August 1991

Bounded Linear Logic

Jean-Yves Girard
University of Pennsylvania

Andre Scedrov
University of Pennsylvania

Philip J. Scott
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Sciences Technical Report No. MS-CIS-91-59.

Bounded Linear Logic

Abstract

A typed, modular paradigm for polynomial time computation is proposed.

Comments

University of Pennsylvania Department of Computer and Information Sciences Technical Report No. MS-CIS-91-59.

Bounded Linear Logic: A Modular Approach to Polynomial Time Computability

MS-CIS-91-59
LOGIC & COMPUTATION 36

Jean-Yves Girard
Andre Scedrov
Philip J. Scott

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

August 1991

Bounded Linear Logic :

A Modular Approach to Polynomial Time Computability

Jean-Yves Girard Andre Scedrov Philip J. Scott

1 Introduction

In recent years, especially with the development of large-scale computation and with the possibility of machines making dramatic decisions, issues such as software reliability, maintenance, and verification have become essential in theoretical computer science. In other words, the study of these topics and of *program specifications* as a means of facilitating them is now central, independently of traditional tenets such as the search for good algorithms.

There are many forms of specifications. For example, one can think of input/output specifications, among the most basic being when one is asked to specify that an input is an integer. There are also probabilistic specifications, when a certain percentage of error is allowed. One can also have complexity specifications about space or time needed to execute a program. In all cases the specifications are well-defined mathematical properties, which can be expressed in the usual formalism for mathematics, even if they are somewhat unusual from the viewpoint of standard mathematics. In particular a classical mathematical proof that an algorithm meets a given specification will be considered as completely satisfactory, even if the proof is not constructive. The situation changes radically, however, if one now insists on a standardized, modular, "industrial" approach to the question.

Since there has been much discussion about the merits of typed vs. untyped programming paradigms, and since our paper proposes yet another typed calculus, it is appropriate to say at the outset that if one absolutely insists on efficient programs, one should use languages that allow maximum flexibility. However, if one is interested in reliable programs that can be maintained in a changing systems environment, then a language will have to be concerned with specifications. In the first case one shall get a *hand crafted* or *one off* product that may be a marvel or might contain awful bugs–and such programs use so many "tricks" that proofs of their correctness with respect to a given specification will be rather exceptional. In the second case the programming methods are limited–usually by a rather awkward system of basic specifications called *types* – but the product is guaranteed a certain

industrial level of "quality control" . [1] The two forms of programming will coexist forever, and the fact that the industrial approach is–at present–very inefficient is not an argument for the superiority of the other "hand crafted" one. In particular, it does not make the slightest difference that the system that we are presenting here is so delicate to use that in its present form it has hardly any practical value. Rather, our paper shows the possibility of a logical (i.e., standardized, modular, industrial) approach to complexity specifications, and *this* phenomenon is radically new.

The industrial approach to specifications therefore relies on an integrated computational paradigm, where basic specifications are built together with the algorithms. At compilation time these basic specifications are checked, then erased to produce machine code. This is the description of the *typed* approach; in this approach we can combine instructions only in certain cases. Each program and subprogram is assigned a type which describes in shorthand which pluggings it accepts : for instance the type A *implies* B given to a program means that one can see it as a function waiting for an input labeled A and which can in turn give an output labeled B. Hence the main activity of typed programming is to match types, which can often be difficult in the extant systems.

The activity of manipulating types was recognized a long time ago as analogous to proving theorems in intuitionistic logic–this is now technically known as the Curry-Howard isomorphism (or the propositions-as -types paradigm)–but the origins of this idea date back to old intuitionism in the early 1900s and especially to Heyting and Kolmogorov in the 1920s. The situation is actually more involved: the idea makes better sense when combined with formalist tradition; logic offers not only a paradigm for basic specifications but also a mode of execution, namely through cut-elimination or its variants, e.g., natural deduction, all dating back to Gentzen's work in the 1930s. Here we show that this paradigm "computation as cut elimination" is flexible enough to express a notion of *feasible computation.*

There are many *typed lambda calculi* , among them is system $\mathcal{F}$ introduced in 1970 by one of the authors [12, 20]. This system is characterized by an extreme economy of means and the possibility of easy definition of many current specifications. Although the system is far from being as flexible as we would like it to be, it is fair to say that $\mathcal{F}$ or its various improvements do not look ridiculous as integrated

[1] The broader distinction between *hand crafted* and *industrial* may be illustrated by the fact that a product such as the "daily special" in a restaurant with traditional cuisine can be very good or very bad, with its price bearing no relation to its quality, whereas an industrial product such as fast food is just a "safe bet"–neither very good nor very bad–and with a good ratio between price and quality. Moreover, any quarrel between these two activities is nonsense as long as one of them does not pretend to invade the domain of the other–domains which are distinguished by verbs like *to dine* versus *to eat* . Another important difference between one off and industrial products usually becomes apparent when the product breaks down. Just imagine trying to get an antique clock repaired. It is very likely that the entire clock will have to be replaced or some of its parts will have to be custom made (at a great cost) because the parts or even the materials may be unavailable. On the other hand, if a modern watch breaks down, it probably needs just a new module. In other words, a one off product is an indivisible object, while an industrial product is more an idea of an object, realized by exchangeable parts.

programming paradigms. The expressive power of $\mathcal{F}$ is immense and precisely for that reason, because it accepts too many functions, $\mathcal{F}$ only yields input/output specifications. Here let us be precise : it is true that we can abstractly measure the expressive power of typed systems, and that for instance certain restrictions can drastically lower the complexity of typed algorithms, but all current restrictions cannot be detected from the viewpoint of feasibility. For example, some authors insist on the impredicativity of $\mathcal{F}$ as a defect–as if one were still living in a world of lurking paradoxes–but restricting $\mathcal{F}$ to so-called ramified systems contradicts the flexibility of typing without yielding any feasibly detectable lowering of complexity. Rather, the issue lies already on the propositional level. One must keep in mind that even simply typed calculi have complexity measures which are towers of exponentials.

In fact, prior to this paper there has been no example of an integrated typed system yielding complexity specifications; the input/output style of specification is the only one that has been treated. For instance, we understand that probabilistic specification needs a kind of probabilistic logic which is sorely missing.

Before arriving at our solution, let us try to position it with respect to vaguely related works on the theme logic and complexity. There are obvious solutions to obtain a feasibly typed system; e.g., take system $\mathcal{F}$ and a clock: instead of typing something INT *implies* INT (from integers to integers) we can type it INT *implies* (ERR + INT), which means that when the time is over, the program (if it has not finished computing) returns an error message. The brutal character of this answer is plain but the true reason for its stupidity is not completely obvious: in our opinion this system guarantees temporal specifications but is no longer able to guarantee input/output specifications, which are much more basic!

A more elaborate system, Cook and Urquhart's feasible functionals of finite type [8, 7], proposes a generalization of polynomial time to finite types. Although this system is interesting in its own right, it is not helpful for our purposes because its higher type dependencies are not polynomial but hyperexponential. Furthermore, it cannot be used to check that a numerical function is polynomial time, since it takes as primitive all polynomial time functions.

The first connection between proof theory and polynomial time computability was established in the work of Buss [3, 2], who introduced first-order systems of "bounded arithmetic" with proof-theoretic strength corresponding to polynomial time computation and in which precisely the polynomial time functions could be defined by certain formulas. In this work the logical systems are external to the computational paradigm itself, which is given by a Turing machine with a clock (in the form of "bounded primitive recursion"). As far as we know, there has been no proposal of an integrated paradigm given by the inherent structure of proofs in the systems of bounded arithmetic.

The originally interesting idea of a polynomially graded logic [27] did produce a type system with explicit resource bounds, but it stumbled on the impossibility of having a notion of higher type feasible functional without artificial complications, which were partly caused by superimposing the external computational paradigm

of a Turing machine with a clock onto the Curry-Howard paradigm. As we observed above, the notion proposed by Cook et al. is not really an answer since the higher type dependencies are hyperexponential, i.e., the inner structure of the systems in [8, 7] is not polynomial. In [27] the authors actually got closer to a true solution; unfortunately it was necessary to introduce indices of polynomial time machines, i.e., to presuppose here too a polynomial time structure.

Outside proof theory, it is difficult not to be struck by the work on finite model theory [24, 31, 22, 23, 25], which has nothing to do *a priori* with typing but which has the inner coherence that is missing in all the approaches just discussed : in the finite model theory approach, polynomials appear for combinatorial reasons, simply because DATALOG computations can be bounded by means of binomial coefficients. Here it is important to remark that the computational mechanism of DATALOG itself (given by its forward chaining style) is not addressed by logical characterizations stemming from finite model theory. However, this lack of an integrated paradigm is compensated for by non-trivial results of Immerman [24] and Vardi [31], who show that polynomial time functions receive a definition not in terms of polynomial time. This is obviously a great logical achievement, even if we are not seeking this kind of logical analysis here. What we shall present below has very little to do with this approach but the fact is that we will also derive polynomial time from something not presupposing polynomial time and that at a certain point combinatorial polynomials will play an essential role. In the future a connection between these two approaches would be very exciting, but proof theory and model theory are often such orthogonal approaches that one should not daydream too much.

Our aim in this paper is not just to express polynomial time *computability* as *provability* of formulas in a logical system, but to provide a notion of polynomial time *computation* intrinsically within a logical system, according to the Curry-Howard paradigm "computation as cut elimination". In other words, we propose a modular typed computation paradigm as an alternative to the paradigm of a Turing machine with a clock. Let us start with the idea of a *complexity* specification (not necessarily polynomial time). The above discussion of previous typed attempts makes it plain that taking the actual complexity as a primitive parameter does not lead to an integrated typed system. This should not be too surprising; in mathematics certain interesting notions never led to conceptualization for want of modularity (for instance, there is a non-commutative theory of groups but no theory of non-commutative groups). We therefore seek a notion more primitive than complexity, i.e., something that can produce complexity restrictions but which cannot be reduced to complexity. In our approach, this more primitive notion will basically be the contraction rule of Gentzen sequent calculus, to which Linear Logic [13] gives a very special status. It will turn out that controlling contraction is an indirect way of controlling time complexity, but for instance no way–direct or indirect–is known of controlling space complexity in this manner. (Does this mean that space complexity has no *logical* meaning in the sense above, or does this only indicate that our present tools are desperately poor?)

A first approximation to Linear Logic is Rudimentary Linear Logic (RLL) , i.e., sequent calculus without weakening and contraction. One of the most immediate features of RLL is its linear time normalization procedure: in fact it is easy to see that the number of rules strictly decreases during normalization, i.e., the procedure can be carried out in a kind of "shrinking space" which forces linear time. Moreover, and this should be emphasized, the situation still holds with full second order. So we begin to see a possible restriction of system $\mathcal{F}$ along feasibility lines. This is a fantastic medicine with respect to problems of complexity, except that the patient is dead! Without contraction the expressive power of logic is so weak that one can hardly program more than programs permuting the components of a pair.

Fortunately, Linear Logic is not about the removal of contraction and weakening, but about their transformation into logical rules for special connectives, the so-called *exponentials* ! and ?. In fact by allowing the use of exponentials we compensate for the drastic fall of expressive power, compensate so much that we get the usual intuitionistic systems (this is no surprise since Linear Logic has been carefully designed to obtain roughly the same expressive power). Therefore our only hope is to seek an intermediate system between RLL and full Linear Logic. The first attempt will be successful.

Linear Logic is based on the idea of *resources* , an idea violently negated by the contraction rule. The contraction rule states precisely that a resource is potentially infinite, which is often a sensible hypothesis, but not always. The symbol ! can be used precisely to distinguish those resources for which there are no limitations. From a computational point of view $!A$ means that the datum A is stored in the memory and may be referenced an unlimited number of times. In some sense, $!A$ means "A forever". In these times of great utopias falling, "forever" is no longer a viable expression, and in Bounded Linear Logic (BLL) it is replaced by more realistic goals: reuse will be possible, but only a certain number of times limited in advance. Instead of $!A$ BLL features bounded reuse operators $!_x A$, which intuitively mean that the datum A is stored in the memory and may be referenced up to x times. Now the fundamental point about BLL is that the basic properties involve polynomials. This is very easy to understand: the worst use of a bounded exclamation mark $!_x A$ is morally the same as using A *and* A *and* ... *and* A, x times, when *and* is given the technical meaning of the multiplicative conjunction *times* ($= \otimes$). When we thus interpret the rules of exclamation mark we see polynomials occuring in a very natural way. In fact this translation can also be applied to proofs ; in this case we discover that we have a locally polynomial translation from BLL to RLL: $!_2 \ldots !_2 A$ (n times) translates to $\bigotimes_{2^n} A$, but $!_{2x} \ldots !_{2x} A$ translates to $\bigotimes_{2^n x^n} A$, a bound which is a polynomial in x. This is why, whereas the full translation is exponential, we can speak of local polynomiality, here x^n. The combination of a locally polynomial procedure and a linear procedure (normalization in RLL) yields a locally polynomial time way of computing.

Technically speaking what we have just explained is only a rough justification of our main result stated below; this way of computing should not be used because

it is only *polynomial space.* Our main result states that if a function is typed in BLL from dyadic integers (i.e., integers in dyadic notation) to themselves, then normalization terminates in polynomially many steps in the length of the input (Theorem 4.4). Furthermore, in this case the size of normalization is also polynomial in the length of the input and as a consequence, the function is polynomial time (Theorem 5.3). Less unexpected, but of course also essential, is the converse theorem which says that all polynomial time functions can be obtained in this way, i.e., can be typed in BLL (Theorem 6.1). Although the idea is clear, the main result is technically difficult and uses very refined techniques of Linear Logic; the converse simply avoids the trap of RLL, i.e., a system with no expressive power. We have basically justified that the input/output ratio (in terms of the number of rules written) is locally polynomial. Common sense (and good experience in proof theory) is enough to convince one that the execution process itself should be polynomial time. If one is not satisfied with this kind of handwaving, then one must go into painful justifications, for instance, in this paper we use sophisticated techniques from linear logic, variations of proof nets. An example of what might happen is that in the presence of quantifiers, n iterated substitutions can produce an exponential blow-up. This is why at run-time we erase the types (i.e., formulas) and only keep an underlying graph (a kind of proof net), which is enough for input/output encoding, and the size of which remains tame. On the other hand, we must admit that our way of representing polynomial time functions in BLL is not very flexible and that in its present form BLL is not really practical ... so what *did* we achieve ?

Surely without the slightest doubt we have shown that a purely logical approach to complexity is not absurd. The polynomial time functions arise from a bounded exclamation mark and nothing else. In addition, BLL has the capability of directly specifying computational complexity on various data types, by using their representations as BLL types rather than encoding them as certain lists. (For instance we can type primality test of type TALLY INTEGERS *implies* BOOLE.) In BLL one also has all the usual facilities of polymorphism–even if we were a bit awkward here in making full use of them. From a strict logical viewpoint, as well as regarding the flexibility of the system, the main weakness of BLL is the presence of explicit resource parameters. Logic is useful only if it is implicit enough. If one prefers, logic is the maintenance of implicit data. At the moment we see no way to avoid mentioning the resources and still be able to synthesize them. An alternative way might be to forget resources by means of existential quantification over them, but unfortunately all our polynomial computations would break down immediately if we tried to do this. However, it must be noted that the resources occur in BLL only through input/output ratios and not at all as complexity measures: the complexity remains hidden and *this* is the reason why our approach avoids the problems encountered by previous ones.

2 Linear Logic

We recall Gentzen's sequent calculus for intuitionistic logic, e.g. from [20]. A *sequent* is a formal expression $\Gamma \vdash A$, where Γ is a finite list of formulas and A is a formula. One can informally interpret $\Gamma \vdash A$ as meaning "hypotheses Γ intuitionistically entail A."

Gentzen's sequent calculus involves three structural rules:

Exchange $$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

Weakening $$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

Contraction $$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

Although these rules are all problematic from the point of view of management of limited resources, contraction is by far the worst, cf. [13, 17, 29]. The contraction rule expresses unlimited capability of duplication; it is also the reason for the potential infinity of disjunctions in Herbrand expansions. In this paper we are especially interested in the effect of Contraction on cut elimination:

$$\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\,\omega\\ \Delta, A, A \vdash B\end{array}}{\Delta, A \vdash B}}{\Gamma, \Delta \vdash B} \quad \text{reduces to} \quad \dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\,\omega\\ \Delta, A, A \vdash B\end{array}}{\Gamma, \Delta, A \vdash B}}{\begin{array}{c}\Gamma, \Gamma, \Delta \vdash B\\ \vdots\\ \Gamma, \Delta \vdash B\end{array}}$$

Here, *one* cut is replaced by *two* cuts and by instances of Contraction and Exchange on the formulas in Γ. Also notice that this step requires duplication of the entire proof of $\Gamma \vdash A$. The Contraction rule is the reason why termination of cut elimination in intuitionistic propositional logic is not feasible (termination is well-known to be hyperexponential, see below.)

2.1 Rudimentary Linear Logic (RLL)

Linear Logic dispenses with the problematic structural rules Contraction and Weakening, [13]. In the absence of these structural rules, the propositional connectives assume a different character. We first discuss one extreme case in this vein, a rudimentary propositional system in which Contraction and Weakening

are removed altogether. Our intuitive description of the connectives of this rudimentary system is based on the propositions-as-types paradigm mentioned in the introduction.

The system RLL (Rudimentary Linear Logic) has *formulae* defined inductively from atomic formulae (propositional letters) $\alpha, \beta, \ldots$ by two binary propositional connectives:

(i) $A \otimes B$ (A *tensor* B = conjunction with no sharing of variables),

(ii) $A \multimap B$ (A *linearly implies* B = the type of functions looking at their argument exactly once)

Instead of stating the Exchange Rule explicitly, it is convenient to formulate sequents as formal expressions $\Gamma \vdash A$, where Γ is a finite multiset of formulas and A is a formula. The sequents satisfy the following axioms and rules:

Axiom: $A \vdash A$

Cut: $$\frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$$

Logical:

$$\otimes\mathrm{L} \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \otimes\mathrm{R} \quad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$$

$$\multimap\mathrm{L} \quad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C} \qquad \multimap\mathrm{R} \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

A *proof* in RLL of a sequent $\Gamma \vdash A$ is a finite labeled rooted tree in which the nodes are labeled by sequents in such a way that the leaves are labeled by instances of the Identity Axiom, the root is labeled by $\Gamma \vdash A$, and the label of each node is obtained from the label of its immediate precessor(s) by one instance of a rule of RLL. In an instance of a Cut, the formula denoted by A is called the *cut formula occurrence*. We often simply refer to "the cut formula " in a Cut, when the context is clear.

2.2 Normalization in RLL

We now state the reduction steps in normalization (i.e. cut elimination) in RLL and we give a measure # on proofs in RLL that yields a polynomial upper bound on the number of reduction steps. This measure will in fact be a bound on the number of instances of the rules, including the axioms, in the resulting cut-free proof. As for the number of reduction steps, if one counts all the reduction steps, including the so-called commutative reductions (all described below), the upper bound is cubic in our measure. However, there is a more subtle approach based on *proof nets* [13] that yields a linear upper bound.

Let #Axiom = 1. If a proof π is obtained from a proof ρ by a unary rule, let $\#\pi = \#\rho + 1$. If a proof π is obtained from proofs ρ and σ by a binary rule except Cut, let $\#\pi = \#\rho + \#\sigma + 1$. If a proof π is obtained from proofs ρ and σ by an

application of the Cut rule, let $\#\pi = \#\rho + \#\sigma$. (Because we are seeking a cut-free proof, we do not need to count the Cut rule.)

The following figures state the reduction steps. We simultaneously compute the measures.

2.2.1 Axiom Reductions

The reduction steps of this form apply when one premise of a Cut is an axiom. There are two cases:

Case AL

$$\dfrac{A \vdash A \qquad \begin{array}{c}\vdots\,\rho\\ \Delta, A \vdash B\end{array}}{\Delta, A \vdash B} \qquad \text{reduces to} \qquad \begin{array}{c}\vdots\,\rho\\ \Delta, A \vdash B\end{array} \quad .$$

Case AR

$$\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \qquad A \vdash A}{\Gamma \vdash A} \qquad \text{reduces to} \qquad \begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \quad .$$

In both cases, the measure decreases from $\#\rho + 1$ to $\#\rho$.

2.2.2 Symmetric Reductions

The intuitive motivation of these reductions is that they should apply when the left premise of a Cut comes by a logical right rule and the right premise comes by the corresponding left rule:

Case S$\otimes$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\pi\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\,\rho\\ \Delta \vdash B\end{array}}{\Gamma, \Delta \vdash A \otimes B} \qquad \dfrac{\begin{array}{c}\vdots\,\omega\\ \Lambda, A, B \vdash C\end{array}}{\Lambda, A \otimes B \vdash C}}{\Gamma, \Delta, \Lambda \vdash C}$$

$$\text{reduces to} \qquad \dfrac{\begin{array}{c}\vdots\,\rho\\ \Delta \vdash B\end{array} \qquad \dfrac{\begin{array}{c}\vdots\,\pi\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\,\omega\\ \Lambda, A, B \vdash C\end{array}}{\Gamma, \Lambda, B \vdash C}}{\Gamma, \Delta, \Lambda \vdash C}$$

Let $m = \#\pi$, $n = \#\rho$, $k = \#\omega$. Before reduction the measure is $m+n+k+2$; after reduction it is $m + n + k$.

Case S$\multimap$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\pi\\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A \multimap B} \qquad \dfrac{\begin{array}{c}\vdots\,\rho\\ \Delta \vdash A\end{array} \quad \begin{array}{c}\vdots\,\omega\\ \Lambda, B \vdash C\end{array}}{\Delta, \Lambda, A \multimap B \vdash C}}{\Gamma, \Delta, \Lambda \vdash C}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Delta \vdash A\end{array} \quad \begin{array}{c}\vdots\,\pi\\ \Gamma, A \vdash B\end{array}}{\Gamma, \Delta \vdash B} \qquad \begin{array}{c}\vdots\,\omega\\ \Lambda, B \vdash C\end{array}}{\Gamma, \Delta, \Lambda \vdash C}$$

Let $m = \#\pi$, $n = \#\rho$, $k = \#\omega$. Before reduction the measure is $m + n + k + 2$; after reduction it is $m + n + k$.

2.2.3 Commutative Reductions

The intuitive motivation for commutative reductions is simply to change the order in which an instance of a Cut appears in a proof. Commutative reductions should apply when at least one premise of a Cut is a consequence of a rule that does not operate on the cut formula.

Case CL$\otimes$L

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma, C, D \vdash A\end{array}}{\Gamma, C \otimes D \vdash A} \quad \begin{array}{c}\vdots\,\omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, C \otimes D \vdash B}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma, C, D \vdash A\end{array} \quad \begin{array}{c}\vdots\,\omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, C, D \vdash B}}{\Gamma, \Delta, C \otimes D \vdash B}$$

The measure is $\#\rho + \#\omega + 1$, both before and after this reduction step.

Case CL $\multimap$L:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash C\end{array} \quad \begin{array}{c}\vdots\,\sigma\\ \Lambda, D \vdash A\end{array}}{\Gamma, \Lambda, C \multimap D \vdash A} \quad \begin{array}{c}\vdots\,\omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Lambda, \Delta, C \multimap D \vdash B}$$

$$\text{reduces to}\qquad \dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\,\sigma\\ \Lambda, D \vdash A\end{array} \qquad \begin{array}{c}\vdots\,\omega\\ \Delta, A \vdash B\end{array}}{\Lambda, \Delta, D \vdash B}}{\Gamma, \Lambda, \Delta, C \multimap D \vdash B}$$

The measure is $\#\rho + \#\sigma + \#\omega + 1$ both before and after this reduction.

Case CR $\otimes$R:

$$\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\,\sigma\\ \Delta, A \vdash B\end{array} \qquad \begin{array}{c}\vdots\,\omega\\ \Lambda \vdash C\end{array}}{\Delta, \Lambda, A \vdash B \otimes C}}{\Gamma, \Delta, \Lambda \vdash B \otimes C}$$

$$\text{reduces to}\qquad \dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \qquad \begin{array}{c}\vdots\,\sigma\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta \vdash B} \qquad \begin{array}{c}\vdots\,\omega\\ \Lambda \vdash C\end{array}}{\Gamma, \Delta, \Lambda \vdash B \otimes C}$$

The measure is $\#\rho + \#\sigma + \#\omega + 1$ both before and after the reduction. The case in which the cut formula A comes from the right premise of the $\otimes$R rule is treated analogously.

