined with simultaneous conjunction in order to reflect the context split in the $\multimap$L rule.

$$\frac{A_2 \gg P \setminus G}{A_1 \multimap A_2 \gg P \setminus G \otimes A_1}$$

The order is determined by our general convention on conjunctive non-determinism: we first solve G and then A, so G is written on the left. The corresponding right rule for $\otimes$ is familiar, but used here in a slight different form, so we restate it explicitly.

$$\frac{\Gamma; \Delta_1 \Longrightarrow G \uparrow \qquad \Gamma; \Delta_2 \Longrightarrow A \uparrow\uparrow}{\Gamma; \Delta_1, \Delta_2 \Longrightarrow G \otimes A \uparrow} \otimes\mathrm{R}$$

Of course, we use resource management as shown before to postpone the splitting of $\Delta = (\Delta_1, \Delta_2)$ when using this rule during logic program execution.

Alternative Conjunction. If the focus formula is an alternative conjunction $A_1 \& A_2$, then we must choose either to use the $\&\mathrm{L}_1$ or $\&\mathrm{L}_2$ rule. This choice is residuated into an external choice, for which we have the two corresponding rules $\oplus\mathrm{R}_1$ and $\oplus\mathrm{R}_2$.

$$\frac{A_1 \gg P \setminus G_1 \qquad A_2 \gg P \setminus G_2}{A_1 \& A_2 \gg P \setminus G_1 \oplus G_2}$$

Again, we repeat the two right rules for $\oplus$.

$$\frac{\Gamma; \Delta \Longrightarrow G_1 \uparrow}{\Gamma; \Delta \Longrightarrow G_1 \oplus G_2 \uparrow} \oplus\mathrm{R}_1 \qquad \frac{\Gamma; \Delta \Longrightarrow G_2 \uparrow}{\Gamma; \Delta \Longrightarrow G_1 \oplus G_2 \uparrow} \oplus\mathrm{R}_2$$

Additive Truth. If the focus formula is $\top$ then there is no left rule that can be applied and focusing fails. Correspondingly, there is no right rule for $\mathbf{0}$, so $\top$ residuates to $\mathbf{0}$.

$$\frac{}{\top \gg P \setminus \mathbf{0}}$$

As usual, this can be seen as a zero-ary alternative conjunction. And there is no right rule for $\mathbf{0} \uparrow$.

Unrestricted Implication. If the focus formula is an unrestricted implication $A_1 \supset A_2$ we have to solve A_1 as a subgoal, but with access to any of the linear resources.

$$\frac{A_2 \gg P \setminus G}{A_1 \supset A_2 \gg P \setminus G \otimes! A_1}$$

Here we should think of $\otimes!$ as a single binary connective, a point overlooked in [CHP00]. The reason is that it should be statically obvious when solving G

in a proposition $G \otimes! A_1$ that A_1 cannot consume any resources remaining from G. The new connective is characterized by the following right rule.

$$\frac{\Gamma; \Delta \Longrightarrow G \uparrow \qquad \Gamma; \cdot \Longrightarrow A \uparrow\uparrow}{\Gamma; \Delta \Longrightarrow G \otimes! A \uparrow} \otimes! \mathrm{R}$$

For the left rule (which is not needed here) see Exercise 5.2.

Universal Quantification. The universal quantifier is simply residuated into a corresponding existential quantifier.

$$\frac{[a/x]A \gg P \setminus [a/x]G}{\forall x.\, A \gg P \setminus \exists x.\, A}\, a$$

where a must be a new parameter. The new version of the right rule is just as before with a new judgment

$$\frac{\Gamma; \Delta \Longrightarrow [t/x]G \uparrow}{\Gamma; \Delta \Longrightarrow \exists x.\, G \uparrow} \exists \mathrm{R}$$

Thus we needed the following language of *residual goals* G where A ranges over linear hereditary Harrop formulas.

$$G \quad ::= \quad P' \doteq P \mid G \otimes A \mid G_1 \oplus G_2 \mid \mathbf{0} \mid G \otimes! A \mid \exists x.\, G$$

We call the system of uniform proofs where we replace the choice rules $\mathsf{decideL}$ and $\mathsf{decideL!}$ with $\mathsf{decideL}\gg$ and $\mathsf{decideL!}\gg$ the system of *resolution*. Now the have the following theorem asserting the correctness of resolution.