Case CR$\multimap$R

$$\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\,\omega\\ \Delta, A, C \vdash D\end{array}}{\Delta, A \vdash C \multimap D}}{\Gamma, \Delta \vdash C \multimap D}$$

$$\text{reduces to}\qquad \dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array} \qquad \begin{array}{c}\vdots\,\omega\\ \Delta, A, C \vdash D\end{array}}{\Gamma, \Delta, C \vdash D}}{\Gamma, \Delta \vdash C \multimap D}$$

The measure stays $\#\rho + \#\omega + 1$ during this step.

Case CR$\otimes$L:

$$\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma\vdash A\end{array}\qquad \dfrac{\begin{array}{c}\vdots\,\omega\\ \Delta,A,C,D\vdash B\end{array}}{\Delta,A,C\otimes D\vdash B}}{\Gamma,\Delta,C\otimes D\vdash B}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma\vdash A\end{array}\qquad \begin{array}{c}\vdots\,\omega\\ \Delta,A,C,D\vdash B\end{array}}{\Gamma,\Delta,C,D\vdash B}}{\Gamma,\Delta,C\otimes D\vdash B}$$

The measure stays $\#\rho+\#\omega+1$ during this step.

Case CR$\multimap$L:

$$\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma\vdash A\end{array}\qquad \dfrac{\begin{array}{c}\vdots\,\sigma\\ \Delta\vdash C\end{array}\qquad \begin{array}{c}\vdots\,\omega\\ \Lambda,A,D\vdash B\end{array}}{\Delta,\Lambda,A,C\multimap D\vdash B}}{\Gamma,\Delta,\Lambda,C\multimap D\vdash B}$$

reduces to

$$\dfrac{\begin{array}{c}\vdots\,\sigma\\ \Delta\vdash C\end{array}\qquad \dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma\vdash A\end{array}\qquad \begin{array}{c}\vdots\,\omega\\ \Lambda,A,D\vdash B\end{array}}{\Gamma,\Lambda,D\vdash B}}{\Gamma,\Delta,\Lambda,C\multimap D\vdash B}$$

The measure is $\#\rho+\#\sigma+\#\omega+1$. The case in which the cut formula A comes from the left premise of the $\multimap$L rule is treated analagously.

In other words, the measure stays the same during each commutative reduction step.

Now given a proof π and a Cut in π (not necessarily at the root of π), we can use one of the reduction steps to replace the Cut and so obtain a proof π' . If this reduction step is one of the Axiom Reductions or the Symmetric Reductions, one obtains $\#\pi' < \#\pi$. If the reduction step is one of the Commutative Reductions, one obtains $\#\pi' = \#\pi$.

In order to derive an upper bound on the total number of reduction steps starting with a given proof, we will use $\#\pi$ to estimate the number of consecutive commutative reduction steps that can be performed starting with any proof π. To this end we introduce an auxiliary measure, the *cut-size* $|\pi|$ of a proof π. The cut-size has the same inductive definition as $\#\pi$, except that in the case of a Cut we let:

$$|\pi| = |\pi_1| + |\pi_2| + \#\pi$$

Proposition 2.1 $|\pi| \leq (\#\pi)^2$ *for each RLL proof* π.

Proof : By induction on the complexity of the proof π. In the induction step the interesting case is when π is obtained by a Cut from π_1 and π_2: because the

measure $\#$ is always a positive integer, one has $\#\pi_1 + \#\pi_2 \leq 2(\#\pi_1)(\#\pi_2)$, and thus $|\pi| \leq (\#\pi_1)^2 + (\#\pi_2)^2 + \#\pi_1 + \#\pi_2 \leq (\#\pi_1 + \#\pi_2)^2 = (\#\pi)^2$. $\square$

We now verify that the cut-size decreases in commutative reductions. We continue the notation introduced above in the definitions of commutative reduction steps.

Case CL$\otimes$L: the cut size before the reduction step is $|\pi| = |\rho|+1+|\omega|+\#\pi$. After the reduction step the cut-size is $|\pi'| = |\rho|+|\omega|+(\#\pi'-1)+1 = |\rho|+|\omega|+\#\pi' = |\rho|+|\omega|+\#\pi < |\pi|$.

Case CL$\multimap$L: The cut size before the reduction step is $|\pi| = |\rho|+|\sigma|+1+|\omega|+\#\pi$. After the reduction step the cut-size is $|\pi'| = |\sigma|+|\omega|+(\#\pi'-\#\rho-1)+|\rho|+1 = |\rho|+|\sigma|+|\omega|+\#\pi-\#\rho < |\pi|$.

Case CR$\otimes$R: Before reduction the cut-size is $|\pi| = |\rho|+|\sigma|+|\omega|+1+\#\pi$. After reduction it is $|\pi'| = |\rho|+|\sigma|+(\#\pi'-\#\omega-1)+|\omega|+1 = |\rho|+|\sigma|+|\omega|+\#\pi-\#\rho < |\pi|$.

Case CR$\multimap$R: The cut size before reduction is $|\pi| = |\rho|+|\omega|+1+\#\pi$. On the other hand, after reduction $|\pi'| = |\rho|+|\sigma|+|\omega|+(\#\pi'-1)+1 = |\rho|+|\omega|+\#\pi < |\pi|$.

Case CR$\otimes$L: Before reduction the cut-size is $|\pi| = |\rho|+|\omega|+1+\#\pi$. After reduction it is $|\pi'| = |\rho|+|\omega|+(\#\pi'-1)+1 = |\rho|+|\omega|+\#\pi < |\pi|$.

Case CR$\multimap$L: Here $|\pi| = |\rho|+|\sigma|+|\omega|+1+\#\pi$. After reduction $|\pi'| = |\rho|+|\omega|+(\#\pi'-\#\sigma-1)+|\sigma|+1 = |\rho|+|\sigma|+|\omega|+\#\pi-\#\sigma < |\pi|$.

In summary, when normalizing an RLL proof π, there can be at most $\#\pi$ axiom reduction steps or symmetric reduction steps, and between each of those steps there can be at most $(\#\pi)^2$ consecutive commutative reduction steps. Thus the total number of reduction steps can be at most $(\#\pi)^3$.

Another approach, which allows us to dispense with all commutative reductions, is to consider sequent calculus proofs up to the order of the rules, e.g. to consider the *proof net* representation introduced in [13].

2.2.4 Proof Nets

For the discussion of this approach we assume that the reader is familiar with chapters 1–4 of [13] (or cf. [14] or Section 3 of [11].) First, RLL sequents and proofs may be represented in the one-sided sequent calculus for the multiplicative fragment of linear logic. Indeed, linear implication $A \multimap B$ is definable as $A^\perp \wp B$, i.e. $(A \otimes B^\perp)^\perp$. An RLL sequent $A_1, \ldots, A_n \vdash B$ is translated as the one-sided sequent $\vdash A_1^\perp, \ldots, A_n^\perp, B$. It is readily checked that this translation takes rules of inference to rules of inference (and hence proofs to proofs): the rules $\multimap$R and $\otimes$L are translated as the $\wp$ rule and the rule $\multimap$L is translated as the $\otimes$ rule. Let us also mention the fact that this translation is conservative, i.e., if the translation of an RLL sequent is provable in the multiplicative fragment of linear

logic, then the sequent is provable in RLL.

Second, we use the proof net representation of the one-sided sequent calculus for the multiplicative fragment of linear logic given in [13]. Combining the two interpretations then yields the proof net representation of RLL proofs. The converse follows from the conservativity of the first translation mentioned above, i.e. the Sequentialization Theorem in [13] also holds for RLL.

We identify RLL proofs with the same proof net representation. In this way the only required reduction steps are Axiom Reductions and Symmetric Reductions. Any sequence of these reduction steps starting with an RLL proof π must terminate in at most $\#\pi$ steps in a cut-free proof net representing a cut-free RLL proof.

2.2.5 Discussion of second order RLL

All of the properties of RLL described above, except conservativity of the translation into one-sided sequent calculus, remain true if we add impredicative second order universal quantification over propositions. The additional rules are

$$\forall\mathrm{L}\quad \frac{\Gamma, A[\alpha := T] \vdash B}{\Gamma, (\forall\alpha)A \vdash B} \qquad\qquad \forall\mathrm{R}\quad \frac{\Gamma \vdash A}{\Gamma \vdash (\forall\alpha)A}$$

where $A[\alpha := T]$ is the result of substituting a second order abtraction term T for all free occurrences of the propositional variable α in A, and where in the rule $\forall$R the propositional variable α does not occur free in the formulas in Γ. We often omit parentheses around quantifiers.

In extending the measure $\#\pi$ and the cut-size $|\pi|$ the rules $\forall$L and $\forall$R are treated simply as unary rules. Observe that the measure and the cut-size do not increase under substitution of second order abstraction terms. In normalization, the additional symmetric reduction step is:

Case S$\forall$:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\rho\\ \Gamma \vdash A\end{array}}{\Gamma \vdash \forall\alpha A} \qquad \dfrac{\begin{array}{c}\vdots\,\omega\\ \Delta, A[\alpha := T] \vdash B\end{array}}{\Delta, \forall\alpha A \vdash B}}{\Gamma, \Delta \vdash B}$$

reduces to

$$\dfrac{\begin{array}{c}\vdots\,\rho[\alpha:=T]\\ \Gamma \vdash A[\alpha := T]\end{array} \qquad \begin{array}{c}\vdots\,\omega\\ \Delta, A[\alpha := T] \vdash B\end{array}}{\Gamma, \Delta \vdash B}$$

In this reduction of the proof π to π', the measure decreases from $\#\pi = \#\rho + \#\omega + 2$ to $\#\pi' = \#\rho + \#\omega < \#\pi$.

Let us also check the additional commutative reductions that involve the quantifier rules.

Case CL$\forall$L:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma, C[\alpha := T] \vdash A\end{array}}{\Gamma, \forall\alpha C \vdash A} \qquad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, \forall\alpha C \vdash B}$$

reduces to
$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma, C[\alpha := T] \vdash A\end{array} \qquad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, C[\alpha := T] \vdash B}}{\Gamma, \Delta, \forall\alpha C \vdash B}$$

The measure is $\#\pi = \#\pi' = \#\rho + \#\omega + 1$ both before and after this reduction step. The cut-size before reduction is $|\pi| = |\rho| + 1 + |\omega| + \#\pi$. After reduction, it is $|\pi'| = |\rho| + |\omega| + (\#\pi' - 1) + 1 = |\rho| + |\omega| + \#\pi$, less than before reduction.

Case CR∀R :

$$\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Delta, A \vdash \forall\alpha B}}{\Gamma, \Delta, \forall\alpha B}$$

reduces to
$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta \vdash B}}{\Gamma, \Delta \vdash \forall\alpha B}$$

We may assume the α does not occur free in Γ, by renaming the bound variables in B if necessary (which does not change the measure or the cut-size). Note that $\#\pi = \#\pi' = \#\rho + \#\omega + 1$. Also $|\pi| = |\rho| + |\omega| + 1 + \#\pi$. $|\pi'| = |\rho| + |\omega| + (\#\pi' - 1) + 1 = |\rho| + |\omega| + \#\pi < |\pi|$.

Case CR∀L :

$$\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, A, C[\alpha := T] \vdash B\end{array}}{\Delta, A, \forall\alpha C \vdash B}}{\Gamma, \Delta, \forall\alpha C \vdash B}$$

reduces to
$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A, C[\alpha := T] \vdash B\end{array}}{\Gamma, \Delta, C[\alpha := T] \vdash B}}{\Gamma, \Delta, \forall\alpha C \vdash B}$$

Note that $\#\pi = \#\pi' = \#\rho + \#\omega + 1$. Also $|\pi| = |\rho| + |\omega| + 1 + \#\pi$. $|\pi'| = |\rho| + |\omega| + (\#\pi' - 1) + 1 = |\rho| + |\omega| + \#\pi < |\pi|$.

Thus, even for second order RLL, there can be at most $(\#\pi)^3$ sequent reduction steps in the normalization of a proof π. This upper bound can again be lowered to $\#\pi$, this time by using the proof net representation given in [15].

The results of Section 2.2 may be summarized as follows:

Theorem 2.2 *Let π be a proof in RLL or in second order RLL. Any sequence of reduction steps on π must terminate in a cut-free proof in at most $(\#\pi)^3$ sequent calculus reduction steps. This cut-free proof is unique up to order of the instances of the rules. It has at most $\#\pi$ instances of the rules, including the axioms. Furthermore, any sequence of proof net reduction steps on the proof net representing π must terminate in at most $\#\pi$ reduction steps.*

Remark : In fact, proof net reductions are completely asynchronous; they do not have to be performed sequentially [13, 15].

The Sequentialization Theorem [15] applies to the discussion of second order RLL only with respect to one-sided sequent calculus. There is no claim of conservativity of one-sided sequent calculus over the two-sided style for the second order RLL considered here. However, the fact still remains that for these two presentations of the sequent calculus, there exists a natural correspondence of normal forms of certain types, for example

$$\forall\alpha(\underbrace{(\alpha \multimap \alpha) \otimes \ldots \otimes (\alpha \multimap \alpha)}_{n} \multimap (\alpha \multimap \alpha)) \qquad .$$

Hence for our purposes here, we can still freely use proof net interpretations for the two-sided sequent calculi.

2.2.6 Adding Unrestricted Weakening

The phenomenon of shrinking proofs observed above remains valid even if one adds the structural rule of Weakening:

$$\frac{\Gamma \vdash A}{\Gamma, C \vdash A}$$

or equivalently, if one reformulates the axioms as:

$$\Gamma, A \vdash A$$

In this latter system the measure of an axiom is still 1. It is readily checked that if π is a proof of a sequent $\Delta \vdash A$, then for any finite multiset of formulas Γ, there is a proof ρ of the sequent $\Gamma, \Delta \vdash A$, where $\#\rho \leq \#\pi$, $|\rho| \leq |\pi|$, and ρ has the same underlying rooted tree as π. This fact allows us to transfer the measure of the computations given above to the system with axioms of the form $\Gamma, A \vdash A$.

2.3 Linear Logic

While RLL and the related systems discussed above enjoy fast normalization, these systems have little expressive power. The problem of adding expressive power to RLL may be resolved by adding a new connective "!" for storage. $!A$ means A can be reused *ad nauseam.* The system LL of linear logic is obtained from RLL by adding rules for ! :

Storage $\dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}$ **Weakening** $\dfrac{\Gamma \vdash B}{\Gamma, !A \vdash B}$

Contraction $\dfrac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}$ **Dereliction** $\dfrac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}$

There is now a tremendous increase of expressive power: we can represent first order function types by $A \Rightarrow B := !A \multimap B$ [13] . It is folklore on finite types that there can be no realistic time bounds on computations. Specifically, take a ground type *0* and define higher types $n{+}1 := n \Rightarrow n$. Now define the analog of Church numerals **p** of type $n{+}2$, Y^p_{n+2} as $\lambda f.f^p$, where the variable f is of type $n{+}1$. One easily verifies that modulo β-conversion, $Y^p_{n+2}(f) \circ Y^q_{n+2}(f) = Y^{p+q}_{n+2}(f)$, $Y^p_{n+2} \circ Y^q_{n+2} = Y^{pq}_{n+2}$, and $Y^p_{n+3}(Y^q_{n+2}) = Y^{q^p}_{n+2}$. Therefore $Y^2_{n+2} Y^2_{n+1} \cdots Y^2_2 = Y^c_2$, where $c = 2^{2^{2^{\cdots}}}$ is a tower of 2's.

Furthermore, adding full impredicative second order quantification ($\forall$) yields a system of LL^2 as strong as system $\mathcal{F}$ (= second-order polymorphic lambda calculus). In particular, every provably total recursive function of second order arithmetic is representable in the system. In other words, in order to produce a total numerical function which is not representable in LL^2 , one has to go beyond most current mathematics.

2.4 Toward Bounded Linear Logic

We seek a system intermediate between second order RLL and full second order linear logic, which would enjoy feasible normalization and would yet be powerful enough to express all feasible functions. To this end we consider *bounded reuse* , roughly $!_x A$ with the intuitive meaning that datum A may be reused less than x times. Let us first present just a simplified version of the desired intermediate system and the basic intuition behind it; the precise consideration will be taken up in sections 3 and 4 below. If Γ is $A_1, \ldots, A_n$ we write $!_{\vec{y}}\Gamma$ for $!_{y_1} A_1, \ldots, !_{y_n} A_n$.

The rules for storage naturally induce polynomials:

Storage $\dfrac{!_{\vec{y}}\Gamma \vdash A}{!_{x\vec{y}}\Gamma \vdash !_x A}$ **Weakening** $\dfrac{\Gamma \vdash B}{\Gamma, !_0 A \vdash B}$

Contraction $\dfrac{\Gamma, !_x A, !_y A \vdash B}{\Gamma, !_{x+y} A \vdash B}$ **Dereliction** $\dfrac{\Gamma, A \vdash B}{\Gamma, !_1 A \vdash B}$

We may interpret these rules in second order RLL, by translating $!_x A$ as $1 \underbrace{\otimes A \otimes \cdots \otimes A}_{x\ \otimes\text{'s}}$, where there are exactly x tensor signs and where 1 may be thought of as $\forall\alpha(\alpha \multimap \alpha)$. This translation is logically sound only if we add to RLL the unrestricted weakening rules (see section 2.2.6). A consequence of the latter is that from $(n+1)$-ary tensorization one can obtain the n-ary one. The addition of the unrestricted weakening rules to RLL is of course not problematic. As observed at the end of section 2.2.6, proofs still shrink under normalization. The weight(measure) associated to a proof is a polynomial, the key cases of Storage and Contraction being:

$$\frac{!_{\vec{y}}\Gamma \vdash A \qquad p}{!_{x\vec{y}}\Gamma \vdash !_x A \qquad (p+1)x+2nx+n+1}$$

where n is the number of formulas in Γ, and :

$$\frac{\Gamma, !_x A, !_y A \vdash B \qquad p}{\Gamma, !_{x+y} A \vdash B \qquad p+2}$$

In the cases of Weakening and Dereliction one adds 1. The axioms and cut are treated as in RLL.

Discussion : These formulas basically follow from the translation into RLL mentioned above, but they do involve some overestimates for the sake of uniformity in the cases $x = 0, y = 0, \text{or}\, n = 0$. Another advantage over the weights assigned to the Storage and Contraction rules in [21] ($(p+1)x + n$ and $p+1$, respectively) is that the weight of a proof is always positive and hence it easily fits into the pattern discussed in section 2.2. As in [21], however, there is still a problem in the reduction steps that apply when the cut formula is $!_x A$ and the left premise of a Cut rule is a consequence of a Storage rule with Γ nonempty, i.e. $n > 0$. The answer, as in [21], is to consider modified normalization in which such reduction steps are prohibited (see Section 4). Here we present a simplified version of two crucial cases of the modified normalization procedure. Observe that the weight strictly decreases.

Reduction step: Storage vs. Contraction

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \rho \\ \vdash A\end{matrix}}{\vdash !_{u+v}A} \qquad \dfrac{\begin{matrix}\vdots\ \omega \\ \Delta, !_u A, !_v A \vdash B\end{matrix}}{\Delta, !_{u+v}A \vdash B}}{\Delta \vdash B}$$

$$\text{reduces to}\quad \dfrac{\dfrac{\overset{\vdots\,\rho}{\vdash A}}{\vdash !_u A} \qquad \dfrac{\dfrac{\overset{\vdots\,\rho}{\vdash A}}{\vdash !_v A} \qquad \Delta, !_u A, !_v A \vdash B}{\Delta, !_u A \vdash B}}{\Delta \vdash B}$$

Let R and Q be the weights of the proofs ρ and ω, respectively. Let $t = u + v$. The weight of the entire proof before the reduction step is (R+1)t + 1 + Q + 2 = (R+1)t + Q + 3. After the reduction step, the weight is (R+1)u + 1 + (R+1)v + 1 + Q = (R+1)$(u + v)$ + Q + 2 = (R+1)t + Q + 2. There is also a similar reduction in which the cuts in the reduct are done in a different order: the same weights arise in this case.

Reduction Step: Storage vs. Storage

$$\dfrac{\dfrac{\overset{\vdots\,\rho}{\vdash A}}{\vdash !_{vu} A} \qquad \dfrac{\overset{\vdots\,\omega}{!_{\vec{r}}\Delta, !_u A \vdash B}}{!_{v\vec{r}}\Delta, !_{vu} A \vdash !_v B}}{!_{v\vec{r}}\Delta \vdash !_v B}$$

$$\text{reduces to}\quad \dfrac{\dfrac{\dfrac{\overset{\vdots\,\rho}{\vdash A}}{\vdash !_u A} \qquad \overset{\vdots\,\omega}{!_{\vec{r}}\Delta, !_u A \vdash B}}{!_{\vec{r}}\Delta \vdash B}}{!_{v\vec{r}}\Delta \vdash !_v B}$$

Again let R and Q be the weights of the proofs ρ and ω, respectively. Let $t = vu$. The weight of the entire proof before reduction is (R+1)t + 1 + (Q+1)v +(2n+2)v + n + 2 = (R+1)t + (Q+1)v + (2n+2)v + n + 3. After reduction it is [(R+1)u + Q + 2]v + 1 + 2nv + n = (R+1)uv + (Q+1)v + v + 1 + 2nv + n = (R+1)t + (Q+1)v + $(2n + 1)v + n + 1$.

3 The Syntax of Bounded Linear Logic (BLL)

3.1 Resource Polynomials

Let $\binom{x}{n}$ be the usual binomial coefficient. In particular $\binom{x}{0} = 1$. A *monomial* is any (finite) product of binomial coefficients, $\prod_{i=1}^{p} \binom{x_i}{n_i}$, where the variables x_i are distinct and n_i are non-negative integer constants.

A *resource polynomial* is any finite sum of monomials, e.g. $0, 1, y$, $x + (z(z-1)/2)$, etc.

Resource polynomials are closed under sum, product, and composition. Such

polynomials are exactly the *finite dilators* of proof theory [16] and are closely related to *combinatorial functors* [9].

Given resource polynomials p, q write $p \sqsubseteq q$ to denote that $q - p$ is a resource polynomial. If $p \sqsubseteq p'$ and $q \sqsubseteq q'$, then their composites satisfy $q \circ p \sqsubseteq q' \circ p'$.

3.2 Formulae of BLL

Formulae (= types) : atomic formulae have the form $\alpha(\vec{p})$; here α is a second-order variable of given finite positive arity and $\vec{p}$ here denotes an appropriate non-empty list of resource polynomials.

Formulae are closed under the following operations:

(i) $A \otimes B$ and $A \multimap B$ from RLL,

(ii) $(\forall\alpha)A$ (*second order universal quantification*),

(iii) $!_{x<p}A$ (*bounded exclamation mark* with p a resource polynomial not containing x) .

Positive and negative occurrences of resource terms in formulae are defined by induction as usual; in $!_{x<p}A$, p occurs negatively and x is a bound variable not occurring in p. Let the free resource variables $x_1, ..., x_n$ occur only positively in B. Then $\lambda x_1, ..., x_n.B$ is a (second order) abstraction term, say T. $A[\alpha := T]$ denotes the result of substituting T for α in A , i.e. of replacing the atoms $\alpha(p_1, ..., p_n)$ in A by $B[p_1, ..., p_n]$. Given types A and A' , write $A \sqsubseteq A'$ if A and A' only differ in their choice of resource polynomials, and

(i) for any positive occurence of resource polynomial p in A, the homologous p' in A' is such that $p \sqsubseteq p'$.

(ii) for any negative occurence of resource polynomial p in A, the homologous p' in A' is such that $p' \sqsubseteq p$.

If Γ and Γ' are finite multisets of formulae, $\Gamma \sqsubseteq \Gamma'$ iff it is true componentwise.

3.3 BLL Sequents

Sequents have the form $\Gamma \vdash B$, where Γ is a finite (possibly empty) multiset of formulae. The formulae in Γ are considered indexed but not ordered. (*Notation*: parameters p, q, v, w range over resource polynomials. $A[x := p]$ denotes the substitution of p for all free occurences of resource variable x in formula A. From now on we write $!_{y<x}A$ instead of the formula $!_x A$ (cf. 2.4) .) We may intuitively think of $!_{y<p}A$ as $1 \otimes A[y := 0] \otimes \cdots \otimes A[y := p - 1]$

Axiom (Waste of Resources) $A \vdash A'$, where $A \sqsubseteq A'$ (Special case: $A \vdash A$).