Theorem 5.3 (Correctness of Resolution)
$\Gamma; \Delta \Longrightarrow A \Uparrow$ *if and only if* $\Gamma; \Delta \Longrightarrow A \uparrow\uparrow$.

Proof: See Exercise 5.1 □

To see how subgoal residuation is related to compilation, consider a simple concrete program that implements a test if a natural number in unary presentation is even.

$$\mathsf{even}(0) \& \forall x.\, \mathsf{even}(x) \multimap \mathsf{even}(\mathsf{s}(\mathsf{s}(x)))$$

Let us call the proposition E. For illustration purposes, we have combined the unrestricted clauses defining even into one using alternative conjunction, because *unrestricted* assumptions A, B are equivalent to a single unrestricted assumption $A \& B$. If we have generic goal $\mathsf{even}(t)$ for some (unknown) term t, we can calculate

$$E \gg \mathsf{even}(t) \setminus \mathsf{even}(0) \doteq \mathsf{even}(t) \oplus \exists x.\, \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) \doteq \mathsf{even}(t) \otimes \mathsf{even}(x)$$

Now we can read the new connectives in the residual goal as search instructions from left to right:

1. Unify ($\doteq$) the proposition $\mathsf{even}(0)$ with the goal $\mathsf{even}(t)$.
2. If this fails ($\oplus$) then create a new existential variable X ($\exists x$).
3. Then unify ($\doteq$) the proposition $\mathsf{even}(\mathsf{s}(\mathsf{s}(X)))$ with the goal $\mathsf{even}(t)$.
4. If this succeeds ($\otimes$) continue with a recursive call to even with X.

Given the logical reading, we can reason about optimizations using the laws of linear logic. For example, $\mathsf{even}(0) \doteq \mathsf{even}(t)$ succeeds exactly if $0 \doteq t$. In the second line we can unfold the unification further, since one of the terms to be unified is known to be $\mathsf{s}(\mathsf{s}(x))$. This kinds of optimization naturally lead to a higher-level version of the Warren Abstract Machine (WAM) [AK91] which is the basis for almost all modern Prolog compilers.

5.6 Exercises

Exercise 5.1 Carefully prove the correctness of resolution (Theorem 5.3). Explicitly state and prove any generalized induction hypotheses or lemmas you might need.

Exercise 5.2 We consider the new connective $\otimes!$ from subgoal residuation as a connective in full intuitionistic linear logic $A \otimes! B$. The right rule in the sequent calculus is determined by the right rule given in Section 5.5.

$$\frac{\Gamma; \Delta \Longrightarrow A \qquad \Gamma; \cdot \Longrightarrow B}{\Gamma; \Delta \Longrightarrow A \otimes! B} \otimes! \mathrm{R}$$

This corresponds directly to the introduction rule in natural deduction.

1. Show the elimination rule(s) in the natural deduction.
2. Show these rule(s) to be locally sound and complete.
3. Show the corresponding left rule(s) in the sequent calculus.
4. Show the new essential cases in the proof of admissibility of cut.

This demonstrates that $\otimes!$ can be seen as a first-class connective of linear logic, even if $A \otimes! B$ may also be considered to as a shorthand for $A \otimes (!B)$.

Exercise 5.3 We consider the new connective $\doteq$ from subgoal residuation as a connective in full intuitionistic linear logic $A \doteq B$. The right rule in the sequent calculus is determined by the right rule given in Section 5.5:

$$\frac{}{\Gamma; \cdot \Longrightarrow A \doteq A} \doteq \mathrm{R}$$

As usual, this corresponds directly to the introduction rule in natural deduction.

1. Show the elimination rule(s) in the natural deduction.
2. Show these rule(s) to be locally sound and complete.
3. Show the corresponding left rule(s) in the sequent calculus.
4. Show how to modify the proof of admissibility of cut to account for the new connective.

Exercise 5.4 Extend the translation to A-normal form, as used in the encoding of the solitaire game, to arbitrary propositions in linear logic. It should have the property that each proposition is translated into a corresponding proposition using only the additional predicate p. State explicitly which laws you need that are similar to uncurrying to eliminate gratuitous uses of propositions that are left asynchronous.

Exercise 5.5 Implement the Towers of Hanoi puzzle with three pegs and several disks in Lolli.