Cut
$$\frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$$

$$\otimes\mathrm{L} \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \otimes\mathrm{R} \quad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$$

$$\multimap\mathrm{L} \quad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma\ , \Delta, A \multimap B \vdash C} \qquad \multimap\mathrm{R} \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

$$\forall\mathrm{L} \quad \frac{\Gamma, A[\alpha := T] \vdash B}{\Gamma, (\forall\alpha)A \vdash B} \qquad \forall\mathrm{R} \quad \frac{\Gamma \vdash A}{\Gamma \vdash (\forall\alpha)A}$$

(provided α is not free in Γ)

(!W) *Weakening*

$$\frac{\Gamma \vdash B}{\Gamma, !_{x<w}A \vdash B}$$

(!D) *Dereliction*

$$\frac{\Gamma, A[x := 0] \vdash B}{\Gamma, !_{x<1+w}A \vdash B}$$

(!C) *Contraction*

$$\frac{\Gamma, !_{x<p}A, !_{y<q}A[x := p + y] \vdash B}{\Gamma, !_{x<p+q+w}A \vdash B}$$

where $p + y$ is free for x in A.

(S!) *Storage*

$$\frac{!_{z<q_1(x)}A_1[y := v_1(x) + z], \ldots, !_{z<q_n(x)}A_n[y := v_n(x) + z] \vdash B}{!_{y<v_1(p)+w_1}A_1, \ldots, !_{y<v_n(p)+w_n}A_n \vdash !_{x<p}B,}$$

where $v_i(x) + z$ is free for y in A_i, where $v_i(x) = \sum_{z<x} q_i(z)$ and where all formulae to the left of the $\vdash$ have the indicated form.

A *proof* of a sequent $\Gamma \vdash A$ in the BLL sequent calculus is a finite labeled rooted tree in which the nodes are labeled by BLL sequents so that the leaves are labeled by instances of the axiom, the root is labeled by $\Gamma \vdash A$, and the label of each node is obtained from the labels of its immediate predecessor(s) by an instance of a BLL rule.

Remark: The rules of BLL are written in such a way that given any proof ρ of a sequent $\Gamma \vdash A$ and given any $\Gamma' \sqsubseteq \Gamma$ and $A \sqsubseteq A'$ then a simple change of resource parameters will yield a proof ρ' of $\Gamma' \vdash A'$ *without altering the structure of the proof*.

3.3.1 Lambda term assignment for BLL proofs

We remind the reader that in Gentzen sequent calculi, as well as in natural deduction calculi, proofs can be represented by lambda terms, cf.[20, Chapter 5]. In particular, an axiom $A \vdash A'$ is represented by $a : A \ \rhd\ a : A'$, the logical rule $\multimap$R is represented by λ-abstraction or currying:

$$\frac{\vec{c} : \Gamma, x : A \ \rhd\ t : B}{\vec{c} : \Gamma \ \rhd\ \lambda x.t : A \multimap B}$$

while the logical rule $\multimap$L is represented by an application of a functional variable, here denoted by e:

$$\frac{\vec{c}:\Gamma \ \rhd t:A \qquad \vec{d}:\Delta, b:B \ \rhd u:C}{\vec{c}:\Gamma\ ,\ \vec{d}:\Delta, e:A \multimap B \ \rhd u[b:=e(t)]:C}$$

The cut rule may be represented by substitution:

$$\frac{\vec{c}:\Gamma \ \rhd t:A \qquad \vec{d}:\Delta, a:A \ \rhd u:B}{\vec{c}:\Gamma\ ,\ \vec{d}:\Delta \ \rhd u[a:=t]:B}$$

which is certainly denotationally consistent. (However, from a more dynamic viewpoint it would have been more appropriate to define:

$$\frac{\vec{c}:\Gamma \ \rhd t:A \qquad \vec{d}:\Delta, a:A \ \rhd u:B}{\vec{c}:\Gamma\ ,\ \vec{d}:\Delta \ \rhd \mathtt{let}\, a = t\, \mathtt{in}\, u:B}$$

where the explicit substitution is only indicated: it is actually carried out by the reduction steps in cut-elimination.)

This assignment may be extended to the quantifier rules trivially, i.e. the quantifier rules have no effect on lambda terms. Since tensor $A \otimes B$ is definable as $\forall\alpha((A \multimap B \multimap \alpha) \multimap \alpha)$, the assignment given so far yields a lambda term assignment for the tensor rules. Finally, the lambda term assignment may be extended to BLL trivially, i.e. the storage rules have no effect on the lambda terms.

This lambda term assignment is rather crude (for example, among its shortcomings are that storage and quantifier rules have no effect). Its sole purpose here is to serve as a framework for relating our definition of representability of functions in BLL (see section 5) to the usual notion of representability of functions in lambda calculus. The question of a good syntax for term assignments to BLL proofs is open (but see [1] for the case of LL).

3.4 Proof nets for BLL

We now extend to BLL the proof net representation of RLL proofs mentioned in section 2.2.4. Proof structures (with boxes) , and in particular proof nets as discussed below, are defined almost exactly as in Chapter 2 of [13], with the quantifiers treated as in [15]. The exceptions are that our axioms must reflect waste of resources, that we do not consider additive connectives at all, that we consider bounded operators $!_{x<p}$ and $?_{x<p}$ instead of ! and ?, resp., and that the weakening rule is treated as a link, not as a box. An alternative approach to proof nets , given in [10], Chap.3-6, and [11], Section 3, provides an amenable framework for our treatment of the weakening rule.

A proof of a BLL sequent $A_1, \ldots, A_n \vdash B$ will be represented by a proof net with conclusions $A_1^\perp, \ldots, A_n^\perp, B$. For a BLL formula C, $C^\perp$ is a formula defined as follows (see [13]). First, translate $A \multimap B$ as $A^\perp \wp B$. Second, if C is an atomic BLL formula, let $C^\perp$ be a new formula in an expanded language and let $(C^\perp)^\perp$ be C. $(A \otimes B)^\perp$ is $A^\perp \wp B^\perp$, $(A \wp B)^\perp$ is $A^\perp \otimes B^\perp$, $(\forall\alpha A)^\perp$ is $\exists\alpha A^\perp$, $(!_{x<p}A)^\perp$ is $?_{x<p}A^\perp$, $(?_{x<p}A)^\perp$ is $!_{x<p}A^\perp$. Then let $(\lambda x_1, \ldots, x_n.B)^\perp$ be $\lambda x_1, \ldots, x_n.B^\perp$. (The reader will note that $A^{\perp\perp}$ is A.) If a resource term p occurs positively in

A, it occurs negatively in $A^{\perp}$ and vice versa. The relation $A \sqsubseteq A'$ is extended accordingly.

Proof structures are non-empty labeled graphs [2] whose labels consist of (occurrences of) formulas, connected by various kinds of *links* or *boxes* in which there are certain distinguished multisets of *premises* and *conclusions* , as defined below. Links will correspond to the Axiom and Rules of Inference of BLL and are defined as follows (where $A, B, \ldots$ denote formula occurrences) :

- (Axiom Link) $\overbracket{A^{\perp} \qquad A'}$, where $A \sqsubseteq A'$. The conclusions of this link are $A^{\perp}$ and A'. There are no premises. (This link represents the Axiom).

  Note: since conclusions form a multiset, this link is considered to be the same as the link given by the figure

  $\overbracket{A' \qquad A^{\perp}}$, where $A \sqsubseteq A'$.

- (Cut Link)

$$\frac{A \qquad A^{\perp}}{}$$

  is a link, with no conclusions and premises A and $A^{\perp}$. (This link represents the Cut Rule.) Note that since we identify $A^{\perp\perp}$ with A, this Cut Link is considered to be the same as the one where we interchange (the positions of) A and $A^{\perp}$.

- (Tensor Link)

$$\frac{A \quad B}{A \otimes B}$$

  is a link whose conclusion is $A \otimes B$ and whose premises are A and B. (This link represents the rules $\otimes$R and $\multimap$L.) Note that, unlike the Axiom and Cut Links, the tensor link is not symmetric in A and B.

- ($\wp$ Link)

$$\frac{A \quad B}{A \wp B}$$

  is a link whose conclusion is $A \wp B$ and whose premises are A and B. (This link represents the rules $\otimes$L and $\multimap$R.) As in the case of the Tensor Link, the $\wp$ Link is not symmetric in A and B.

- ($\forall$ Link)

$$\frac{A}{\forall \alpha A}$$

[2] not necessarily planar

is a link whose conclusion is $\forall\alpha A$ and whose premise is A. Here α is the eigenvariable of the $\forall$ link and it is forbidden to use it as the eigenvariable of any other $\forall$ link, see [15]. (This link represents the rule $\forall$R.)

- ($\exists$ Link)

$$\frac{A[\alpha := T]}{\exists\alpha A}$$

is a link whose conclusion is $\exists\alpha A$ and whose premise is $A[\alpha := T]$. (This link represents the rule $\forall$L.)

The links for the operators $?_{x<p}$ are as follows:

- (Weakening Link)

$$\frac{}{?_{x<w}A}$$

is a link whose conclusion is $?_{x<w}A$ and which has no premises. (This link represents the Weakening rule !W).

- (Dereliction Link)

$$\frac{A[x := 0]}{?_{x<1+w}A}$$

is a link whose conclusion is $?_{x<1+w}A$ and whose premise is $A[x := 0]$. (This link represents the Dereliction rule !D .)

- (Contraction Link) If $p + y$ is free for x in A, then:

$$\frac{?_{x<p}A \quad ?_{y<q}A[x := p + y]}{?_{x<p+q+w}A}$$

is a link whose conclusion is $?_{x<p+q+w}A$ and whose premises are $?_{x<p}A$ and $?_{y<q}A[x := p + y]$ (This link represents the Contraction Rule !C .)

Finally, proof structures may contain *boxes* [13], which could be considered as links of a special kind, defined as follows:

- ($!_{x<p}$ Boxes)

$$\boxed{\quad ?_{y_1<v_1(p)+w_1}A_1 \quad \ldots \quad ?_{y_n<v_n(p)+w_n}A_n \qquad !_{x<p}B \quad}$$

is a *proof box* (or simply: *box*) whose *main door* is $!_{x<p}B$ and whose *auxiliary doors* are $?_{y_1<v_1(p)+w_1}A_1, \ldots, ?_{y_n<v_n(p)+w_n}A_n$. This box has as conclusions all of the doors, both main and auxiliary. (This box will represent the Storage rule S! .) We allow boxes to contain other proof structures.

Proof structures are built from links and boxes as described above, subject to the following requirements:

- Every occurrence of a formula in the proof structure is the conclusion of exactly one link or box, and a premise of at most one link.
- Whenever a box contains a formula occurring in a link (either as a premise or a conclusion) then this box must contain all other formulas occurring in the link.
- Given any two boxes in a proof structure, either (i) their respective contents and conclusions must be completely disjoint, or else (ii) one box must be properly contained in the other.

The *conclusions of a proof structure* are the conclusions of its links (and of its boxes) that do not appear as premises of other links.

Remark: Since a proof structure within a box may contain other boxes, etc., we have really given a definition of proof structure by induction on the depth of nested boxes.

Following [13], one may assign proof structures to (sequential) Gentzen proofs in BLL. Among all possible proof structures, one can distinguish those which so arise (i.e. from proofs in linear sequent calculus); these are called *proof nets* ([13]). There is a mathematical characterization (so-called *correctness criterion*) picking out those general proof structures which are actually proof nets. This criterion can be phrased in terms of trips, as in [13] but for our purposes, the treatment by means of acyclic connected graphs [11, 10] is somewhat more amenable. In particular, the correctness criterion for a weakening link may be stated as the *existence* of a "pointer" to another link in the proof structure. [3] The correctness criterion for $\forall$-links is stated by means of arbitrary pointers to the links in the proof structure that contain a free occurrence of the eigenvariable [15]. In either case, these pointers are not allowed to enter or exit any boxes. Regarding boxes

[3] This is *not* the way weakening links are treated in [10]. Our precise correctness criterion for a proof structure with weakening links is the *simultaneous existence* for each weakening link of a "virtual premise", i.e., the choice of another (occurrence of a) formula in the proof structure, in such a way that this bigger graph enjoys the correctness criterion known in the absence of weakening links, namely that each subgraph obtained in a certain way is acyclic and connected [11, 15]. The new criterion is easily shown to be correct, but it is not very satisfactory, because its preservation under cut-elimination is not conceptually immediate. Indeed, during cut-elimination some virtual premise may be destroyed. In such a case, we must show that we could have indeed chosen a virtual premise that has not been destroyed. This offers no difficulty, but one has to go through a big number of cases.

in proof nets, they may themselves only contain arbitrary proof nets, rather than arbitrary structures. The correctness criterion for boxes can be found in Chapter 6 of [10] (by means of acyclic connected graphs) or in [13] by means of trips. Finally, in addition to correctness criteria in proof nets, we restrict box formation so that one considers only boxes of the form

$$\left[\begin{array}{ccccc} & \sigma & & & \\ \vdots & & \vdots & \vdots & \vdots \\ ?_{z<q_1(x)}A_1^{\perp}[y_1 := v_1(x)+z] & \cdots & ?_{z<q_j(x)}A_j^{\perp}[y_j := v_j(x)+z] & \cdots & B \\ ?_{y_1<v_1(p)+w_1}A_1^{\perp} & \cdots & ?_{y_j<v_j(p)+w_j}A_j^{\perp} & \cdots & !_{x<p}B \end{array}\right]$$

where σ is a proof *net* whose conclusions are indicated, where $1 \leq j \leq n$, and $0 \leq n$, and the v_j satisfy the same conditions as in the Storage rule.

As in Section 3.3 above, we observe that if $A_i \sqsubseteq A'_i, {}_{1\leq i\leq n}$ and if the A_i's are the conclusions of a proof net ν, then there is another proof net ν' with the same graphical structure whose conclusions are the A'_i's. Finally, observe that the links and boxes presented above can also be thought of as inductive clauses in an inductive definition of a *proof structure with given conclusions* starting from the Axiom Links (see also [10] for a complete treatment).

3.5 The weight of a BLL proof structure

We assign a polynomial $\|\pi\|$ to every BLL proof structure π (and hence to every BLL proof). The polynomial $\|\pi\|$ will be called the *weight* of π.

The weight of every link except Contraction is 1 (this includes the Axiom link). The weight of Contraction is 2. The weight of a box whose content is proof structure σ , with n auxiliary doors and whose resource polynomial at the main door is p, is $\sum_{x<p} (\|\sigma\|(x)+1) + 2np + n + 1$. (If a box has no contents, we arbitrarily set $\|\sigma\| = 0$. This situation never arises in the case of proof nets .) Finally, the weight of a proof structure is defined to be the sum of the weights of its links and boxes.

The following three propositions are readily checked. We use the pointwise order of polynomials with respect to non-negative integer arguments.

Proposition 3.1 *Let $A_1, \ldots, A_n$ be the conclusions of a proof net ν and let $A_i \sqsubseteq A'_i$, $1 \leq i \leq n$. Then a simple change of resource parameters in ν yields a proof net ν' whose conclusions are $A'_1, \ldots, A'_n$, such that $\|\nu'\| \leq \|\nu\|$.*

Proposition 3.2 *Let ν be a proof net and let p be a resource polynomial free for substitution for the free resource variable x in ν. Let $\nu[x := p]$ be the result of*

substituting p for all free occurrences of resource variable x in ν. Then $\nu[x := p]$ is a proof net and $\|\nu[x := p]\| \leq \|\nu\|[x := p]$.

Proposition 3.3 *Let ν be a proof net and let T be a second order abstraction term $\lambda x_1 \ldots \lambda x_n.B$, where all free occurrences of the resource variables $x_1, \ldots, x_n$ in B are positive. Let $\nu[\alpha := T]$ be the result of substituting T for all free occurrences of a second order variable α in the proof net ν. Then ν is a proof net and $\|\nu[\alpha := T]\| \leq \|\nu\|$.*

The analogous properties hold for the BLL sequent calculus.

4 Normalization and Proof Nets

4.1 Normalization in BLL

We shall refer only to proof nets. The analogous discussion for the BLL sequent calculus is indicated in Appendix A. Here we define the proof net reduction steps and simultaneously show that the weight of a proof net decreases. The weight analysis can be extended to the BLL sequent calculus reductions by using cut-size, analogously to section 2.2 above.

Definition 4.1 In BLL proof nets, an instance of the cut link is *boxed* when it is contained in a proof box.

Our normalization procedure will eliminate only non-boxed cuts. We cannot eliminate boxed cuts because the polynomial p at the main door of a box may be 0 , in which case the weight is no longer strictly monotone under reduction.

Definition 4.2 In BLL proof nets, an instance of the cut link is *irreducible* if it is boxed or if one of its premises is a box with at least one auxiliary door, where the cut formula is at the main door, and the other premise is a conclusion of a Weakening, Dereliction, or Contraction link, or a box.

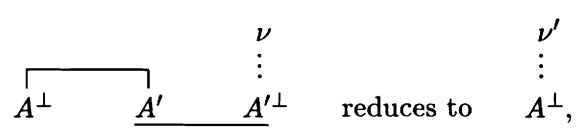
Definition 4.3 A BLL proof net is *irreducible* if it contains only irreducible cuts (if any).

It is understood that the reduction steps given below do not apply to irreducible cuts.

Axiom Reductions

AL

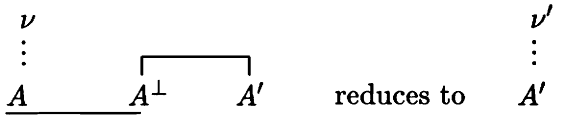
Let $A \sqsubseteq A'$. Then:



see Proposition 3.1. The weight decreases from $\|\nu\| + 1$ to $\|\nu'\| \leq \|\nu\|$.

AR

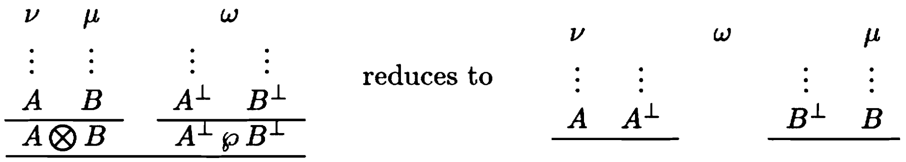
Again, $A \sqsubseteq A'$. Then:



see Proposition 3.1 . The weight again decreases from $\|\nu\| + 1$ to $\|\nu'\| \leq \|\nu\|$.

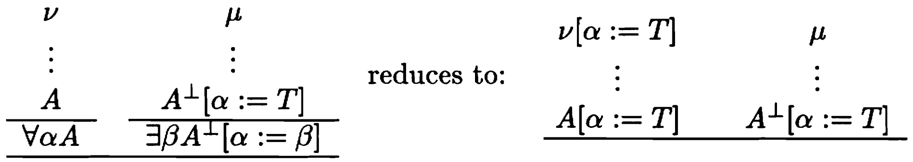
Symmetric Reductions

S$\otimes\wp$



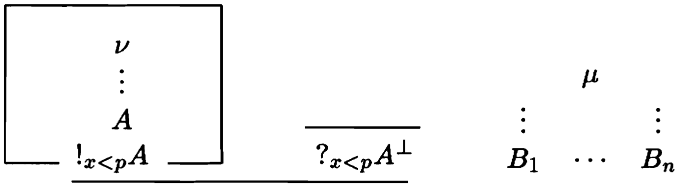
where $A^\perp$ and $B^\perp$ are among the conclusions of the given proof net ω. The weight decreases from $\|\nu\| + \|\mu\| + \|\omega\| + 2$ to $\|\nu\| + \|\mu\| + \|\omega\|$.

S$\forall$



the weight decreases from $\|\nu\| + \|\mu\| + 2$ to $\|\nu[\alpha := T]\| + \|\mu\| \leq \|\nu\| + \|\mu\|$, see Proposition 3.3 .

SSW



$$\text{reduces to:}\quad \begin{array}{ccc} & \mu & \\ \vdots & & \vdots \\ B_1 & \cdots & B_n \end{array}$$

The weight decreases from $\sum_{x<p} (\|\nu\|(x)+1) + \|\mu\| + 2$ to $\|\mu\|$.

SSD

$$\dfrac{\boxed{\begin{array}{c}\nu\\ \vdots\\ A\end{array}}}{!_{x<1+w}A} \qquad \dfrac{\begin{array}{c}\mu\\ \vdots\\ A^{\perp}[x:=0]\end{array}}{?_{x<1+w}A^{\perp}}$$

$$\text{reduces to}\qquad \dfrac{\begin{array}{c}\nu[x:=0]\\ \vdots\\ A[x:=0]\end{array} \qquad \begin{array}{c}\mu\\ \vdots\\ A^{\perp}[x:=0]\end{array}}{}$$

The weight decreases from $\sum_{x<1+w} (\|\nu\|(x)+1) + \|\mu\| + 2$ to $\|\nu[x:=0]\| + \|\mu\|$.

SSC

$$\dfrac{\boxed{\begin{array}{c}\nu\\ \vdots\\ A\end{array}}}{!_{x<p+q+w}A} \qquad \dfrac{\begin{array}{c}\vdots\\ ?_{x<p}A^{\perp}\end{array} \qquad \dfrac{\begin{array}{c}\mu\\ \vdots\\ ?_{y<q}A^{\perp}[x:=p+y]\end{array}}{?_{x<p+q+w}A^{\perp}}}{}$$

reduces to:

$$\dfrac{\boxed{\begin{array}{c}\nu\\ \vdots\\ A\end{array}}}{!_{x<p}A} \qquad \dfrac{\begin{array}{c}\vdots\\ ?_{x<p}A^{\perp}\end{array} \qquad \dfrac{\begin{array}{c}\mu'\\ \vdots\\ ?_{y<q+w}A^{\perp}[x:=p+y]\end{array} \qquad \boxed{\begin{array}{c}\nu[x:=p+y]\\ \vdots\\ A[x:=p+y]\end{array}}}{!_{y<q+w}A[x:=p+y]}}{}$$

The weight decreases from $\sum_{x<p+q+w} (\|\nu\|(x)+1) + \|\mu\| + 3$ to

$$
\begin{aligned}
&\sum_{x<p} (\|\nu\|(x)+1) + \sum_{y<q+w} (\|\nu\|(p+y)+1) + \|\mu'\| + 2 \\
&\quad \leq \sum_{x<p+q+w} (\|\nu\|(x)+1) + \|\mu'\| + 2 \\
&\quad \leq \sum_{x<p+q+w} (\|\nu\|(x)+1) + \|\mu\| + 2
\end{aligned}
$$

(See Proposition 3.1 for the latter inequality.)

SSS

$$
\begin{array}{c}
\left[\begin{array}{c} \nu \\ \vdots \\ A_i \\ !_{y_i<v_i(p)+w_i} A_i \end{array}\right] \quad
\left[\begin{array}{ccccc}
 & & \mu & & \\
\vdots & & \vdots & \vdots & \vdots \\
?_{z<q_i(x)} A_i^{\perp}[y_i := v_i(x)+z] & \cdots & ?_{z<q_j(x)} A_j^{\perp}[y_j := v_j(x)+z] & \cdots & B \\
?_{y_i<v_i(p)+w_i} A_i^{\perp} & \cdots & ?_{y_j<v_j(p)+w_j} A_j^{\perp} & \cdots & !_{x<p} B
\end{array}\right]
\end{array}
$$

reduces to:

$$
\left[\begin{array}{c}
\left[\begin{array}{c} \nu[y_i := v_i(x)+z] \\ \vdots \\ A_i[y_i := v_i(x)+z] \\ !_{z<q_i(x)} A_i[y_i := v_i(x)+z] \end{array}\right] \quad
\left[\begin{array}{ccccc}
 & & \mu & & \\
\vdots & & \vdots & \vdots & \vdots \\
?_{z<q_i(x)} A_i^{\perp}[y_i := v_i(x)+z] & \cdots & ?_{z<q_j(x)} A_j^{\perp}[y_j := v_j(x)+z] & \cdots & B \\
 & \cdots & ?_{z<q_j(x)+w_j} A_j^{\perp}[y_j := v_j(x)+z] & \cdots & !_{x<p} B
\end{array}\right]
\end{array}
$$

The weight decreases from

$$
\sum_{y_i<v_i(p)+w_i} (\|\nu\|(y_i)+1) + \sum_{x<p} (\|\mu\|(x)+1) + (2n+2)p + n + 3
$$

to:

$$
\begin{aligned}
&\sum_{x<p} \left(\sum_{z<q_i(x)} (\|\nu\|(v_i(x)+z)+1) + \|\mu\|(x) + 2 \right) + 2np + n + 1 \\
&\quad = \sum_{x<p} \left(\sum_{z<q_i(x)} (\|\nu\|(v_i(x)+z)+1) \right) + \sum_{x<p} (\|\mu\|(x)+1) + (2n+1)p + n + 1 \\
&\quad = \sum_{y_i<v_i(p)} (\|\nu\|(y_i)+1) + \sum_{x<p} (\|\mu\|(x)+1) + (2n+1)p + n + 1
\end{aligned}
$$

$$\leq \sum_{y_i < v_i(p)+w_i} (\|\nu\|(y_i)+1) + \sum_{x<p} (\|\mu\|(x)+1) + (2n+1)p + n + 1.$$

Therefore:

Theorem 4.4 *In any BLL proof net ν, any sequence of reductions on reducible cuts must terminate in at most $\|\nu\|$ steps.*

It may be readily seen that the reductions are locally confluent (i.e. weak Church-Rosser), and hence that:

Proposition 4.5 *The proof net reductions on reducible cuts satisfy the Church-Rosser property.*

Definition 4.6 The *irreducible form* of a proof net π is the result of eliminating all reducible cuts in π.

Elimination of reducible cuts yields a kind of subformula property given in Lemma 4.8 below. Let us first begin with a

Definition 4.7 A formula in the expanded language for BLL (see Section 3.4) is *accessible* if each negative occurrence of a universal quantifier or a bounded exclamation mark and each positive occurrence of an existential quantifier or a bounded question mark, is nested within a positive occurrence of a bounded exclamation mark (i.e. negative occurrence of a bounded question mark).

The proof of the following Lemma is left to the reader.

Lemma 4.8 *An irreducible proof net with accessible conclusions contains only boxed cuts.*

While the analog of this lemma can be established within the BLL sequent calculus, the argument is much more direct and perspicuous by means of proof nets.

BLL versions of polymorphic definitions of most common data types (lists, trees, etc.) will not be accessible in the sense of the definition above. The solution to this problem will be presented in the next section.

5 Normalization in BLL as Polynomial Time Computation

An aspect of modularity in BLL is that the notion of size of data is given by their type. For example, the data type of tally natural numbers of size at most x is:

$$\mathbf{N}_x \equiv \forall\alpha\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)\ ) \multimap (\alpha(0) \multimap \alpha(x)\ )$$

From now on, we simplify notation, associating the linear implication $\multimap$ to the right. Also the scope of the quantifier is the maximum possible. Note that erasing the resource information gives the linear logic version of the polymorphic type of natural numbers $\forall\alpha\,!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$, ([20], Chapter 11) in which the (tally) natural numbers are represented as freely generated by a tally "successor" function, by reusing this function under iteration as much as one likes. Our definition of $\mathbf{N}_x$ follows the same pattern, except that access to the "successor" function is allowed only up to x times (see Example 5.1).

Similarly, we consider the type of lists on two symbols, of size at most x:

$$\mathbf{N}_x^2 \equiv \forall\alpha\,!_{y<x}(\alpha(y) \multimap \alpha(y+1)\ ) \multimap !_{y<x}(\alpha(y) \multimap \alpha(y+1)\ ) \multimap \alpha(0) \multimap \alpha(x)$$

Again, erasing the resource information yields the linear logic version of the polymorphic type of lists on two symbols $\forall\alpha\,!(\alpha \multimap \alpha\ ) \multimap !(\alpha \multimap \alpha\ ) \multimap \alpha \multimap \alpha$, where such lists are represented as freely generated by two "successor" functions ("append first symbol" or "append second symbol"), by reusing these functions under combinations of iteration and composition as much as one likes. Our definition of $\mathbf{N}_x^2$ follows the same pattern, except that access to the two "successors" is allowed only up to x times (see Example 5.2).

We choose to write the first symbol as 1 and the second symbol as 2, anticipating the dyadic notation used in Section 6 below.

Most common data types (lists, trees, etc.) can be given a similar treatment in BLL, by maintaining the analogy with their representation in system $\mathcal{F}$ given, for example, in [20]: section 11.4. For the purposes of establishing the connection between normalization in BLL and the ordinary notion of polynomial time computability, we shall concentrate on the type of dyadic lists and on a simpler but related type of tally natural numbers.

Example 5.1 : The tally natural number **2** is represented by the following cut-free BLL proof of $\vdash \mathbf{N}_2$. (Hint: reading BLL proofs bottom up gives a much better understanding of the structure).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha(0)\vdash\alpha(0) \quad \alpha(1)\vdash\alpha(1)}{\alpha(0)\multimap\alpha(1),\alpha(0)\vdash\alpha(1)}\multimap\mathrm{L} \quad \alpha(2)\vdash\alpha(2)}{\alpha(0)\multimap\alpha(1),\alpha(1)\multimap\alpha(2),\alpha(0)\vdash\alpha(2)}\multimap\mathrm{L}}{!_{y<1}(\alpha(y)\multimap\alpha(y+1)),\alpha(1)\multimap\alpha(2),\alpha(0)\vdash\alpha(2)}!\mathrm{D}}{!_{y<1}(\alpha(y)\multimap\alpha(y+1)),!_{y<1}(\alpha(y+1)\multimap\alpha(y+2)),\alpha(0)\vdash\alpha(2)}!\mathrm{D}}{!_{y<2}(\alpha(y)\multimap\alpha(y+1)),\alpha(0)\vdash\alpha(2)}!\mathrm{C}}{!_{y<2}(\alpha(y)\multimap\alpha(y+1))\vdash\alpha(0)\multimap\alpha(2)}\multimap\mathrm{R}}{\vdash !_{y<2}(\alpha(y)\multimap\alpha(y+1))\multimap\alpha(0)\multimap\alpha(2)}\multimap\mathrm{R}}{\vdash\forall\alpha\,!_{y<2}(\alpha(y)\multimap\alpha(y+1))\multimap\alpha(0)\multimap\alpha(2)}\forall\mathrm{R}$$

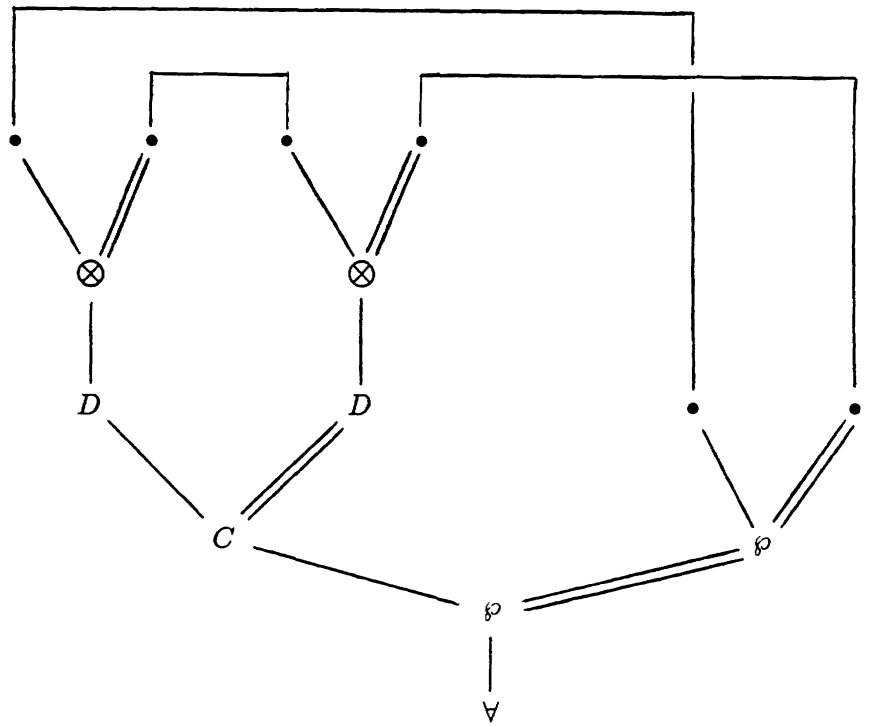
The lambda term assignment mentioned in section 3.3.1 assigns Church numeral **2** , i.e. $\lambda f.\lambda a.f(f(a))$, to the BLL proof above:

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a : \alpha(0) \ \rhd\ a : \alpha(0) \quad b : \alpha(1) \ \rhd\ b : \alpha(1)}{f : \alpha(0) \multimap \alpha(1),\, a : \alpha(0) \ \rhd\ f(a) : \alpha(1)}\ \multimap\mathrm{L} \qquad c : \alpha(2) \ \rhd\ c : \alpha(2)}{f : \alpha(0) \multimap \alpha(1),\, g : \alpha(1) \multimap \alpha(2),\, a : \alpha(0) \ \rhd\ g(f(a)) : \alpha(2)}\ \multimap\mathrm{L}}{f : !_{y<1}(\alpha(y) \multimap \alpha(y+1)),\, g : \alpha(1) \multimap \alpha(2),\, a : \alpha(0) \ \rhd\ g(f(a)) : \alpha(2)}\ !\mathrm{D}}{f : !_{y<1}(\alpha(y) \multimap \alpha(y+1)),\, g : !_{y<1}(\alpha(y+1) \multimap \alpha(y+2)),\, a : \alpha(0) \ \rhd\ g(f(a)) : \alpha(2)}\ !\mathrm{D}}{f : !_{y<2}(\alpha(y) \multimap \alpha(y+1)),\, a : \alpha(0) \ \rhd\ f(f(a)) : \alpha(2)}\ !\mathrm{C}}{f : !_{y<2}(\alpha(y) \multimap \alpha(y+1)) \ \rhd\ \lambda a.f(f(a)) : \alpha(0) \multimap \alpha(2)}\ \multimap\mathrm{R}}{\rhd\ \lambda f.\lambda a.f(f(a)) : !_{y<2}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(2)}\ \multimap\mathrm{R}}{\rhd\ \lambda f.\lambda a.f(f(a)) : \forall\alpha\, !_{y<2}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(2)}\ \forall\mathrm{R}
\end{array}
$$

Let us also display the proof net representation of the BLL proof of $\mathbf{N}_2$ given above:

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\alpha(0) \quad \alpha(1)^{\perp}}{\alpha(0) \otimes \alpha(1)^{\perp}} \qquad \dfrac{\alpha(1) \quad \alpha(2)^{\perp}}{\alpha(1) \otimes \alpha(2)^{\perp}}}{?_{y<1}(\alpha(y) \otimes \alpha(y+1)^{\perp}) \quad ?_{y<1}(\alpha(y+1) \otimes \alpha(y+2)^{\perp})} \qquad \dfrac{\alpha(0)^{\perp} \quad \alpha(2)}{\alpha(0)^{\perp} \wp \alpha(2)}}{?_{y<2}(\alpha(y) \otimes \alpha(y+1)^{\perp}) \wp (\alpha(0)^{\perp} \wp \alpha(2))}}{\forall\alpha ?_{y<2}(\alpha(y) \otimes \alpha(y+1)^{\perp}) \wp (\alpha(0)^{\perp} \wp \alpha(2))}
\end{array}
$$

This proof net determines the BLL proof given above uniquely up to the order of the rules $\multimap$L and !D . The proof net is itself uniquely determined by its conclusion $\mathbf{N}_2$ and by the labeled graph with the indicated order on binary links:



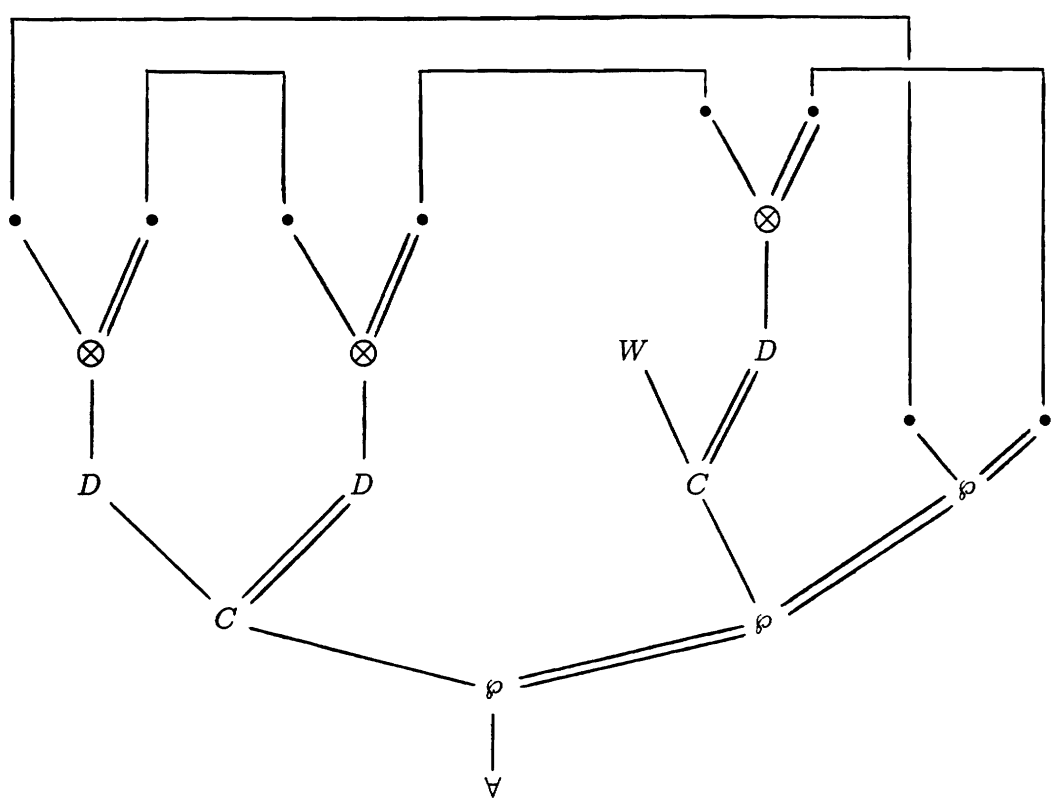
We shall use this important fact in the discussion of polynomial time computation by normalization, given below.

All the facts mentioned in this example easily generalize to any Church numeral $\lambda f.\lambda a.f(f\ldots(f(a)\ldots)$. Note that because of waste of resources expressed by axioms and rules, the tally natural number **n** can be represented by a cut-free proof of $\vdash \mathbf{N}_k$, for any $n \leq k$. This is the most general form of cut-free proofs of $\vdash \mathbf{N}_k$, up to the order of sequent calculus rules . □

Example 5.2 : The dyadic list 112 is represented by the following cut-free proof of $\vdash \mathbf{N}_3^2$.

$$
\begin{array}{c}
\alpha(0) \vdash \alpha(0) \qquad \alpha(1) \vdash \alpha(1) \\ \hline
\alpha(0) \multimap \alpha(1), \alpha(0) \vdash \alpha(1) \qquad\qquad \alpha(2) \vdash \alpha(2) \\ \hline
\alpha(0) \multimap \alpha(1), \alpha(1) \multimap \alpha(2), \alpha(0) \vdash \alpha(2) \\ \hline
!_{y<1}(\alpha(y) \multimap \alpha(y+1)), \alpha(1) \multimap \alpha(2), \alpha(0) \vdash \alpha(2) \\ \hline
!_{y<1}(\alpha(y) \multimap \alpha(y+1)), !_{y<1}(\alpha(y+1) \multimap \alpha(y+2)), \alpha(0) \vdash \alpha(2) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(2) \qquad\qquad \alpha(3) \vdash \alpha(3) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)), \alpha(2) \multimap \alpha(3), \alpha(0) \vdash \alpha(3) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)), !_{y<1}(\alpha(y+2) \multimap \alpha(y+3)), \alpha(0) \vdash \alpha(3) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)), !_{y<2}(\alpha(y) \multimap \alpha(y+1)), !_{y<1}(\alpha(y+2) \multimap \alpha(y+3)), \alpha(0) \vdash \alpha(3) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)), !_{y<3}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(3) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)), !_{y<3}(\alpha(y) \multimap \alpha(y+1)) \vdash \alpha(0) \multimap \alpha(3) \\ \hline
!_{y<3}(\alpha(y) \multimap \alpha(y+1)) \vdash !_{y<3}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(3) \\ \hline
\vdash !_{y<3}(\alpha(y) \multimap \alpha(y+1)) \multimap !_{y<3}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(3) \\ \hline
\vdash \forall \alpha !_{y<3}(\alpha(y) \multimap \alpha(y+1)) \multimap !_{y<3}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(3)
\end{array}
$$

The associated lambda term is $\lambda f.\lambda g.\lambda a.g(f(f(a))) : \mathrm{N}_3^2$. The proof net representation of the cut-free proof just given is displayed in Figure 5.1 . This proof net determines the cut-free proof given above uniquely up to the order of the rules. The proof net is in turn uniquely determined by its conclusion N_3^2 and by the labeled graph:



This important fact will be used in the discussion of polynomial time computation by normalization (see below). The facts stated in this example easily generalize to all dyadic lists. As in Example 5.1, note that because of waste of resources expressed by axioms and rules, any dyadic list of length $\leq k$ can be represented by a cut-free proof of $\mathbf{N}_k^2$. This is the most general form of cut-free proofs of $\mathbf{N}_k^2$, up to the order of sequent calculus rules. $\square$

$$
\begin{prooftree}
\AxiomC{$\alpha(0)$}
\AxiomC{$\alpha(1)^\perp$}
\RightLabel{}
\BinaryInfC{$\alpha(0) \otimes \alpha(1)^\perp$}
\RightLabel{}
\UnaryInfC{$?_{y<1}(\alpha(y) \otimes \alpha(y+1)^\perp)$}
\AxiomC{$\alpha(1)$}
\AxiomC{$\alpha(2)^\perp$}
\RightLabel{}
\BinaryInfC{$\alpha(1) \otimes \alpha(2)^\perp$}
\RightLabel{}
\UnaryInfC{$?_{y<1}(\alpha(y+1) \otimes \alpha(y+2)^\perp)$}
\RightLabel{}
\BinaryInfC{$?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp)$}
\AxiomC{}
\RightLabel{}
\UnaryInfC{$?_{y<2}(\alpha(y) \otimes \alpha(y+1)^\perp)$}
\AxiomC{$\alpha(2)$}
\AxiomC{$\alpha(3)^\perp$}
\RightLabel{}
\BinaryInfC{$\alpha(2) \otimes \alpha(3)^\perp$}
\RightLabel{}
\UnaryInfC{$?_{y<1}(\alpha(y+2) \otimes \alpha(y+3)^\perp)$}
\RightLabel{}
\BinaryInfC{$?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp)$}
\AxiomC{$\alpha(0)^\perp$}
\AxiomC{$\alpha(3)$}
\RightLabel{}
\BinaryInfC{$\alpha(0)^\perp \wp \alpha(3)$}
\RightLabel{}
\BinaryInfC{$?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp) \wp (\alpha(0)^\perp \wp \alpha(3))$}
\RightLabel{}
\BinaryInfC{$?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp) \wp (?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp) \wp (\alpha(0)^\perp \wp \alpha(3)))$}
\RightLabel{}
\UnaryInfC{$\forall \alpha\, ?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp) \wp (?_{y<3}(\alpha(y) \otimes \alpha(y+1)^\perp) \wp (\alpha(0)^\perp \wp \alpha(3)))$}
\end{prooftree}
$$

Figure 5.1

$$
\begin{prooftree}
\AxiomC{$\alpha(0) \vdash \alpha(0) \quad \alpha(x) \vdash \alpha(x)$}
\RightLabel{$\multimap$L}
\UnaryInfC{$!_{y<x}(\alpha(y) \multimap \alpha(y+1)) \vdash !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \quad \alpha(0) \multimap \alpha(x), \alpha(0) \vdash \alpha(x)$}
\RightLabel{$\multimap$L}
\UnaryInfC{$!_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x), !_{y<x}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x)$}
\RightLabel{$\forall$L}
\UnaryInfC{$\mathbf{N}_x\,,\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x)$}
\AxiomC{$\alpha(x+1) \vdash \alpha(x+1)$}
\RightLabel{$\multimap$L}
\BinaryInfC{$\mathbf{N}_x\,,\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)), \alpha(x) \multimap \alpha(x+1), \alpha(0) \vdash \alpha(x+1)$}
\RightLabel{!D}
\UnaryInfC{$\mathbf{N}_x\,,\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)), !_{y<1}(\alpha(x+y) \multimap \alpha(x+y+1)), \alpha(0) \vdash \alpha(x+1)$}
\RightLabel{!C}
\UnaryInfC{$\mathbf{N}_x\,,\, !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x+1)$}
\RightLabel{$\multimap$R}
\UnaryInfC{$\mathbf{N}_x\,,\, !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)) \vdash \alpha(0) \multimap \alpha(x+1)$}
\RightLabel{$\multimap$R}
\UnaryInfC{$\mathbf{N}_x \vdash (!_{y<x+1}(\alpha(y) \multimap \alpha(y+1))) \multimap \alpha(0) \multimap \alpha(x+1)$}
\RightLabel{$\forall$R}
\UnaryInfC{$\mathbf{N}_x \vdash \mathbf{N}_{x+1}$}
\end{prooftree}
$$

Figure 5.2: The Tally Successor $\mathbf{N}_x \vdash \mathbf{N}_{x+1}$

$$
\begin{array}{c}
\begin{array}{c}
!_{y<x}(\alpha(y) \otimes \alpha(y+1)^{\perp})^{\perp} \quad \begin{array}{c} \alpha(0) \quad \alpha(x)^{\perp} \\ \hline \alpha(0) \otimes \alpha(x)^{\perp} \end{array} \\
\hline
!_{y<x}(\alpha(y) \otimes \alpha(y+1)^{\perp})^{\perp} \otimes (\alpha(0) \otimes \alpha(x)^{\perp})
\end{array} \\
\mathbf{N}_x^{\perp}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
?_{y<x}(\alpha(y) \otimes \alpha(y+1)^{\perp}) \quad \begin{array}{c} \begin{array}{c} \alpha(x) \quad \alpha(x+1)^{\perp} \\ \hline \alpha(x) \otimes \alpha(x+1)^{\perp} \end{array} \\ \hline ?_{y<1}(\alpha(x+y) \otimes \alpha(x+y+1)^{\perp}) \end{array} \\
\hline
?_{y<x+1}(\alpha(y) \otimes \alpha(y+1)^{\perp})
\end{array} \quad \begin{array}{c} \alpha(0)^{\perp} \quad \alpha(x+1) \\ \hline \alpha(0)^{\perp} \wp \alpha(x+1) \end{array} \\
\hline
?_{y<x+1}(\alpha(y) \otimes \alpha(y+1)^{\perp}) \wp (\alpha(0)^{\perp} \wp \alpha(x+1)) \\
\mathbf{N}_{x+1}
\end{array}
$$

Figure 5.3: Proof net of the Tally Successor $\mathbf{N}_x \vdash \mathbf{N}_{x+1}$

$$
\begin{array}{c}
\alpha(0) \vdash \alpha(0) \quad \alpha(x) \vdash \alpha(x) \\
\hline
!_{y<x+1}(\alpha(y) \multimap \alpha(y+1)) \vdash !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \quad \alpha(0) \multimap \alpha(x), \alpha(0) \vdash \alpha(x) \\
\hline
!_{y<x}(\alpha(y) \multimap \alpha(y+1)) \vdash !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \quad !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x) \\
\hline
!_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x), !_{y<x}(\alpha(y) \multimap \alpha(y+1)), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x) \\
\hline
\mathbf{N}^2_x, !_{y<x}(\alpha(y) \multimap \alpha(y+1)), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x) \quad \alpha(x+1) \vdash \alpha(x+1) \\
\hline
\mathbf{N}^2_x, !_{y<x}(\alpha(y) \multimap \alpha(y+1)), \alpha(x) \multimap \alpha(x+1), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x+1) \\
\hline
\mathbf{N}^2_x, !_{y<x}(\alpha(y) \multimap \alpha(y+1)), !_{y<1}(\alpha(x+y) \multimap \alpha(x+y+1)), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x+1) \\
\hline
\mathbf{N}^2_x, !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x+1) \\
\hline
\mathbf{N}^2_x \vdash !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)) \multimap !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x+1) \\
\hline
\mathbf{N}^2_x \vdash \mathbf{N}^2_{x+1}
\end{array}
\quad
\begin{array}{l}
\multimap\mathrm{L} \\
\multimap\mathrm{L} \\
\multimap\mathrm{L} \\
\multimap\mathrm{L} \\
\forall\mathrm{L} \\
\multimap\mathrm{L} \\
!\mathrm{D} \\
!\mathrm{C} \\
\multimap\mathrm{R}\ 3\ \text{times} \\
\forall\mathrm{R}
\end{array}
$$