Exercise 5.6 Define a translation that maps a regular expressions r to a predicates p and some unrestricted linear hereditary Harrop formulas. This translation, to be described on paper, should have the property that a word w is in the language of r if and only if $p(w)$ can be proven by the logic programming interpreter underlying Lolli. Implement words as list of single character constants and make sure to pay attention to termination issues.

Apply your translation to $r = a(b + c)^*a$ and execute the resulting program on several successful and failing queries.

Bibliography

[ABCJ94] D. Albrecht, F. Bäuerle, J. N. Crossley, and J. S. Jeavons. Curry-Howard terms for linear logic. In ??, editor, *Logic Colloquium '94*, pages ??–?? ??, 1994.

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[ACS98] Roberto Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchrounous π-calculus. *Theoretical Computer Science*, 195(2):291–423, 1998.

[AK91] Hassan Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, 1991.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

[AP91] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems: A linear logic approach. In P. Schröder-Heister, editor, *Proceedings of Workshop to Extensions of Logic Programming, Tübingen, 1989*, pages 1–30. Springer-Verlag LNAI 475, 1991.

[Bar96] Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Department of Computer Science, University of Edinburgh, September 1996.

[Bib86] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[Bie94] G. Bierman. On intuitionistic linear logic. Technical Report 346, University of Cambridge, Computer Laboratory, August 1994. Revised version of PhD thesis.

[BS92] G. Bellin and P. J. Scott. On the π-calculus and linear logic. Manuscript, 1992.

[Cer95] Iliano Cervesato. Petri nets and linear logic: a case study for logic programming. In M. Alpuente and M.I. Sessa, editors, *Proceedings of the Joint Conference on Declarative Programming (GULP-PRODE'95)*, pages 313–318, Marina di Vietri, Italy, September 1995. Palladio Press.

[CHP00] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1–2):133–163, February 2000. Special issue on Proof Search in Type-Theoretic Languages, D. Galmiche and D. Pym, editors.

[Doš93] Kosta Došen. A historical introduction to substructural logics. In Peter Schroeder-Heister and Kosta Došen, editors, *Substructural Logics*, pages 1–30. Clarendon Press, Oxford, England, 1993.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. Translated under the title *Investigations into Logical Deductions* in [Sza69].

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir93] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[Her30] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et de Lettres de Varsovic*, 33, 1930.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[Hod94] Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[HP97] James Harland and David Pym. Resource-distribution via boolean constraints. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, pages 222–236, Townsville, Australia, July 1997. Springer-Verlag LNAI 1249.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'91)*, pages

133–147, Geneva, Switzerland, July 1991. Springer-Verlag LNCS 512.

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII, September 1976.

[Kni89] Kevin Knight. Unification: A multi-disciplinary survey. *ACM Computing Surveys*, 2(1):93–124, March 1989.

[Lin92] P. Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, Spring 1992.

[Mil92] D. Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of the Workshop on Extensions of Logic Programming*, pages 242–265. Springer-Verlag LNCS 660, 1992.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[MM76] Alberto Martelli and Ugo Montanari. Unification in linear time and space: A structured presentation. Internal Report B76-16, Istituto di Elaborazione delle Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, July 1976.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[MOM91] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey. *Journal on Foundations of Computer Science*, 2(4):297–399, December 1991.

[PD01] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

[Pra65] Dag Prawitz. *Natural Deduction.* Almquist & Wiksell, Stockholm, 1965.

[PW78] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, April 1978.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[Rob71] J. A. Robinson. Computational logic: The unification computation. *Machine Intelligence*, 6:63–72, 1971.

[Sce93] A. Scedrov. A brief guide to linear logic. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 377–394. World Scientific Publishing Company, 1993. Also in Bulletin of the European Association for Theoretical Computer Science, volume 41, pages 154–165.

[SHD93] Peter Schroeder-Heister and Kosta Došen, editors. *Substructural Logics*. Number 2 in Studies in Logic and Computation. Clarendon Press, Oxford, England, 1993.

[Sza69] M. E. Szabo, editor. *The Collected Papers of Gerhard Gentzen*. North-Holland Publishing Co., Amsterdam, 1969.

[Tro92] A. S. Troelstra. *Lectures on Linear Logic*. CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford, California, 1992.

[Tro93] A. S. Troelstra. Natural deduction for intuitionistic linear logic. Prepublication Series for Mathematical Logic and Foundations ML-93-09, Institute for Language, Logic and Computation, University of Amsterdam, 1993.