Figure 5.4: Append 1

$$
\dfrac{\dfrac{\dfrac{!_{y<x}(\alpha(y)\otimes\alpha(y+1)^{\perp})^{\perp} \quad \dfrac{!_{y<x}(\alpha(y)\otimes\alpha(y+1)^{\perp})^{\perp} \quad \dfrac{\alpha(0) \quad \alpha(x)^{\perp}}{\alpha(0)\otimes\alpha(x)^{\perp}}}{!_{y<x}(\alpha(y)\otimes\alpha(y+1)^{\perp})^{\perp}\otimes(\alpha(0)\otimes\alpha(x)^{\perp})}}{!_{y<x}\ (\alpha(y)\otimes\alpha(y+1)^{\perp})^{\perp}\otimes\left(!_{y<x}(\alpha(y)\otimes\alpha(y+1)^{\perp})^{\perp}\otimes(\alpha(0)\otimes\alpha(x)^{\perp})\right)}}{\mathbf{N}_x^{2\perp}}
$$

$$
\dfrac{\dfrac{\dfrac{?_{y<x}(\alpha(y)\otimes\alpha(y+1)^{\perp}) \quad \dfrac{\dfrac{\alpha(x) \quad \alpha(x+1)^{\perp}}{\alpha(x)\otimes\alpha(x+1)^{\perp}}}{?_{y<1}(\alpha(x+y)\otimes\alpha(x+y+1)^{\perp})}}{?_{y<x+1}(\alpha(y)\otimes\alpha(y+1)^{\perp})} \quad \dfrac{?_{y<x+1}(\alpha(y)\otimes\alpha(y+1)^{\perp}) \quad \dfrac{\alpha(0)^{\perp} \quad \alpha(x+1)}{\alpha(0)^{\perp}\,\wp\,\alpha(x+1)}}{?_{y<x+1}(\alpha(y)\otimes\alpha(y+1)^{\perp})\,\wp\,(\alpha(0)^{\perp}\,\wp\,\alpha(x+1))}}{?_{y<x+1}(\alpha(y)\otimes\alpha(y+1)^{\perp})\,\wp\,?_{y<x+1}(\alpha(y)\otimes\alpha(y+1)^{\perp})\,\wp\,(\alpha(0)^{\perp}\,\wp\,\alpha(x+1))}}{\mathbf{N}_{x+1}^{2}}
$$

Figure 5.5: Proof Net for Append 1

$$
\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathbf{b}\\ \vdash \mathrm{N}^2_n\end{array} \qquad \dfrac{\begin{array}{c}\vdots\ F(x:=n)\\ \vdash \mathrm{N}^2_n \multimap \mathrm{N}^2_{p(n)}\end{array} \qquad \dfrac{\mathrm{N}^2_n \vdash \mathrm{N}^2_n \quad \mathrm{N}^2_{p(n)} \vdash \mathrm{N}^2_{p(n)}}{\mathrm{N}^2_n \multimap \mathrm{N}^2_{p(n)}, \mathrm{N}^2_n \vdash \mathrm{N}^2_{p(n)}}\ \multimap\mathrm{L}}{\mathrm{N}^2_n \vdash \mathrm{N}^2_{p(n)}}\ cut}{\mathrm{N}^2_{p(n)}}\ cut \qquad \dfrac{\begin{array}{c}\text{Reuse, with } x:=p(n)\\ \vdots\\ \vdash \mathrm{N}^2_{p(n)} \multimap !_{y<1}\mathrm{N}^2_{p(n)}\end{array} \qquad \dfrac{\mathrm{N}^2_{p(n)} \vdash \mathrm{N}^2_{p(n)} \quad !_{y<1}\mathrm{N}^2_{p(n)} \vdash !_{y<1}\mathrm{N}^2_{p(n)}}{\mathrm{N}^2_{p(n)} \multimap !_{y<1}\mathrm{N}^2_{p(n)}, \mathrm{N}^2_{p(n)} \vdash !_{y<1}\mathrm{N}^2_{p(n)}}\ \multimap\mathrm{L}}{\mathrm{N}^2_{p(n)} \vdash !_{y<1}\mathrm{N}^2_{p(n)}}\ cut}{\vdash !_{y<1}\mathrm{N}^2_{p(n)}}\ cut
$$

Figure 5.6: F represents function ϕ

Example 5.3 : The successor on the tally natural numbers is represented by the cut-free BLL proof of $\mathbf{N}_x \vdash \mathbf{N}_{x+1}$ shown in Figure 5.2.

The reader will easily verify that the lambda term assignment mentioned in section 3.3.1 yields:

$$e : \mathbf{N}_x \ \rhd \lambda f.\lambda a.f(e(f)(a)) : \mathbf{N}_{x+1}$$

or, equivalently,

$$e : \mathbf{N}_x \ \rhd \lambda f.f\circ(e(f)) : \mathbf{N}_{x+1}$$

The proof net representation of this BLL proof is shown in Figure 5.3 . □

Example 5.4 : There are two successors on dyadic lists. One of them, "Append 1" , is displayed in Figure 5.4. The lambda term assignment mentioned in section 3.3.1 yields:

$$e : \mathbf{N}_x^2 \ \rhd \lambda f.\lambda g.\lambda a.f(e(f)(g)(a)) : \mathbf{N}_x^2$$

The proof net corresponding to the BLL proof given in Figure 5.4 is shown in Figure 5.5. □

Example 5.5 : A *Reuse* or *Storage Functional* on dyadic lists is given by the following proof of $\vdash \mathbf{N}_x^2 \multimap !_{y<1}\mathbf{N}_x^2$, where x, y are resource variables. Let ν be the canonical proof of $\vdash \mathbf{N}_0^2$ representing the empty list (analogous to Example 5.2), and let s_1 and s_2 be the canonical proofs of $\mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2$, representing the two dyadic successors (see Example 5.4). Consider the following proof E_0:

$$\dfrac{\begin{array}{c}\vdots\ \nu\\ \vdash \mathbf{N}_0^2\end{array}}{\vdash !_{y<1}\mathbf{N}_0^2}\ \mathrm{S!}$$

and consider the proofs σ_1, σ_2 , where σ_1 is:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\ s_1\\ \mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2\end{array}}{!_{y<1}\mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2}\ \mathrm{!D}}{!_{y<1}\mathbf{N}_z^2 \vdash !_{y<1}\mathbf{N}_{z+1}^2}\ \mathrm{S!}}{\vdash !_{y<1}\mathbf{N}_z^2 \multimap !_{y<1}\mathbf{N}_{z+1}^2}\ \multimap\mathrm{R}$$

where the reader will notice that $\sum_{y<1} 1 = 1$, as required in the formulation of the rule S! in Section 3.3. σ_2 is built from s_2 accordingly. We proceed as follows, writing $\tau[x]$ for $!_{y<1}\mathbf{N}_x^2$ and $B[x]$ for $!_{z<x}(\tau[z] \multimap \tau[z+1]) \multimap !_{z<x}(\tau[z] \multimap \tau[z+1]) \multimap \tau[0] \multimap \tau[x]$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \sigma_1}{\vdash \tau[z] \multimap \tau[z+1]}}{\vdash !_{z<x}(\tau[z] \multimap \tau[z+1])}\,\mathrm{S!} \quad \dfrac{\dfrac{\dfrac{\vdots\ \sigma_2}{\vdash \tau[z] \multimap \tau[z+1]}}{\vdash !_{z<x}(\tau[z] \multimap \tau[z+1])}\,\mathrm{S!} \quad \dfrac{\dfrac{\vdots\ \mathrm{E}_0}{\vdash \tau[0]} \quad \tau[x] \vdash \tau[x]}{\tau[0] \multimap \tau[x] \vdash \tau[x]}\,\multimap\mathrm{L}}{!_{z<x}(\tau[z] \multimap \tau[z+1]) \multimap \tau[0] \multimap \tau[x] \vdash \tau[x]}\,\multimap\mathrm{L}}{B[x] \vdash \tau[x]}\,\multimap\mathrm{L}}{\mathbf{N}_x^2 \vdash \tau[x]}\,\forall\mathrm{L}}{\vdash \mathbf{N}_x^2 \multimap \tau[x]}\,\multimap\mathrm{R}
$$

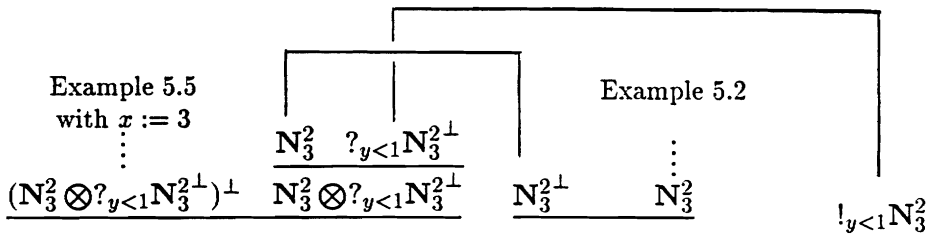
This completes the construction of the reuse functional of type $\mathbf{N}_x^2 \multimap !_{y<1}\mathbf{N}_x^2$. A similar and somewhat simpler definition can be given for the reuse functional of type $\mathbf{N}_x \multimap !_{y<1}\mathbf{N}_x$. These functionals are the BLL versions of the "shaving" or "linearizing" functionals discussed in [18], Theorem 3. They are also closely related to Krivine's recent work on "storage functionals" [26].

Let ϵ, S_1, S_2 be the lambda terms assigned to the proofs E_0, σ_1, σ_2, resp. Then $\mathbf{r} = \lambda b.b(S_1)(S_2)(\epsilon)$ is the term assigned to the reuse functional of type $\mathbf{N}_x^2 \multimap !_{y<1}\mathbf{N}_x^2$. Here ϵ is $\lambda f.\lambda g.\lambda a.a$, S_1 is $\lambda e.\lambda f.\lambda g.\lambda a.f(e(f)(g)(a))$, and S_2 is $\lambda e.\lambda f.\lambda g.\lambda a.g(e(f)(g)(a))$. We conclude this example by observing that the part of the construction of the reuse functional given by the last proof figure displayed above works in general for any type $\tau[x]$, for any proof μ of $\vdash \tau[0]$, and for any proofs ϕ and γ of $\vdash \tau[z] \multimap \tau[z+1]$. If the corresponding lambda terms are t, F, and G, then the lambda term assigned to the resulting proof of $\vdash \mathbf{N}_x^2 \multimap \tau[x]$ is $\lambda b.b(F)(G)(t)$, see the Iteration Lemma in Section 6 below. A similar observation holds for tally natural numbers $\mathbf{N}_x$ instead of dyadic lists $\mathbf{N}_x^2$. We shall also denote by $\mathbf{r}$ the lambda term associated to the tally reuse functional $\mathbf{N}_x \multimap !_{y<1}\mathbf{N}_x$. $\square$

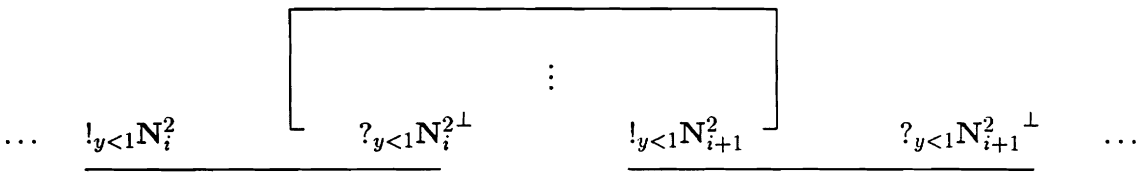
Example 5.6: Let *112* be the cut-free BLL proof of $\vdash \mathbf{N}_3^2$ representing the dyadic list 112, given in Example 5.2. Consider the following BLL proof:

$$
\dfrac{\dfrac{\vdots\ 112}{\vdash \mathbf{N}_3^2} \quad \dfrac{\begin{array}{c}\text{Example 5.5}\\ \text{with } x := 3\\ \vdots\\ \vdash \mathbf{N}_3^2 \multimap !_{y<1}\mathbf{N}_3^2\end{array} \quad \dfrac{\mathbf{N}_3^2 \vdash \mathbf{N}_3^2 \quad !_{y<1}\mathbf{N}_3^2 \vdash !_{y<1}\mathbf{N}_3^2}{\mathbf{N}_3^2 \multimap !_{y<1}\mathbf{N}_3^2,\ \mathbf{N}_3^2 \vdash !_{y<1}\mathbf{N}_3^2}\,\multimap\mathrm{L}}{\mathbf{N}_3^2 \vdash !_{y<1}\mathbf{N}_3^2}\,\mathrm{Cut}}{\vdash !_{y<1}\mathbf{N}_3^2}\,\mathrm{Cut}
$$

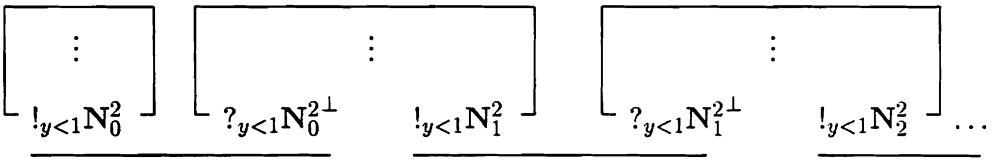
and its corresponding proof net:



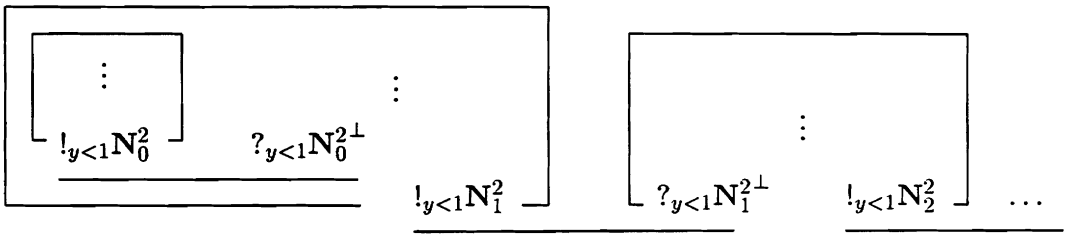
We urge the reader to work out the complete analysis of the reduction in this simple example. One important observation here is that in the configuration



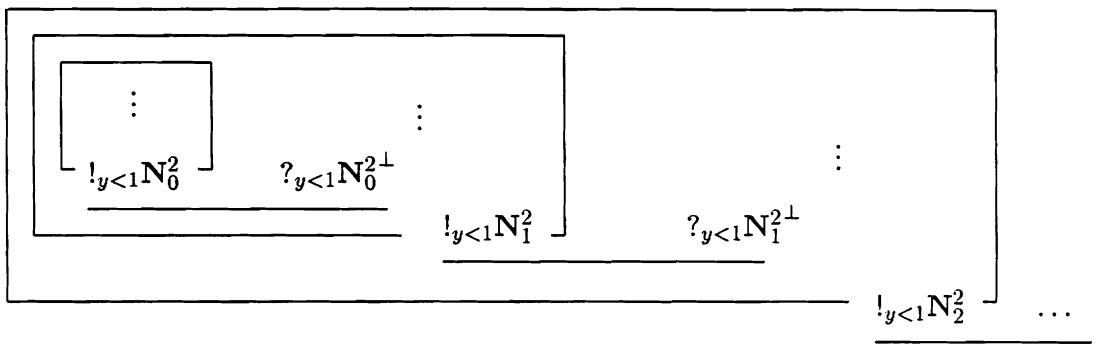
where $!_{y<1}\mathbf{N}_{i+1}^2$ is at the main door and neither $!_{y<1}\mathbf{N}_i^2$ nor $?_{y<1}(\mathbf{N}_{i+1}^2)^\perp$ is a conclusion of an axiom link, the cut involving the main door cannot be eliminated until the cut involving the auxiliary door has been eliminated (cf. the reduction step SSS). In turn, the cut involving the auxiliary door cannot be eliminated unless its other premise $!_{y<1}\mathbf{N}_i^2$ is at the main door of a box that has no auxiliary doors (again, cf. the reduction step SSS). Thus, at some point in the reduction process, we may encounter at worst the configuration



At this point, the **O** box must first "enter" the **1** box through its auxiliary door by the reduction step SSS:

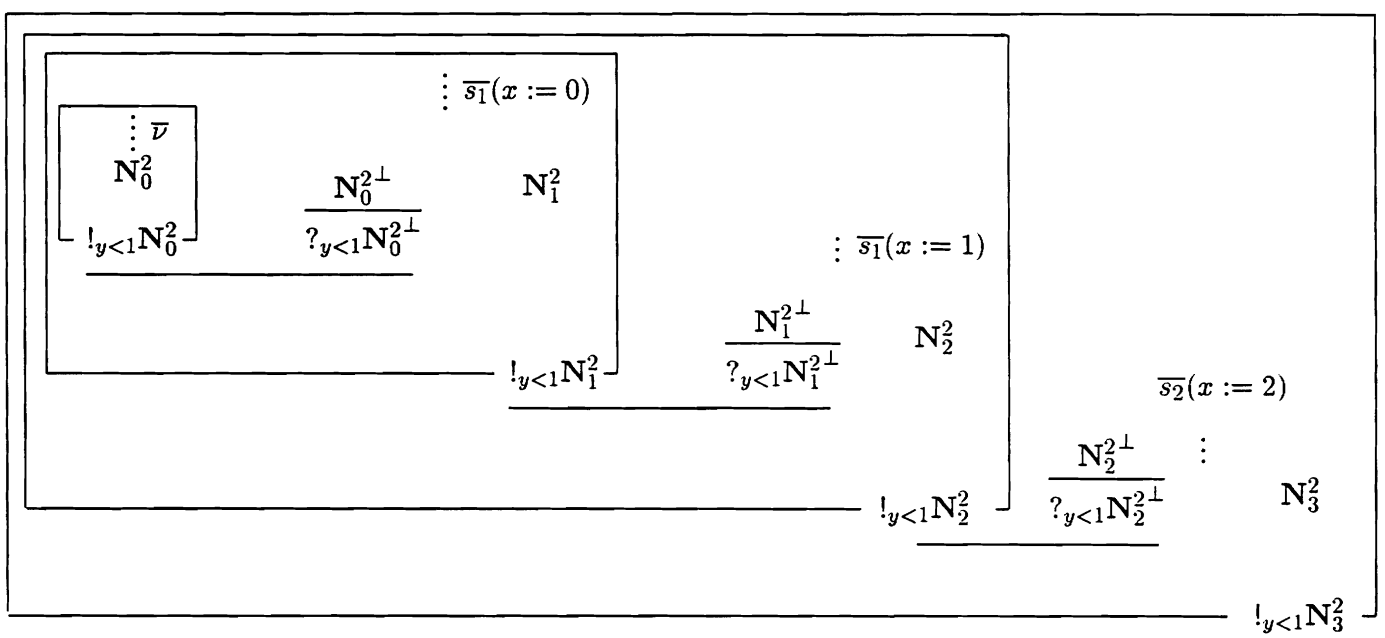


Now the **1** box must first enter the **2** box through its auxiliary door by the reduction step SSS:



etc.

The irreducible form of this proof net is:



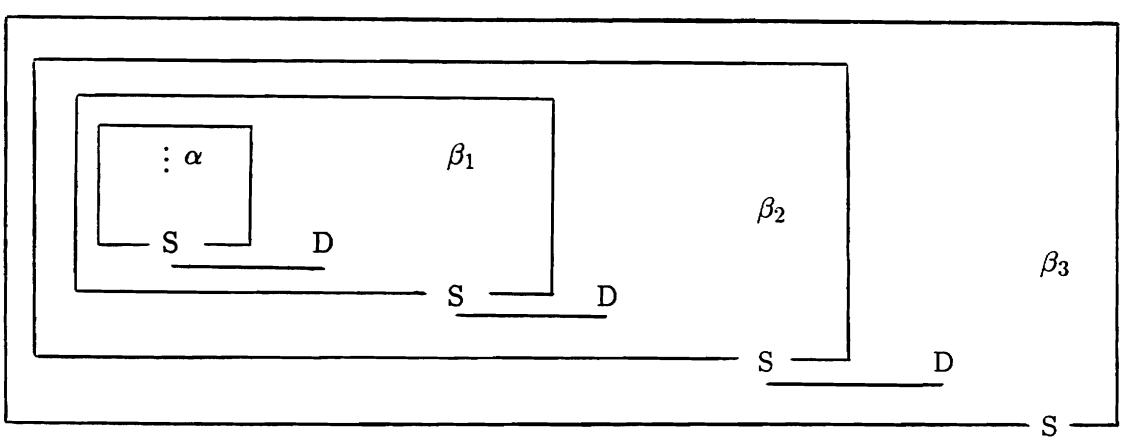
where $\overline{\nu}, \overline{s_1}, \overline{s_2}$ are proof net representations of the proofs ν, s_1 and s_2, resp. from Example 5.5. This proof net comes from the following BLL proof, up to the order of the rules:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \nu}{\vdash \mathbf{N}_0^2}}{\vdash !_{y<1}\mathbf{N}_0^2}\,\mathrm{S!} \qquad \dfrac{\dfrac{\vdots\ s_1(x:=0)}{\mathbf{N}_0^2 \vdash \mathbf{N}_1^2}}{!_{y<1}\mathbf{N}_0^2 \vdash \mathbf{N}_1^2}\,\mathrm{!D}}{\vdash \mathbf{N}_1^2}\,\mathrm{Cut}}{\vdash !_{y<1}\mathbf{N}_1^2}\,\mathrm{S!} \qquad \dfrac{\dfrac{\vdots\ s_1(x:=1)}{\mathbf{N}_1^2 \vdash \mathbf{N}_2^2}}{!_{y<1}\mathbf{N}_1^2 \vdash \mathbf{N}_2^2}\,\mathrm{!D}}{\vdash \mathbf{N}_2^2}\,\mathrm{Cut}}{\vdash !_{y<1}\mathbf{N}_2^2}\,\mathrm{S!} \qquad \dfrac{\dfrac{\vdots\ s_2(x:=2)}{\mathbf{N}_2^2 \vdash \mathbf{N}_3^2}}{!_{y<1}\mathbf{N}_2^2 \vdash \mathbf{N}_3^2}\,\mathrm{!D}}{\vdash \mathbf{N}_3^2}\,\mathrm{Cut}}{\vdash !_{y<1}\mathbf{N}_3^2}\,\mathrm{S!}
$$

Recall from Examples 5.2 and 5.5 that $\mathbf{b} = \lambda f.\lambda g.\lambda a.g(f(f(a)))$ is the lambda term assigned to the proof *112* and that $\mathbf{r} = \lambda b.b(S_1)(S_2)(\epsilon)$ is the lambda term assigned to the reuse functional on dyadic lists. The lambda term assigned to the proof given at the beginning of this example is $\mathbf{r}(\mathbf{b}) : !_{y<1}\mathbf{N}_3^2$. It is important to observe that the lambda term assigned to the last BLL proof just considered, which corresponds to the irreducible form, is *not* the ordinary normal form of $\mathbf{r}(\mathbf{b})$, since the irreducible proof (net) still does contain some cuts.

Furthermore, it is important to observe that an irreducible form such as the one given above can be recovered up to waste of resources from the information containing:

- the conclusion $!_{y<1}\mathbf{N}_k^2$,
- the formal structure obtained by forgetting the types but keeping the boxes and the rule labels, including those inside the boxes, which in this example is:



and

- the knowledge that each β_i comes from either one of the dyadic successors s_1, s_2.

In particular, the formal structures $\beta_i, \beta_j, \ldots$ distinguish between the two successors. A similar and somewhat easier observation of this nature can be made in the case of $!_{y<1}\mathbf{N}_k$. (For a deeper analysis of the last point, see Lemma 5.1 below.)

We conclude this example by noting that, in contrast to the situation just described, the cut-free proof net representing the dyadic list 112 (see Example 5.2) is the irreducible form of

$$\begin{array}{cccccccc}
\vdots\ \overline{\nu} & \vdots\ \overline{s_1}(x:=0) & \vdots\ \overline{s_1}(x:=1) & \vdots\ \overline{s_2}(x:=2) \\
\underline{\mathbf{N}_0^2 \quad \mathbf{N}_0^{2\perp}} & \underline{\mathbf{N}_1^2 \quad \mathbf{N}_1^{2\perp}} & \underline{\mathbf{N}_2^2 \quad \mathbf{N}_2^{2\perp}} & \mathbf{N}_3^2
\end{array}$$

This proof net represents the proof

$$\begin{array}{c}
\underline{\underline{\underline{\vdots\ \nu \quad \vdots\ s_1(x:=0) \\ \vdash \mathbf{N}_0^2 \quad \mathbf{N}_0^2 \vdash \mathbf{N}_1^2}}_{\vdash \mathbf{N}_1^2} \quad \underline{\vdots\ s_1(x:=1) \\ \mathbf{N}_1^2 \vdash \mathbf{N}_2^2}}_{\vdash \mathbf{N}_2^2} \quad \underline{\vdots\ s_2(x:=2) \\ \mathbf{N}_2^2 \vdash \mathbf{N}_3^2}}_{\vdash \mathbf{N}_3^2}
\end{array}$$

for which the associated lambda term is
$\lambda f.\lambda g.\lambda a.(\lambda f.\lambda g.\lambda a(\lambda f.\lambda g.\lambda a.(\lambda f.\lambda g.\lambda a.a)(f)(g)(a))(f)(g)(a))(f)(g)(a) : \mathbf{N}_3^2$. The normal form of this term is $\lambda f.\lambda g.\lambda a.g(f(f(a)))$, the term associated to the proof representing 112 (see Example 5.2). □

Now we can formulate the solution to the problem mentioned at the end of Section 4, that $\mathbf{N}_x$ and $\mathbf{N}_x^2$ are not accessible. The main idea is to employ the reuse functional and rather than normalizing reducible cuts in the proof net representation of a given proof of $\mathbf{N}_x^2$, we instead normalize reducible cuts in the proof net representation of a given proof π of $\vdash \mathbf{N}_x^2$

$$\begin{array}{c}
\underline{\underline{\vdots\ \pi \\ \vdash \mathbf{N}_x^2}\quad \underline{\overset{\text{Example 5.5}}{\vdots} \\ \vdash \mathbf{N}_x^2 \multimap !_{y<1}\mathbf{N}_x^2}\quad \underline{\underline{\mathbf{N}_x^2 \vdash \mathbf{N}_x^2 \quad !_{y<1}\mathbf{N}_x^2 \vdash !_{y<1}\mathbf{N}_x^2}\ \multimap\mathrm{L} \\ \mathbf{N}_x^2 \multimap !_{y<1}\mathbf{N}_x^2,\ \mathbf{N}_x^2 \vdash !_{y<1}\mathbf{N}_x^2}\ Cut}_{\mathbf{N}_x^2 \vdash !_{y<1}\mathbf{N}_x^2}\ Cut \\
\vdash !_{y<1}\mathbf{N}_x^2
\end{array}$$

The conclusion $!_{y<1}\mathbf{N}_x^2$ is accessible and Lemma 4.8 will apply (similarly, we work with $!_{y<1}\mathbf{N}_x$ instead of $\mathbf{N}_x$).

As we have already mentioned, this method is motivated independently by two sources. One is in the notions of "shaving" or "linearizing" functionals in [18], Theorem 3. The other source is Krivine's recent work [26], the point of which is to use leftmost reduction in lambda calculus: in spite of the obvious fact that leftmost reduction can force us to compute the same integer several times, Krivine manages to force the evaluation to occur exactly once. His storage functionals do not yield the ordinary normal form of the integer but rather something like n cuts applied to 0 and successor. Similar methods have been applied in Linear Logic in recent work of L. Regnier.

Now the rules we use in BLL are a kind of symmetrization of leftmost reduction. The idea is that the only relevant difference between left and right is that in $f(a)$, a is inside an S! box.

Lemma 5.1 below is stated for the particular case of tally integers. The case of dyadic lists is completely analogous.

Lemma 5.1 *In Bounded Linear Logic, one can construct proofs with associated lambda terms as follows:*

(i) $\rhd 0 : !_{y<1}\mathbf{N}_0$

(ii) $\rhd S : !_{y<1}\mathbf{N}_x \multimap !_{y<1}\mathbf{N}_{x+1}$

(iii) $\rhd \mathbf{r} : \mathbf{N}_x \multimap !_{y<1}\mathbf{N}_x$

such that whenever a tally integer $\mathbf{n}$ *is the ordinary normal form of a closed term* $t : \mathbf{N}_k$*, then an irreducible proof with associated term* $S^n(0) : !_{y<1}\mathbf{N}_k$ *is the result of eliminating all reducible cuts from the proof with associated term* $\mathbf{r}(t) : !_{y<1}\mathbf{N}_k$.

Before we give the proof of Lemma 5.1, let us explain the irreducible forms we are looking for. We describe them in the framework of sequent calculus; up to the order of rules this will contain all possible irreducible forms. We remind the reader of examples 5.5 and 5.6. In particular, part (iii) of the statement of Lemma 5.1 concerns the tally version of the reuse functional given in Example 5.5.

A *zero-proof* Z_a, where a is a non-negative integer, is the only proof of $!_{y<1}\mathbf{N}_a$ obtained by waste of resources from the cut-free proof with associated Church numeral $0 = \lambda f.\lambda a.a$ (recall Example 5.1).

A *successor step* S_{abc}, where the non-negative integers a,b, c satisfy $a \leq b < c$, is the only cut-free proof obtained from the cut-free proof S_x of $\mathbf{N}_x \vdash \mathbf{N}_{x+1}$ given in Figure 5.2 by the following steps: let $x = b$, obtaining a proof $\mathbf{N}_b \vdash \mathbf{N}_{b+1}$. Then waste $\mathbf{N}_a$ into $\mathbf{N}_b$ and $\mathbf{N}_{b+1}$ into $\mathbf{N}_c$, obtaining a proof $\mathbf{N}_a \vdash \mathbf{N}_c$. Applying !D , we obtain S_{abc}, which proves $!_{y<1}\mathbf{N}_a \vdash \mathbf{N}_c$.

If Π is a proof of a formula $!_{y<1}\mathbf{N}_a$, a *successor* of Π is any proof Π' of some $!_{y<1}\mathbf{N}_c$ obtained from Π by first applying a cut with some S_{abc}, then applying the S! rule.

The irreducible proof that we obtain will consist of iterated successors of some zero-proof, analogous to Example 5.6.

We emphasize that for the eventual computation, resources will be erased and therefore only the number of successors made from zero will be remembered.

Proof of **Lemma** 5.1:
By induction on $k' \leq k+1$ we shall show that either the irreducible form of $\mathbf{r}(t)$ is some iterated successor of zero, the number of iterations being less than k' , or it is a k'-iterated successor of a proof of some $!_{y<1}\mathbf{N}_a$, which is an S! box. With $k' = k+1$ only the first possibility remains. As in the case of Lemma 4.8 , the argument can be formulated either by means of sequents or proof nets, the difference being that the latter dispenses with a lot of bureaucracy. In the case $k' = 0$ we just have to show that the irreducible form of $\mathbf{r}(t)$ is a box, which is plain from Lemma 4.8 . To move from k' to $k'+1$ we can assume that the irreducible form of $\mathbf{r}(t)$ is the k'-th iterated successor of some box which proves $!_{y<1}\mathbf{N}_a$. Now observe that any exclamation mark box in the irreducible proof net has a well-defined ancestor in the original proof $\mathbf{r}(t)$. Now it may be seen that the ancestor of this box cannot lie within t . This is because in the reduction step S$\forall$ the second order abstraction term $\lambda z.!_{y<1}\mathbf{N}_z$ has been substituted in t for a generic predicate, i.e. for a second order variable which has no internal structure. Therefore, the ancestor is located within $\mathbf{r}$. Among the candidates, only two boxes type-check. First, the box for 0, which yields the first possibility of the case $k'+1$. The second one is the box for S (which comes from S_x mentioned above by applying successively !D and S!); this box for S has two conclusions and it can be changed into a box with one conclusion only by making some proof Π "enter" S through cut-elimination. In this case our proof $\mathbf{r}(t)$ will be a $(k'+1)$-iteration of Π. Now when Π enters some descendent of this box for S, Π must be a box: this comes from the restrictions on cut-elimination for storage rules. This yields the second possibility of the case $k'+1$.□
Remark: The reasoning in this proof clearly applies to Linear Logic when the resource information is omitted.

Recalling the discussion before the proof of Lemma 5.1 and Examples 5.5 and 5.6 and the subsequent discussion, let us state:

Definition 5.2 A function ϕ from dyadic lists to dyadic lists is *represented in Bounded Linear Logic by a proof F* of $\mathbf{N}^2_x \multimap \mathbf{N}^2_{p(x)}$ if for every dyadic list b of length $< n$ and the corresponding cut-free proof $\mathbf{b}$ of $\mathbf{N}^2_n$, the irreducible proof net with conclusion $!_{y<1}\mathbf{N}^2_{p(n)}$ that corresponds to the dyadic list $\phi(b)$ is the irreducible form of the proof net representation of the BLL proof displayed in Figure 5.6 .

A function ϕ from dyadic lists to dyadic lists is *representable in Bounded Linear Logic* if there exists a resource polynomial $p(x)$ and a BLL proof of $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{p(x)}$ that represents ϕ. Similarly for functions on tally natural numbers and for functions of several arguments, perhaps some tally, some dyadic. □

We again emphasize that in the actual computation by cut elimination, much

of the type information is erased first, so the result will be a graphical configuration of the successors "append 1" and "append 2" that uniquely determines $\phi(b)$.

Discussion: In regard to Definition 5.2 above, note first of all that the lambda term $\mathbf{f}$ associated to the BLL proof F has type $\mathbf{N^2} \Rightarrow \mathbf{N^2}$ in system $\mathcal{F}$ (erase all resource information and exclamation marks from the BLL proof F). Thus if $\overline{\mathbf{b}}$ is the dyadic Church numeral corresponding to the cut-free proof $\mathbf{b}$, then $\mathbf{f}(\overline{\mathbf{b}})$ has a normal form which must be a dyadic Church numeral. Let $\mathbf{r}$ be the lambda term associated to the reuse functional given in Example 5.5. $\mathbf{r}(\mathbf{f}(\overline{\mathbf{b}}))$ is the term associated to the proof given in Figure 5.6. Furthermore, because of the dyadic case of Lemma 5.1, the irreducible form in the situation described in Definition 5.2 is the same as for the following:

$$\dfrac{\begin{array}{c}\vdots\ \overline{\phi(b)}\\ \vdash \mathbf{N}^2_{p(n)}\end{array} \qquad \dfrac{\begin{array}{c}\text{Reuse, with } x{:=}p(n)\\ \vdots\\ \vdash \mathbf{N}^2_{p(n)} \multimap !_{y<1}\mathbf{N}^2_{p(n)}\end{array} \qquad \dfrac{\mathbf{N}^2_{p(n)} \vdash \mathbf{N}^2_{p(n)} \quad !_{y<1}\mathbf{N}^2_{p(n)} \vdash !_{y<1}\mathbf{N}^2_{p(n)}}{\mathbf{N}^2_{p(n)} \multimap !_{y<1}\mathbf{N}^2_{p(n)}, \mathbf{N}^2_{p(n)} \vdash !_{y<1}\mathbf{N}^2_{p(n)}}\ \multimap\mathrm{L}}{\mathbf{N}^2_{p(n)} \vdash !_{y<1}\mathbf{N}^2_{p(n)}}\ \mathrm{Cut}}{\vdash !_{y<1}\mathbf{N}^2_{p(n)}}\ \mathrm{Cut}$$

whose associated lambda term is $\mathbf{r}(\overline{\phi(b)})$, where $\overline{\phi(b)}$ is the Church numeral associated to $\phi(b)$. In other words, because of the dyadic case of Lemma 5.1 , we may equivalently require that the ordinary normal form of $\mathbf{f}(\overline{\mathbf{b}})$ must represent $\phi(b)$. Lemma 5.1 seemingly states only one direction, but the other follows also by Lemma 5.1 because of the Church-Rosser property and because $\mathbf{f}(\overline{\mathbf{b}})$ has a normal form. However our representation by irreducible forms rather than by normal forms is much more economical from the point of view of computation. In particular, our approach yields a feasible computation.

Theorem 5.3 *Any function from dyadic lists to dyadic lists represented by a proof of* $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{p(x)}$ *in Bounded Linear Logic is computable in polynomial time. Furthermore, the required polynomial can be obtained explicitly from the weight of the representing BLL proof of* $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{p(x)}$.

Proof: Let F be a proof of $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{p(x)}$ that represents a function ϕ from dyadic lists to dyadic lists. Let $\|F\|(x)$ be the weight of F and let $\rho(x)$ be the weight of the proof that defines the reuse functional, given in Example 5.5. For any dyadic list b of length at most n, let $\mathbf{b}$ be the cut-free proof of $\vdash \mathbf{N}^2_n$ representing b (see Example 5.2). Observe that the weight $\|\mathbf{b}\|$ is linear in n. Referring to the proof displayed in Figure 5.6, we readily see that its weight is $Q(n) = \|F\|(n) + \rho(n) + kn + l$, for some constants k and l. By Theorem 4.4, therefore, the irreducible form of this proof, which by Lemma 5.1 and by the discussion after Definition 5.2 uniquely determines $\phi(b)$ will be reached in at most $Q(n)$ steps. Thus it suffices to show that the computation uses only polynomial space with respect to resource parameters (if we have a procedure that uses polynomially many steps and is polynomial

space, then it is polynomial time.) The argument will apply to any proof net with conclusion $\mathbf{N}_m^2$.

We first observe that we may erase the types (i.e. formulas) and keep only the information about links and boxes. This information suffices to carry out the reduction steps. As we have noted in Example 5.6 and Lemma 5.1, after eliminating all reducible cuts, we are left with a formal trace of the irreducible form of a given proof, from which the irreducible form can be uniquely determined. In fact, the formal trace itself already determines the required dyadic list $\phi(b)$ uniquely. However, we are still left with the problem of duplication of boxes in the reduction step SSC.

The solution will involve creating a new pointer to the address of the box instead of duplicating the box explicitly. More precisely, let us assume that we have a finite list of *signatures* $\bullet \quad \bullet \quad \bullet \ldots \bullet \quad ; \bullet$, which are basically natural numbers. (A natural number n is intended to indicate n auxiliary doors of a black box.) We shall consider a "dynamical proof net" without formulas, by using formal axioms from the given finite list of signatures. Such a dynamical proof net may have many formal axiom links that have any positive number of conclusions. Any such formal axiom link with $n+1$ conclusions will refer to a unique item on the list of signatures, which is an occurrence of n. Such a formal axiom link with $n+1$ conclusions is intended to replace a maximal box with $n+1$ doors in a proof net without formulas. Therefore, these formal axiom links will also refer to a list of certain formal boxes in a way that will be inductively defined below. We will establish that the size of this dynamical proof net evolves polynomially during the elimination of reducible cuts.

The initial structure may be described as follows. Let β_i be the immediate content of the ith box in the formal structure obtained from a given proof net by erasing the formulas but keeping the names of links. β_0 is taken as minimal, i.e., there are no boxes inside β_0. The signature of β_0 is obvious: there must be a bijection between the signature and the doors of β_0, which relates the distinguished dot to the main door. β_1, on the other hand, might use β_0 as a module. We indicate that by using, if needed, only the formal axiom link that refers to the signature of β_0. Furthermore, β_1 has its own signature defined as in the case of β_0, etc. Notice that for this initial configuration, the structure of using a box inside another will be quite limited, because the nested structure is a non-rooted tree, and if the same box should occur in two different places in this tree, we cannot expect to see that at compile time. Because we are referring only to immediate subboxes, a formal box can be referred to by at most one other formal box later in the list.

The list of formal boxes will increase during the elimination of reducible cuts, but the increase will take place only in the case SSS, see Section 4. Each time such a reduction step takes place, a new formal box is created because a minor premise (a formal box with signature $; \bullet$) enters a major premise (a formal box with arbitrary signature) through a distinguished auxiliary door. Therefore the new formal box is uniquely described by:

- the address of the major premise in the previously created list,

- the address of the minor premise in the previously created list,
- the signature of the new formal box, and
- an integer $j \leq n$, where n refers to the signature of the major premise.

Let us recall that in each reduction step the weight strictly decreases, and therefore the list of formal boxes is polynomial in the resource parameters. Let us also observe that there is a fixed bound, say N, on the number of auxiliary doors of each formal box in the created list.

During the elimination of reducible cuts, at each moment we have a dynamical proof net with pointers to the list of formal boxes and to their signatures. We now show that the size of these dynamical proof nets evolves polynomially.

We have to count the number of links. This is mainly taken care of by the weight. In particular, for the formal axiom links that replace boxes with $n+1$ doors, the weight is at least $n+1$ (see section 3.5). However, the weight does not count cuts, which therefore must be counted separately. Given a cut, consider the two links leading to the premises and arbitrarily pick one of them. Recall (from section 3.5) that each link except cut has a positive weight. Because of the presence of formal axiom links in a dynamical proof net, links may have at most max{ N+1, 2 } $\leq$ N+2 conclusions.

Since we chose an injection from cuts to other links, then each such link in a dynamical proof net is related to at most N + 2 cuts. Thus the number of cuts is at most (N+2) times the weight. Therefore the total size is linear in the weight.

It remains to state the reduction steps involving formal boxes, but relying only on the information available at each step. There are four cases (see section 4):

SSW: Note that one simply destroys a part of the structure.

SSC: Create a copy of a formal axiom link (not the whole box) together with a pointer to the same place as the original.

SSS: Remove the cut with the minor premise, replace the major premise with another formal axiom with a smaller number of conclusions, and make a pointer to the new box described above.

SSD: Consider the formal box γ involved in the cut. Trace back through the hereditary major premises until we get to the initial list of formal boxes, say to the item i. Make a cut with the conclusion of β_i that gives rise to the main door of the ith formal box on the initial list. However, this formal box has side doors, all of which are linked to formal boxes through:

$$\dfrac{minor \qquad \dfrac{minor \qquad \dfrac{minor \quad i}{\vdots}}{Major}}{Major}$$
$$\gamma$$

where the Majors give the addresses for the relevant side doors. In β_i make cuts between these side doors and new axioms, each with exactly one conclusion. Each of these new axioms should now refer to the corresponding minor, the address of which is known (see the four items in the creation of new boxes discussed above) . □

The argument is analogous in the tally case. We obtain:

Theorem 5.4 *Any function from tally natural numbers to tally natural numbers represented by a proof of* $\vdash \mathbf{N}_x \multimap \mathbf{N}_{p(x)}$ *in Bounded Linear Logic is computable in polynomial time with respect to tally length. Furthermore, the required polynomial can be obtained explicitly from the weight of the representing BLL proof of* $\vdash \mathbf{N}_x \multimap \mathbf{N}_{p(x)}$.□

Another proof of Theorem 5.4 will be given at the end of Section 6.

6 Representing Polynomial Time Functions in BLL

In this section we show the converse of Theorem 5.3:

Theorem 6.1 *Every polynomial time computable function can be represented in Bounded Linear Logic by a proof of* $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{p(x)}$, *for some resource polynomial* p.

The reader will recall that the notion of "function represented by a proof in BLL" is specified in Definition 5.2.

This section is simply a series of exercises about the flexibility of the typing rules of BLL. Let us point out that the lambda terms associated to the BLL proofs (cf. 3.3.1) that we construct for the purpose of representing polynomial time functions on natural numbers are convertible to the type erasures of lambda terms ordinarily used for representing these functions in system $\mathcal{F}$ as functions on dyadic lists, i.e. as terms of type $\mathbf{N}^2 \Rightarrow \mathbf{N}^2$. Thus, from this point of view, we do not actually construct any new representations. In particular, the question of representing fast algorithms and the question of tightness of time bounds arising from BLL representations will be studied elsewhere. Here we simply check that resource information is incorporated into the BLL inference rules in a flexible enough way to express the ordinary lambda term representations of certain functions on dyadic lists. We rely on Cobham's well-known characterization of the class of polynomial time functions $\mathcal{P}$ as the smallest class of functions closed under composition and limited recursion on notation, and containing certain initial functions [5].

Remark: The presentations in the literature of Cobham's characterization of $\mathcal{P}$ vary, e.g. [28] considers functions whose inputs and outputs are only those binary lists that encode natural numbers under the usual binary encoding, while [30] and [32] consider functions whose inputs and outputs are natural numbers in dyadic

notation. We consider dyadic notation, customarily written as lists of 1's and 2's, instead of binary lists of 0's and 1's. Denote the function that appends symbol i to (the end of) list l by $l * i$. Formally, we define $dya(0) = \varepsilon =$ the empty list ; $dya(2i+1) = dya(i) * 1, dya(2i+2) = dya(i) * 2$. In this way we obtain a one-to-one correspondence between natural numbers and dyadic lists.

The following two lemmas are obtained by straightforward generalization of the construction given in Example 5.5.

Lemma 6.2 (Iteration Lemma)
1. Tally Case. *Let* $\tau[z]$ *be a type in which all free occurrences of resource parameter* z *are positive. Given BLL proofs of* $\vdash \tau[0]$ *and* $\vdash \tau[z] \multimap \tau[z+1]$ *with associated lambda terms* t *and* F*, respectively, one can construct a proof of* $\vdash \mathbf{N}_x \multimap \tau[x]$ *with associated lambda term* $\lambda n.(n)(F)(t)$.
2. Dyadic Case. *Let* $\tau[z]$ *be a type in which the free occurrences of resource parameter* z *are positive. Given a BLL proof of* $\vdash \tau[0]$ *, and two BLL proofs of* $\vdash \tau[z] \multimap \tau[z+1]$ *with associated lambda terms* t*,* F*, and* G*, respectively, one can construct a BLL proof of* $\vdash \mathbf{N}_x^2 \multimap \tau[x]$ *with associated lambda term* $\lambda w.(w)(F)(G)(t)$. □

Intuitively, the role of the Iteration Lemma (say, its dyadic case) may be described as follows. For instance, $\tau[z]$ may be $\mathbf{N}_{q(z)}^2$ and the given BLL proofs of $\vdash \mathbf{N}_{q(0)}^2$ and (two proofs of) $\vdash \mathbf{N}_{q(z)}^2 \multimap \mathbf{N}_{q(z+1)}^2$ may represent a dyadic list α and functions f and g, respectively. Then the function h defined by iteration

$$\begin{aligned} h(\varepsilon) &= \alpha \\ h(b*1) &= f(h(b)) \\ h(b*2) &= g(h(b)) \end{aligned}$$

will be represented by the proof of $\vdash \mathbf{N}_x^2 \multimap \mathbf{N}_{q(x)}^2$. (Strictly speaking, the sense in which the two given BLL proofs of $\vdash \mathbf{N}_{q(z)}^2 \multimap \mathbf{N}_{q(z+1)}^2$ represent functions f and g, respectively, is not specified by Definition 5.2, but the required more general definition is obvious from Definition 5.2).

Lemma 6.3 (Reuse Lemma)
1. Tally Case: *Let* 0 *and* S *be lambda terms expressing the tally numeral zero and the tally successor, respectively, described in Examples 5.1 and 5.3. One can construct a BLL proof of* $\vdash \mathbf{N}_x \multimap !_{y<y'}\mathbf{N}_x$ *with associated lambda term* $\lambda n.n(S)(0)$.
2. Dyadic Case: *Let* ϵ*,* S_1*, and* S_2 *be lambda terms expressing the empty list and the two successors "Append 1" and "Append 2", respectively, described in Examples 5.4 and 5.5. One can construct a BLL proof of* $\vdash \mathbf{N}_x^2 \multimap !_{y<y'}\mathbf{N}_x^2$ *with associated lambda term* $\lambda b.b(S_1)(S_2)(\epsilon)$. □

In either case of Lemma 6.3, we denote the resulting lambda term by $\mathbf{r}$.

Among the consequences of the Reuse Lemma are the following derived inference rules:

$$\frac{\Gamma \vdash B}{\Gamma\,,\,\mathbf{N}_x^2 \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma\,,\,\mathbf{N}_x \vdash B}$$

$$\frac{\Gamma\,,\,\mathbf{N}_x^2\,,\,\mathbf{N}_x^2 \vdash B}{\Gamma\,,\,\mathbf{N}_x^2 \vdash B} \qquad \frac{\Gamma\,,\,\mathbf{N}_x\,,\,\mathbf{N}_x \vdash B}{\Gamma\,,\,\mathbf{N}_x \vdash B}$$

i.e., unrestricted Weakening and Contraction on $\mathbf{N}_x^2$ and $\mathbf{N}_x$. Furthermore, the induced lambda term assignment is:

$$\frac{\vec{c} : \Gamma \;\rhd\; t : B}{\vec{c} : \Gamma\,,\,a : \mathbf{N}_x^2 \;\rhd\; t : B} \qquad \frac{\vec{c} : \Gamma \;\rhd\; t : B}{\vec{c} : \Gamma\,,\,a : \mathbf{N}_x \;\rhd\; t : B}$$

$$\frac{\vec{c} : \Gamma\,,\,a : \mathbf{N}_x^2\,,\,b : \mathbf{N}_x^2 \;\rhd\; t : B}{\vec{c} : \Gamma\,,\,a : \mathbf{N}_x^2 \;\rhd\; t[a := \mathbf{r}(a), b := \mathbf{r}(a)] : B} \qquad \frac{\vec{c} : \Gamma\,,\,a : \mathbf{N}_x\,,\,b : \mathbf{N}_x \;\rhd\; t : B}{\vec{c} : \Gamma\,,\,a : \mathbf{N}_x \;\rhd\; t[a := \mathbf{r}(a), b := \mathbf{r}(a)] : B}$$

Another way to apply the Reuse Lemma is through interaction with the S! rule, e.g. as in the proof of Proposition 6.5 below.

Let us first show that polynomials on tally natural numbers are representable in BLL.

Proposition 6.4 *Tally addition is representable by a BLL proof of* $\vdash \mathbf{N}_x \otimes \mathbf{N}_y \multimap \mathbf{N}_{x+y}$

Proof: As in system $\mathcal{F}$ and untyped lambda calculus, the intuitive motivation for this representation is the equation

$$f^n \circ f^m = f^{n+m} \qquad ,$$

where f^k is the k-fold composition $f \circ f \circ \ldots \circ f$ of an endofunction with itself. First, using $\multimap$L and $\forall$L with $\mathbf{N}_y$ as $\forall \alpha A$ and $\lambda z.\alpha(x+z)$ as T yields a BLL proof of:

$$\mathbf{N}_y\,,\,!_{z<y}(\alpha(x+z) \multimap \alpha(x+z+1))\,,\,\alpha(x) \vdash \alpha(x+y) \qquad .$$

On the other hand, a BLL proof of:

$$\mathbf{N}_x\,,\,!_{z<x}(\alpha(z) \multimap \alpha(z+1))\,,\,\alpha(0) \vdash \alpha(x)$$

is readily obtained by $\forall$L and $\multimap$L. Now we use Cut on these two proofs, with $\alpha(x)$ as the cut formula, and thus obtain a BLL proof of:

$$\mathbf{N}_x\,,\,\mathbf{N}_y\,,\,!_{z<x}(\alpha(z) \multimap \alpha(z+1))\,,\,!_{z<y}(\alpha(x+z) \multimap \alpha(x+z+1))\,,\,\alpha(0) \vdash \alpha(x+y) \qquad ,$$

whence by Contraction:

$$\mathbf{N}_x\,,\,\mathbf{N}_y\,,\,!_{z<x+y}(\alpha(z) \multimap \alpha(z+1))\,,\,\alpha(0) \vdash \alpha(x+y) \qquad .$$

By $\multimap$R and $\forall$R we obtain:

$$\mathbf{N}_x\,,\,\mathbf{N}_y \vdash \mathbf{N}_{x+y} \quad .$$

Now the desired BLL proof of

$$\vdash \mathbf{N}_x \otimes \mathbf{N}_y \multimap \mathbf{N}_{x+y}$$

is obtained by $\otimes$L and $\multimap$R . $\square$

Proposition 6.5 *Tally multiplication is representable by a BLL proof of* $\vdash \mathbf{N}_x \otimes \mathbf{N}_y \multimap \mathbf{N}_{xy}$.

Proof: Again as in system $\mathcal{F}$, the intuitive motivation for this representation is the equation:

$$(f^n)^m = f^{nm}$$

where f^k is the k-fold composition $f \circ f \circ \ldots \circ f$ of an endofuction with itself. First, using $\multimap$L and using $\forall$L with $\mathbf{N}_y$ as $\forall \alpha A$ and $\lambda z'.\alpha(zy+z')$ as T yields a BLL proof of:

$$\mathbf{N}_y\,,\,!_{z'<y}(\alpha(zy+z') \multimap \alpha(zy+z'+1))\,,\,\alpha(zy) \vdash \alpha(zy+y) \quad ,$$

hence by $\multimap$R:

$$\mathbf{N}_y\,,\,!_{z'<y}(\alpha(zy+z') \multimap \alpha(zy+z'+1)) \vdash \alpha(zy) \multimap \alpha(zy+y) \quad ,$$

and thus by Dereliction:

$$!_{z'<1}\mathbf{N}_y\,,\,!_{z'<y}(\alpha(zy+z') \multimap \alpha(zy+z'+1)) \vdash \alpha(zy) \multimap \alpha(zy+y) \quad .$$

One now applies the S! rule [4], where the formula $\alpha(zy+z') \multimap \alpha(zy+z'+1)$ is thought of as $\alpha(y') \multimap \alpha(y'+1)[y' := zy+z']$, and where y' does not occur in $!_{z'<1}\mathbf{N}_y$. One obtains a BLL proof of:

$$!_{y'<x}\mathbf{N}_y\,,\,!_{y'<xy}(\alpha(y') \multimap \alpha(y'+1)) \vdash !_{z<x}\alpha(zy) \multimap \alpha(zy+y) \quad .$$

On the other hand, using $\multimap$L and using $\forall$L, with $\mathbf{N}_x$ as $\forall \alpha A$ and $\lambda z.\alpha(zy)$ as T yields a BLL proof of:

$$\mathbf{N}_x\,,\,!_{z<x}(\alpha(zy) \multimap \alpha(zy+y))\,,\,\alpha(0) \vdash \alpha(xy) \quad .$$

Now use Cut on the two proofs constructed, with $!_{z<x}(\alpha(zy) \multimap \alpha(zy+y))$ as the cut formula, obtaining a BLL proof of:

[4] In which we set, in order, $y = y', z = z', x = z, p = x, q_1(z) = 1, q_2(z) = y$; thus, $v_1(p) = p$, and $v_2(p) = py$.

$$\mathbf{N}_x\,,\,!_{y'<x}\mathbf{N}_y\,,\,!_{y'<xy}(\alpha(y')\multimap\alpha(y'+1))\,,\,\alpha(0)\vdash\alpha(xy)\quad,$$

hence by $\multimap$R and $\forall$R:

$$\mathbf{N}_x\,,\,!_{y'<x}\mathbf{N}_y\vdash\mathbf{N}_{xy}\quad,$$

By the Reuse Lemma and by using Cut, one obtains

$$\mathbf{N}_x\,,\,\mathbf{N}_y\vdash\mathbf{N}_{xy}\quad.$$

and thus the desired BLL proof of

$$\vdash\mathbf{N}_x\otimes\mathbf{N}_y\multimap\mathbf{N}_{xy}\quad.$$

can be obtained by $\otimes$L and $\multimap$R. □

Corollary 6.6 *Any polynomial $p(x_1,\ldots x_k)$ in k arguments with non-negative integer coefficients is representable by a BLL proof of*
$\vdash\mathbf{N}_{x_1}\otimes\mathbf{N}_{x_2}\ldots\otimes\mathbf{N}_{x_k}\multimap\mathbf{N}_{p(x_1,\ldots,x_k)}$.

Proof: First, note that any such polynomial is a resource polynomial. Next, since $x^k=x\cdot\ldots\cdot x$ (k times) , the proof of Proposition 6.5 yields a BLL proof of:

$$\mathbf{N}_x\,,\,\ldots\,,\,\mathbf{N}_x\vdash\mathbf{N}_{x^k}$$

and hence by multiple contraction on $\mathbf{N}_x$ (justified by the Reuse Lemma) one obtains a BLL proof of $\mathbf{N}_x\vdash\mathbf{N}_{x^k}$ and hence of $\vdash\mathbf{N}_x\multimap\mathbf{N}_{x^k}$ by $\multimap$R. Similarly, nx can be represented by a BLL proof of $\mathbf{N}_x\vdash\mathbf{N}_{nx}$. Now the corollary follows by Proposition 6.4 and by Reuse. □

We have already represented in BLL dyadic constants, the two dyadic successors, and projections, cf. Examples 5.2 and 5.4 and comments after the Reuse Lemma (Lemma 6.3). We now represent in BLL a relatively fast-growing function that can be used to majorize the initial functions in Cobham's characterization of $\mathcal{P}$.

Proposition 6.7 *Let $g(a,b)$ be defined by iteration on b:*

$$\begin{aligned} g(a,\varepsilon) &= 2\\ g(a,b*i) &= f(a,g(a,b))\quad i=1,2\quad, \end{aligned}$$

where $f(a,d)$ is itself defined by iteration on a:

$$\begin{aligned} f(\varepsilon,d) &= d\\ f(a*i,d) &= (f(a,d))*1\quad i=1,2\quad. \end{aligned}$$

The function f is representable by a BLL proof of $\vdash\mathbf{N}^2_x\otimes\mathbf{N}^2_y\multimap\mathbf{N}^2_{y+x}$. The function g is representable by a BLL proof of $\vdash\mathbf{N}^2_x\otimes\mathbf{N}^2_y\multimap\mathbf{N}^2_{xy+1}$.

Proof: First we shall use the dyadic case of the Iteration Lemma (Lemma 6.2) to represent f. It suffices to consider $\lambda d.f$. Let $\sigma[z]$ be $\mathbf{N}^2_y \multimap \mathbf{N}^2_{y+z}$. For $z = 0$ let $\vdash \mathbf{N}^2_y \multimap \mathbf{N}^2_y$ be obtained from an Axiom by $\multimap$R.

Both cases of the iterative step are the same. The required BLL proof of $\vdash \sigma[z] \multimap \sigma[z+1]$ is constructed as follows. Consider the BLL proof s_1 of $\mathbf{N}^2_x \vdash \mathbf{N}^2_{x+1}$ given in Figure 5.4 (which represents the successor "Append 1") and set $x := y + z$ in that entire BLL proof (with the bound resource variable y renamed to y'). We obtain a BLL proof of $\mathbf{N}^2_{y+z} \vdash \mathbf{N}^2_{y+z+1}$. Also consider the BLL proof of $\mathbf{N}^2_y \multimap \mathbf{N}^2_{y+z}, \mathbf{N}^2_y \vdash \mathbf{N}^2_{y+z}$ obtained by applying one instance of $\multimap$L to two Axioms. Now use Cut, where the cut formula is $\mathbf{N}^2_{y+z}$, thus concluding $\mathbf{N}^2_y \multimap \mathbf{N}^2_{y+z}, \mathbf{N}^2_y \vdash \mathbf{N}^2_{y+z+1}$. Now use $\multimap$R twice. The Iteration Lemma then yields a BLL proof of $\vdash \mathbf{N}^2_x \multimap (\mathbf{N}^2_y \multimap \mathbf{N}^2_{y+x})$. The corresponding BLL proof of $\vdash \mathbf{N}^2_x \otimes \mathbf{N}^2_y \multimap \mathbf{N}^2_{y+x}$ represents the function f introduced above.

Now let us represent the function g. We consider $\lambda a.g$. Let $\tau[z]$ be $\mathbf{N}^2_x \multimap \mathbf{N}^2_{xz+1}$. For $z = 0$, let $\vdash \mathbf{N}^2_1$ be the BLL proof representing the list 2, cf. Example 5.2 for comparison. Apply weakening on $\mathbf{N}^2_x$, as justified by the Reuse Lemma (cf. the comments after Lemma 6.3) and thus obtain a BLL proof of $\mathbf{N}^2_x \vdash \mathbf{N}^2_1$. Apply $\multimap$R. The resulting BLL proof of $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_1$ represents the constant function **2**.

In the iterative step both cases are the same. The desired BLL proof of $\vdash \tau[z] \multimap \tau[z+1]$ may be obtained as follows. From the BLL proof of $\vdash \mathbf{N}^2_x \otimes \mathbf{N}^2_y \multimap \mathbf{N}^2_{y+x}$ that represents f, given above, and the BLL proof of $\mathbf{N}^2_x, \mathbf{N}^2_y, \mathbf{N}^2_x \otimes \mathbf{N}^2_y \multimap \mathbf{N}^2_{y+x} \vdash \mathbf{N}^2_{y+x}$ obtained by $\otimes$R and $\multimap$L from Axioms, one can construct a BLL proof of $\mathbf{N}^2_x, \mathbf{N}^2_y \vdash \mathbf{N}^2_{y+x}$ by using Cut, where the cut formula is $\mathbf{N}^2_x \otimes \mathbf{N}^2_y \multimap \mathbf{N}^2_{y+x}$. Let $y = xz + 1$ in this entire BLL proof, renaming bound variables if necessary. One obtains a BLL proof of $\mathbf{N}^2_x, \mathbf{N}^2_{xz+1} \vdash \mathbf{N}^2_{x(z+1)+1}$. On the other hand, a BLL proof of $\mathbf{N}^2_x \multimap \mathbf{N}^2_{xz+1}, \mathbf{N}^2_x \vdash \mathbf{N}^2_{xz+1}$ is readily available by using $\multimap$L on Axioms. Now use Cut on the latter two BLL proofs, where the cut formula is $\mathbf{N}^2_{xz+1}$, yielding:

$$\mathbf{N}^2_x, \mathbf{N}^2_x, \mathbf{N}^2_x \multimap \mathbf{N}^2_{xz+1} \vdash \mathbf{N}^2_{x(z+1)+1} \quad .$$

Now use contraction on $\mathbf{N}^2_x$ as justified by the Reuse Lemma (cf. the comments after Lemma 6.3). One obtains:

$$\mathbf{N}^2_x, \mathbf{N}^2_x \multimap \mathbf{N}^2_{xz+1} \vdash \mathbf{N}^2_{x(z+1)+1} \quad .$$

Now use $\multimap$R twice to obtain the required BLL proof of $\vdash \tau[z] \multimap \tau[z+1]$. Hence by the Iteration Lemma (Lemma 6.2) one constructs a BLL proof of $\mathbf{N}^2_y \multimap (\mathbf{N}^2_x \multimap \mathbf{N}^2_{xy+1})$. The corresponding BLL proof of $\vdash \mathbf{N}^2_x \otimes \mathbf{N}^2_y \multimap \mathbf{N}^2_{xy+1}$ represents the function g. □

We now show how to represent several auxiliary functions.

Proposition 6.8 *The functions $p_1(a) = 1 * a$ and $p_2(a) = 2 * a$ are representable by BLL proofs of $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{x+1}$.*

Proof : We consider p_1 , the representation of p_2 being analogous. Recall the BLL proof s_1 that was given in Figure 5.4, which represents the successor S_1, "Append 1". We modify that proof as follows. Consider the larger(left) branch above the lowest instance of $\multimap$L. Throughout that branch, except in $\mathbf{N}^2_x$, replace each instance of $!_{y<x}(\alpha(y) \multimap \alpha(y+1))$ and of $!_{y<x+1}(\alpha(y) \multimap \alpha(y+1))$ by $!_{y<x}(\alpha(y+1) \multimap \alpha(y+2))$, $\alpha(0)$ by $\alpha(1)$, and $\alpha(x)$ by $\alpha(x+1)$. The last rule in the branch is $\forall$L, but now with $\lambda y.\alpha(y+1)$ as T (instead of $\lambda y.\alpha(y)$, as in Figure 5.4). The branch now concludes with:

$$\mathbf{N}^2_x\,,\, !_{y<x}(\alpha(y+1) \multimap \alpha(y+2))\,,\, !_{y<x}(\alpha(y+1) \multimap \alpha(y+2))\,,\, \alpha(1) \vdash \alpha(x+1) \quad .$$

We can use $\multimap$L on this proof and on Axiom $\alpha(0) \vdash \alpha(0)$ to obtain:

$$\mathbf{N}^2_x, \alpha(0) \multimap \alpha(1), !_{y<x}(\alpha(y+1) \multimap \alpha(y+2)), !_{y<x}(\alpha(y+1) \multimap \alpha(y+2)), \alpha(0) \vdash \alpha(x+1)\,,$$

hence by Dereliction:

$$\mathbf{N}^2_x, !_{y<1}(\alpha(y) \multimap \alpha(y+1)), !_{y<x}(\alpha(y+1) \multimap \alpha(y+2)), !_{y<x}(\alpha(y+1) \multimap \alpha(y+2)), \alpha(0) \vdash \alpha(x+1)$$

and then by Contraction:

$$\mathbf{N}^2_x, !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), !_{y<x}(\alpha(y+1) \multimap \alpha(y+2)), \alpha(0) \vdash \alpha(x+1) \quad .$$

Another instance of $!_{y<1}(\alpha(y) \multimap \alpha(y+1))$ can be introduced by Weakening, and then Contraction can be used w.r.t. the other $!_{y<x}$ to conclude:

$$\mathbf{N}^2_x, !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), !_{y<x+1}(\alpha(y) \multimap \alpha(y+1)), \alpha(0) \vdash \alpha(x+1),$$

from which one obtains $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{x+1}$ by three applications of $\multimap$R, one application of $\forall$R, and another application of $\multimap$R. Note that the lambda term associated to this entire BLL proof is $\lambda e.\lambda f.\lambda g.\lambda a.e(f)(g)(f(a))$. □

The function reversing dyadic lists, *rev*, may be defined by iteration from p_1 and p_2:

$$\begin{aligned} rev(\varepsilon) &= \varepsilon \\ rev(b*1) &= p_1(rev(b)) \\ rev(b*2) &= p_2(rev(b)) \quad . \end{aligned}$$

The dyadic case of the Iteration Lemma (Lemma 6.2) and Proposition 6.8 readily yield:

Corollary 6.9 *The function rev reversing dyadic lists may be represented by a BLL proof of* $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_x$. □

Let us define, for any types A and B:

$$A \oplus B =_{\text{def}} \forall\, \alpha !_{x<1}(A \multimap \alpha(0)) \multimap !_{x<1}(B \multimap \alpha(0)) \multimap \alpha(0).$$

Note that erasing all resource information results in the usual definition of weak sum in system $\mathcal{F}$. Furthermore, as in system $\mathcal{F}$, in BLL one can derive:

$$A \vdash A \oplus B \qquad B \vdash A \oplus B \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C}$$

so that the associated lambda terms are type erasures of the canonical polymorphic left and right inclusions and of the polymorphic "definition by cases" in system $\mathcal{F}$, respectively.

Although the analog of Lemma 5.1 can be established for type $\mathbf{N}_0^2 \oplus \mathbf{N}_x^2 \oplus \mathbf{N}_x^2$, for our purposes it suffices to state the following proposition by means of lambda terms associated to BLL proofs. The reader should recall the notion of the lambda term $\bar{a}$ representing a dyadic list a as discussed in Example 5.2 as well as the discussion after Definition 5.2.

Proposition 6.10 *One can construct a BLL proof Bpd of* $\vdash \mathbf{N}_x^2 \multimap \mathbf{N}_0^2 \oplus \mathbf{N}_x^2 \oplus \mathbf{N}_x^2$ *whose associated lambda term bpd satisfies for any dyadic list a :*

$$\begin{aligned} bpd(\epsilon) &= left(\epsilon) \\ bpd(S_1(\bar{a})) &= mid(\bar{a}) \\ bpd(S_2(\bar{a})) &= right(\bar{a}) \end{aligned}$$

in the sense of convertibility, where $\bar{a}$ is the lambda term representing a, and where left, mid, and right are type erasures of the canonical polymorphic inclusions into the weak sum.

Proof: We use the dyadic case of the Iteration Lemma (Lemma 6.2). Let $\tau[z]$ be $\mathbf{N}_0^2 \oplus \mathbf{N}_z^2 \oplus \mathbf{N}_z^2$. When $z = 0$ take the BLL proof obtained by Cut from the canonical BLL proof representing the empty list ε and the canonical BLL proof "Left" $\mathbf{N}_0^2 \vdash \mathbf{N}_0^2 \oplus \mathbf{N}_0^2 \oplus \mathbf{N}_0^2$. Its associated lambda term is $left(\epsilon)$. In the iterative step for S_1, first consider the BLL proof of $\mathbf{N}_0^2 \oplus \mathbf{N}_z^2 \oplus \mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2$ defined "by Cases" from Axiom $\mathbf{N}_0^2 \vdash \mathbf{N}_{z+1}^2$, from the BLL proof s_1 of $\mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2$ given in Figure 5.4 (with z for x), and from the BLL proof s_2 of $\mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2$. Then cut this BLL proof defined by cases with the canonical BLL proof "Mid" of $\mathbf{N}_{z+1}^2 \vdash \mathbf{N}_0^2 \oplus \mathbf{N}_{z+1}^2 \oplus \mathbf{N}_{z+1}^2$. Now use $\multimap$R to obtain the BLL proof of $\vdash \tau[z] \multimap \tau[z+1]$; let M be its associated lambda term. In the iterative step for S_2, the construction is the same, but with "Right" instead of "Mid". Consider the BLL proof of $\vdash \tau[z] \multimap \tau[z+1]$ so obtained and let R be its associated lambda term. Apply the dyadic case of the Iteration Lemma to obtain the BLL proof *Bpd* of $\vdash \mathbf{N}_x \multimap \mathbf{N}_0^2 \oplus \mathbf{N}_x^2 \oplus \mathbf{N}_x^2$ and let *bpd* be its associated lambda term. One easily checks that $bpd(S_1(\bar{a})) = M(bpd(\bar{a})) = mid(\bar{a})$ and $bpd(S_2(\bar{a})) = R(bpd(\bar{a})) = right(\bar{a})$ for all binary lists a. □

As a step in representing limited recursion on notation, we represent cutoff on dyadic lists.

Proposition 6.11 *The cut-off function* $\lceil$, *where* $a\lceil n =$ *the list of first n digits of a dyadic list a, is representable by a BLL proof of* $\vdash \mathbf{N}_x \otimes \mathbf{N}_y^2 \multimap \mathbf{N}_x^2$.

Proof: Here one has to be somewhat crafty. We shall use tally iteration (Lemma 6.2), but more to the point, in the iteration we shall first reverse the list and we shall also keep track of the tail that will have been cut off. Let $\tau[z]$ be $\mathbf{N}_y^2 \multimap \mathbf{N}_z^2 \otimes \mathbf{N}_y^2$; the Iteration Lemma (tally case) will yield a BLL proof of $\vdash \mathbf{N}_x \multimap \mathbf{N}_y^2 \multimap \mathbf{N}_x^2 \otimes \mathbf{N}_y^2$. From this we consider the associated BLL proof of $\vdash \mathbf{N}_x \otimes \mathbf{N}_y^2 \multimap \mathbf{N}_x^2 \otimes \mathbf{N}_y^2$ and then take the left projection (available as a consequence of reuse) and thus obtain the desired BLL proof of $\vdash \mathbf{N}_x \otimes \mathbf{N}_y^2 \multimap \mathbf{N}_x^2$. Observe that because of the occurrences of resource variable x, iteration has to be on $\mathbf{N}_x$, i.e., it has to be tally iteration. Before we describe the base and the iteration step, let us give a sample computation which we will be representing. Consider the list 112. Then:

- for $n = 0$: $< \varepsilon, 211 >$
- for $n = 1$: $< 1, 21 >$
- for $n = 2$: $< 11, 2 >$
- for $n = 3$: $< 112, \varepsilon >$
- for $n \geq 4$: $< 112, \varepsilon >$.

Note that we can use *bpd* of the right component at stage k (cf. Proposition 6.10) to define the left component at stage $k+1$ by cases.

Now let us describe the base and the iteration step of this tally iteration. In the base, first take the BLL proof of $\vdash \mathbf{N}_0^2$ representing the empty list (cf. Example 5.2 for comparison). Second, take the BLL proof of $\vdash \mathbf{N}_y^2 \multimap \mathbf{N}_y^2$ that represents *reverse* (cf. Corollary 6.9) and cut with the canonical BLL proof of $\mathbf{N}_y^2 \multimap \mathbf{N}_y^2, \mathbf{N}_y^2 \vdash \mathbf{N}_y^2$ (built from two axioms by $\multimap$L) obtaining a BLL proof of $\mathbf{N}_y^2 \vdash \mathbf{N}_y^2$. Now use $\otimes$R on the two BLL proofs constructed and then $\multimap$R , yielding $\vdash \mathbf{N}_y^2 \multimap \mathbf{N}_0^2 \otimes \mathbf{N}_y^2$, i.e. $\tau[0]$. In the iteration step, it suffices to construct a BLL proof of $\tau[z] \vdash \tau[z+1]$, i.e., of $\mathbf{N}_y^2 \multimap \mathbf{N}_z^2 \otimes \mathbf{N}_y^2 \vdash \mathbf{N}_y^2 \multimap \mathbf{N}_{z+1}^2 \otimes \mathbf{N}_y^2$. This BLL proof will be obtained by $\multimap$R from a BLL proof of $\mathbf{N}_y^2 \multimap \mathbf{N}_z^2 \otimes \mathbf{N}_y^2, \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2 \otimes \mathbf{N}_y^2$, which in turn can be obtained by $\multimap$L from an instance of an axiom $\mathbf{N}_y^2 \vdash \mathbf{N}_y^2$ and from a certain BLL proof of $\mathbf{N}_z^2 \otimes \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2 \otimes \mathbf{N}_y^2$, which we now describe. First consider a BLL proof of $\mathbf{N}_0^2 \oplus \mathbf{N}_y^2 \oplus \mathbf{N}_y^2 \vdash \mathbf{N}_y^2$ defined by cases, where the left case is a waste of resources $\mathbf{N}_0^2 \vdash \mathbf{N}_y^2$ (i.e., an instance of an axiom) and the mid and right cases are the identity $\mathbf{N}_y^2 \vdash \mathbf{N}_y^2$ (again an instance of an axiom). Now use Cut with the BLL proof of $\mathbf{N}_y^2 \vdash \mathbf{N}_0^2 \oplus \mathbf{N}_y^2 \oplus \mathbf{N}_y^2$ corresponding to the BLL proof *Bpd* built in Proposition 6.10, and thus obtain a BLL proof of $\mathbf{N}_y^2 \vdash \mathbf{N}_y^2$. By the Reuse Lemma and weakening, obtain a BLL proof of $\mathbf{N}_z^2, \mathbf{N}_y^2 \vdash \mathbf{N}_y^2$, which we shall here call $\mathcal{R}$. On the other hand, consider the following BLL proof of $\mathbf{N}_z^2, \mathbf{N}_0^2 \oplus \mathbf{N}_y^2 \oplus \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2$ defined by cases. The left case is the BLL proof of $\mathbf{N}_z^2, \mathbf{N}_0^2 \vdash \mathbf{N}_{z+1}^2$ obtained by Reuse and weakening from waste of resources $\mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2$. The mid case is the BLL proof of $\mathbf{N}_z^2, \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2$ obtained by Reuse and weakening from the BLL proof of $\mathbf{N}_z^2 \vdash \mathbf{N}_{z+1}^2$ given in Figure 5.4 in order to represent "Append 1". The right case is similar, with "Append 2" instead. Now cut this BLL proof of $\mathbf{N}_z^2, \mathbf{N}_0^2 \oplus \mathbf{N}_y^2 \oplus \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2$ with the BLL proof of

$\mathbf{N}_y \vdash \mathbf{N}_0^2 \oplus \mathbf{N}_y^2 \oplus \mathbf{N}_y^2$ (corresponding to the BLL proof *Bpd* built in Proposition 6.10), and therefore obtain a BLL proof of $\mathbf{N}_z^2, \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2$ which we shall here call $\mathcal{L}$. Applying $\otimes$R to $\mathcal{L}$ and $\mathcal{R}$ yields:

$$\mathbf{N}_z^2\,,\, \mathbf{N}_z^2\,,\, \mathbf{N}_y^2\,,\, \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2 \otimes \mathbf{N}_y^2 \quad ,$$

whence by Reuse and contractions we obtain:

$$\mathbf{N}_z^2, \mathbf{N}_y^2 \vdash \mathbf{N}_{z+1}^2 \otimes \mathbf{N}_y^2 \ .$$

Now apply $\otimes$L. $\square$

We now represent in BLL the length function from dyadic lists to tally natural numbers:

Proposition 6.12 *The function lth from dyadic lists to tally natural numbers, lth(a) = the number of symbols in a, is representable by a BLL proof of* $\vdash \mathbf{N}_x^2 \multimap \mathbf{N}_x$.

Proof. Since $lth(\varepsilon) = 0, lth(a*i) = lth(a)+1, i = 1,2$, we may use dyadic iteration, see Lemma 6.2. Simply let $\tau[z]$ be $\mathbf{N}_z$, in the base take the cut-free proof of $\vdash \mathbf{N}_0$ representing numeral 0 (see Example 5.1 for comparison), and in either iteration step take the BLL proof of $\vdash \mathbf{N}_z \multimap \mathbf{N}_{z+1}$ representing the tally successor (obtained by $\multimap$R from Figure 5.2 with z for x). $\square$

Let us now consider limited recursion on notation, usually written as follows. Given functions g, h_1, h_2, and ℓ, the function f is defined by limited recursion on notation if it satisfies:

$$\begin{array}{rcl} f(a_1,\ldots,a_n,\varepsilon) & = & g(a_1,\ldots,a_n), \\ f(a_1,\ldots,a_n,b*1) & = & h_1(a_1,\ldots,a_n,b,f(a_1,\ldots,a_n,b)), \\ f(a_1,\ldots,a_n,b*2) & = & h_2(a_1,\ldots,a_n,b,f(a_1,\ldots,a_n,b)), \\ f(a_1,\ldots,a_n,b) & \leq & \ell(a_1,\ldots,a_n,b). \end{array}$$

Recalling the cut-off function $\lceil$ discussed in Proposition 6.11, it suffices to consider the following schema instead of limited recursion on notation:

$$\begin{array}{rcl} f(a_1,\ldots,a_n,\varepsilon) & = & g(a_1,\ldots,a_n), \\ f(a_1,\ldots,a_n,b*1) & = & h_1(a_1,\ldots,a_n,b,f(a_1,\ldots,a_n,b)) \lceil q(k_1,\ldots,k_n,m+1), \\ f(a_1,\ldots,a_n,b*2) & = & h_2(a_1,\ldots,a_n,b,f(a_1,\ldots,a_n,b)) \lceil q(k_1,\ldots,k_n,m+1), \end{array}$$

where q is a polynomial on tally natural numbers with tally natural numbers as coefficients, the length of dyadic list a_i is $k_i, 1 \leq i \leq n$, the length of dyadic list b is m , and the length of dyadic list $g(a_1,\ldots,a_n)$ is at most $q(k_1,\ldots,k_n,0)$. We shall refer to this schema as Schema $(*)$.

Proposition 6.13 *Let f be defined by Schema (*) from dyadic functions g, h_1, h_2, and from tally polynomial q. Let g be representable by a BLL proof of $\vdash \mathbf{N}^2_{y_1} \otimes \cdots \otimes \mathbf{N}^2_{y_n} \multimap \mathbf{N}^2_{q(y_1,\ldots,y_n,0)}$. Let $h_i, i = 1,2$, be representable by a BLL proof of $\vdash \mathbf{N}^2_{y_1} \otimes \cdots \otimes \mathbf{N}^2_{y_n} \otimes \mathbf{N}^2_x \otimes \mathbf{N}^2_u \multimap \mathbf{N}^2_{p_i(y_1,\ldots,y_n,x,u)}$, for some resource polynomials $p_i, i = 1,2$. Then f is representable by a BLL proof of $\vdash \mathbf{N}^2_{y_1} \otimes \ldots \otimes \mathbf{N}^2_{y_n} \otimes \mathbf{N}^2_x \multimap \mathbf{N}^2_{q(y_1,\ldots,y_n,x)}$.*

Proof : We suppress parameters for the sake of simplicity. (We may do so because of $\multimap$R and $\multimap$L.) We shall use dyadic iteration (see Lemma 6.2) with $\tau[z]$ being $\mathbf{N}^2_z \otimes \mathbf{N}^2_{q(z)}$, and then take the right projection. A representation of tally polynomial q was given in Corollary 6.6. In the base case, we take the BLL proof of $\vdash \mathbf{N}^2_0 \otimes \mathbf{N}^2_{q(0)}$ obtained by $\otimes$R from cut-free BLL proofs representing the empty list and the dyadic list g. In the iteration step, let us consider the case $i = 1$; the case $i = 2$ is completely analogous.

Consider the BLL proof of $\mathbf{N}_x, \mathbf{N}^2_y \vdash \mathbf{N}^2_x$ corresponding (by $\multimap$L and Cut) to the BLL proof built in Proposition 6.11 to represent the cut-off function $\lceil$, and let $x = q(z+1), y = p_1(z, q(z))$. Now we will use Cut, where the cut formula is $\mathbf{N}^2_{p_1(z,q(z))}$ and where the left premise of the Cut is obtained by letting $x = z$ and $u = q(z)$ in the BLL proof of $\mathbf{N}^2_x, \mathbf{N}^2_u \vdash \mathbf{N}^2_{p_1(x,u)}$ that corresponds (by $\multimap$L and Cut) to the given BLL proof representing the function h_1. Thus we obtain $\mathbf{N}_{q(z+1)}, \mathbf{N}^2_z, \mathbf{N}^2_{q(z)} \vdash \mathbf{N}^2_{q(z+1)}$. Now we Cut again, where the cut formula is $\mathbf{N}_{q(z+1)}$, and where the left premise of the Cut is the BLL proof obtained by letting $x = z+1$ in the BLL proof of $\vdash \mathbf{N}^2_x \multimap \mathbf{N}^2_{q(x)}$ that corresponds (by $\multimap$L and Cut) to a BLL proof of $\vdash \mathbf{N}_x \multimap \mathbf{N}_{q(x)}$ that represents the polynomial q, see Corollary 6.6. We thus have $\mathbf{N}_{z+1}, \mathbf{N}^2_z, \mathbf{N}^2_{q(z)} \vdash \mathbf{N}^2_{q(z+1)}$. This will now be the right premise of a Cut, where the cut formula is $\mathbf{N}_{z+1}$ and where the left premise is the BLL proof of $\mathbf{N}^2_z \vdash \mathbf{N}_{z+1}$, which is in turn obtained by Cut from the BLL proofs of $\mathbf{N}^2_z \vdash \mathbf{N}^2_{z+1}$ and of $\mathbf{N}^2_{z+1} \vdash \mathbf{N}_{z+1}$, themselves obtained by the obvious change of resource parameters (and by $\multimap$L and Cut) from the BLL proofs representing "Append 1" and *lth* (see Figure 5.4 and Proposition 6.12). In this way we reach $\mathbf{N}^2_z, \mathbf{N}^2_z, \mathbf{N}^2_{q(z)} \vdash \mathbf{N}^2_{q(z+1)}$. Now apply $\otimes$R, the other premise being the BLL proof of $\mathbf{N}^2_z \vdash \mathbf{N}^2_{z+1}$, which corresponds (by $\multimap$L and Cut) to the BLL proof representing "Append 1". We thus reach $\mathbf{N}^2_z, \mathbf{N}^2_z, \mathbf{N}^2_z, \mathbf{N}^2_{q(z)} \vdash \mathbf{N}^2_{z+1} \otimes \mathbf{N}^2_{q(z+1)}$. The Reuse Lemma (Lemma 6.3) allows contractions on $\mathbf{N}^2_z$, so we obtain $\mathbf{N}^2_z, \mathbf{N}^2_{q(z)} \vdash \mathbf{N}^2_{z+1} \otimes \mathbf{N}^2_{q(z+1)}$. Note that the lambda term associated to the BLL proof we built is the term (informally written as)

$$< S_1(\mathbf{r}(b)) \, , \, h_1(\mathbf{r}(b), c) \lceil q(lth(S_1(\mathbf{r}(b)))) > \ .$$

We complete the construction by applying $\otimes$L and $\multimap$R $\square$

The reader will readily check that among the first consequences of Proposition 6.13 are BLL representations of the numerical functions $n+1$, $2n$, and $n \dot{-} 1$ in dyadic notation. Combining these with Proposition 6.7 expresses Cobham's initial functions.

We can now conclude:

Proof of Theorem 6.1. Lemmas 6.1 and 6.2, Propositions 6.3–6.13, and Corollaries 6.6 and 6.9 yield that the class of dyadic functions representable in BLL includes Cobham's initial functions and is closed under limited recursion on notation. By using Cut (and $\multimap$L and $\multimap$R), the class is also closed under general substitution. Hence this class contains all polynomial time computable functions. □

Although the Cobham-style characterization is not known for the tally case, the tally analog of Theorem 6.1 may be obtained by using Theorem 6.1 and the representation of the length function given in Proposition 6.12.

Theorem 6.14 *Every function from tally natural numbers to tally natural numbers computable in polynomial time in tally length can be represented in Bounded Linear Logic by a proof of* $\vdash \mathbf{N}_x \multimap \mathbf{N}_{p(x)}$, *for some resource polynomial* p.

Proof: Let F be a function from tally natural numbers to tally natural numbers computable in polynomial time in tally length. Every tally list is a dyadic list (in which all symbols are 1); this defines a function D from tally natural numbers to dyadic lists such that $D(\varepsilon) = \varepsilon, D(a * 1) = D(a) * 1$. The function D is representable by a BLL proof of $\vdash \mathbf{N}_x \multimap \mathbf{N}_x^2$, obtained by tally iteration. Recall the function lth from dyadic lists to tally natural numbers discussed in Proposition 6.12. The function $D \circ F \circ lth$ is clearly polynomial time computable in dyadic length. So by Theorem 6.1, $D \circ F \circ lth$ is representable by a BLL proof of, say, $\vdash \mathbf{N}_x^2 \multimap \mathbf{N}_{p(x)}^2$. But then $lth \circ D \circ F \circ lth \circ D$ is representable by a BLL proof of $\vdash \mathbf{N}_x \multimap \mathbf{N}_{p(x)}$ obtained by $\multimap$L, Cut, and $\multimap$R from BLL proofs representing D, $D \circ F \circ lth$ and lth. Since $lth \circ D =$ identity, it is the case that $F = lth \circ D \circ F \circ lth \circ D$, and hence F is representable. □

A similar argument may be used to derive the converse of Theorem 6.14, Theorem 5.4, from Theorem 5.3.

7 Acknowledgements

This work was begun during a visit by the first and third authors to the Department of Mathematics of the University of Pennsylvania in the fall semester, 1987. They would both like to thank the Department for its hospitality. Girard's visit to the University of Pennsylvania was partially supported by NSF Grant CCR-87-05596. Scedrov's research is partially supported by ONR contract N00014-88K-0635, NSF Grant CCR-87-05596, and a Young Faculty Award of the Natural Sciences Association of the University of Pennsylvania. Scedrov thanks John C. Mitchell, the Computer Science Department, and the Center for the Study of Language and Information at Stanford University for their hospitality during his 1989-90 Sabbatical leave. Scott's research is supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada and by an FCAR(Quebec)

team grant.

References

[1] S. Abramsky. Computational Interpretation of Linear Logic, *Theoretical Computer Science* , to appear.

[2] S. R. Buss. *Bounded Arithmetic*, Bibliopolis, 1986 (revised version of 1985 Princeton Ph.D. thesis.)

[3] S. R. Buss. The polynomial hierarchy and intuitionistic Bounded Arithmetic, in: *Structure in Complexity* , Springer LNCS, **223** (1986), pp. 77-103.

[4] S. R. Buss and P. J. Scott, eds. *Feasible Mathematics* , Proceedings of an MSI Workshop, Ithaca, NY (June, 1989). Progress in Computer Science and Applied Logic (V.9), Birkhauser, 1990.

[5] A. Cobham. The intrinsic computational difficulty of functions, in: *Proc. of the 1964 International Congress for Logic, Methodology and Philosophy of Science* , Y. Bar-Hillel, ed. North-Holland Publishing Co., Amsterdam, 1964, pp. 24-30.

[6] S. A. Cook. Feasibly constructive proofs and the propositional calculus, in Proc. 7th Annual ACM Symp. on Theory of Computing (1975), pp. 83-97.

[7] S. A. Cook and B. M. Kapron. Characterizations of the Basic Feasible Functionals of Finite Type, in: [4], pp. 71-96.

[8] S. A. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic, Tech. Report 210/88, Dept. of Computer Science, Univ. of Toronto, June, 1988. (Extended Abstract, in: Proc. 21st ACM Symp. on Theory of Computation, pp. 107-112 (1989).)

[9] J. Crossley and A. Nerode. *Combinatorial Functors*, Springer-Verlag, 1974.

[10] V. Danos. La logique linéaire appliquée à l'étude de divers processus de normalisation et principalement du λ-calcul, Thèse de doctorat, U. Paris VII, April 1990.

[11] V. Danos and L. Regnier. The Structure of Multiplicatives, *Arch. Math. Logic* (1989) **28**, pp. 181-203.

[12] J-Y. Girard. Une Extension de l'Interprétation de Gödel a l'Analyse, et Son Application a l'Elimination des Coupures dans l'Analyse et la Théorie des Types, in: J. E. Fenstad, ed. *Proc. 2nd Scandanavian Logic Symposium* , North-Holland (1971), pp. 63-92.

[13] J-Y. Girard. Linear Logic, *Theoretical Computer Science* **50**, 1987, pp. 1-102.

[14] J-Y. Girard. Multiplicatives, *Rend. Sem. Matem. (Torino)* , 1987 (Special Issue: Logic and Computer Science, New Trends and Applications, Oct. 1986), pp. 11-33.

[15] J-Y. Girard. Quantifiers in Linear Logic II, Prépublications Paris 7 Logique, No. 19, January, 1991.

[16] J-Y. Girard. Π^1_2 Logic, Part 1: Dilators, *Ann. Math. Logic* **21** (1981), pp. 75-219.

[17] J-Y. Girard. Towards a Geometry of Interaction, in: *Categories in Computer Science and Logic*, ed. by J.W. Gray and A. Scedrov, *Contemp. Math*, **92**, AMS, 1989, pp. 69-108.

[18] J-Y.Girard. Geometry of Interaction I: Interpretation of system F, in: *Logic Colloquium '88* , ed. R. Ferro, et al. North-Holland, 1989, pp. 221-260.

[19] J-Y.Girard. Geometry of Interaction 2: Deadlock-free Algorithms. COLOG-88 (P. Martin-Löf, G. Mints, eds.) Springer Lecture Notes in Computer Science, **417**, 1990, pp. 76-93.

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

[21] J-Y. Girard, A. Scedrov, and P.J. Scott. Bounded Linear Logic: A Modular Approach to Polynomial Time Computability (Extended Abstract), in: [4], pp. 195-209.

[22] Y. Gurevich. Toward logic tailored for computational complexity, in: *Computation and Proof Theory* , Springer LNM **1104**, 1984, pp. 175-216.

[23] Y. Gurevich. Logic and the challenge of computer science, in: *Current Trends in Theoretical Computer Science* , E. Börger, ed., Computer Science Press, 1988, pp. 1-57.

[24] N. Immerman. Relational queries computable in polynomial time, *Information and Control* **68** (1986), pp. 86-104.

[25] Ph. G. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: tools and a case study. *Proc. 9th ACM Symp. on Principles of Database Systems* , 1990, pp. 61-71.

[26] J-L. Krivine. Opérateurs de mise en mémoire et traduction de Gödel, *Arch. Math. Logic* **30** (1990), pp. 241-267.

[27] A. Nerode, J. B. Remmel, and A. Scedrov. Polynomially Graded Logic I: A Graded Version of System T, in: *Fourth IEEE Symposium on Logic in Computer Science* , Pacific Grove, CA, June, 1989, pp. 375-385.

[28] H.E. Rose. *Subrecursion.* Oxford Logic Guides **9** , 1984.

[29] A. Scedrov. A brief guide to linear logic. *Bulletin EATCS* , **41** (June, 1990), pp. 154-165.

[30] G.J.Tourlakis. *Computability.* Reston (Prentice-Hall), 1984.

[31] M. Y. Vardi. The complexity of relational query languages. *Proc. 14th ACM Symp. on Theory of Computing* , 1982, pp. 137-146.

[32] K.Wagner and G. Wechsung. *Computational Complexity* . D. Reidel, 1986.

Appendix A: Normalization in BLL sequent calculus

We outline a similar analysis to Section 4, based on sequents. The basic definitions are as follows. In BLL sequent calculus we have:

- An instance of the cut rule is *boxed* when it is above a rule S!.
- A cut is *irreducible* if it is boxed or if its left premise is S! with a nonempty context and its right premise is either !W, !C, !D, or S!.
- A BLL sequent calculus proof is *irreducible* if it contains only irreducible cuts (if any).
- A BLL sequent is *accessible* if each negative occurrence of a universal quantifier or a bounded exclamation mark is nested within a positive occurrence of a bounded exclamation mark.

It is understood that none of the reduction steps given below apply to irreducible cuts.

Axiom Reductions

AL

$$\frac{A \vdash A' \quad \begin{array}{c}\vdots\ \rho\\ \Gamma, A' \vdash B\end{array}}{\Gamma, A \vdash B} \quad \text{reduces to} \quad \begin{array}{c}\vdots\ \rho'\\ \Gamma, A \vdash B\end{array}$$

(see the remark on waste of resources in BLL sequent calculus proofs in section 3.3).

AR

$$\frac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad A \vdash A'}{\Gamma \vdash A'} \quad \text{reduces to} \quad \begin{array}{c}\vdots\ \rho'\\ \Gamma \vdash A'\end{array}$$

(again, see the remark on waste of resources in BLL sequent calculus proofs in section 3.3.)

Symmetric Reductions

In addition to the reduction steps S$\otimes$, S$\multimap$, and S$\forall$ described in section 2.2, we stipulate the following four steps in which the cut formula begins with a bounded exclamation mark and the left premise of a Cut is obtained by an instance of the S! rule in which the context is empty, i.e., there are no formulas to the left of the $\vdash$:

SS!D

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \vdash A\end{array}}{\vdash !_{x<1+w}A} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, A[x:=0] \vdash B\end{array}}{\Delta, !_{x<1+w}A \vdash B}}{\Delta \vdash B} \quad \text{reduces to} \quad \dfrac{\begin{array}{c}\vdots\ \rho(x:=0)\\ \vdash A[x:=0]\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A[x:=0] \vdash B\end{array}}{\Delta \vdash B}$$

SS!C

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \vdash A\end{array}}{\vdash !_{x<p+q+w}A} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, !_{x<p}A, !_{y<q}A[x:=p+y] \vdash B\end{array}}{\Delta, !_{x<p+q+w}A \vdash B}}{\Delta \vdash B} \quad \text{reduces to}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \vdash A\end{array}}{\vdash !_{x<p}A} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\ \rho(x:=p+y)\\ \vdash A[x:=p+y]\end{array}}{\vdash !_{y<q+w}A[x:=p+y]} \quad \begin{array}{c}\vdots\ \omega'\\ \Delta, !_{x<p}A, !_{y<q+w}A[x:=p+y] \vdash B\end{array}}{\Delta, !_{x<p}A \vdash B}}{\Delta \vdash B}$$

where ω' is obtained from ω as a special case of the remark on waste of resources in BLL sequent calculus proofs, section 3.3 .

SS!W

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \vdash A\end{array}}{\vdash !_{x<p}A} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta \vdash B\end{array}}{\Delta, !_{x<p}A \vdash B}}{\Delta \vdash B} \quad \text{reduces to} \quad \begin{array}{c}\vdots\ \omega\\ \Delta \vdash B\end{array}$$

SS!S!

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \vdash A_i\end{array}}{\vdash !_{y_i<v_i(p)+w_i}A_i} \qquad \dfrac{\begin{array}{c}\vdots\ \omega\\ \ldots, !_{z<q_j(x)}A_j[y_j := v_j(x)+z], \ldots, !_{z<q_i(x)}A_i[y_i := v_i(x)+z], \ldots \vdash B\end{array}}{\ldots, !_{y_j<v_j(p)+w_j}A_j, \ldots, !_{y_i<v_i(p)+w_i}A_i, \ldots \vdash !_{x<p}B}}{\ldots, !_{y_j<v_j(p)+w_j}A_j, \ldots \vdash !_{x<p}B}$$

reduces to

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho(y_i := v_i(x)+z)\\ \vdash A_i[y_i := v_i(x)+z]\end{array}}{\vdash !_{z<q_i(x)}A_i[y_i := v_i(x)+z]} \qquad \begin{array}{c}\vdots\ \omega\\ \ldots, !_{z<q_j(x)}A_j[y_j := v_j(x)+z], \ldots, !_{z<q_i(x)}A_i[y_i := v_i(x)+z], \ldots \vdash B\end{array}}{\ldots, !_{z<q_j(x)}A_j[y_j := v_j(x)+z], \ldots \vdash B}}{\ldots, !_{y_j<v_j(p)+w_j}A_j, \ldots \vdash !_{x<p}B}$$

Commutative Reductions

In addition to the reduction steps CL⊗L, CL⊸L, CL∀L, CR⊗R, CR⊸R, CR∀R, CR⊗L, CR⊸L, and CR∀L described in Sections 2.2.3, 2.2.5, we stipulate the following commutative reduction steps:

CL!W

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array}}{\Gamma, !_{x<w}C \vdash A} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, !_{x<w}C \vdash B} \qquad \text{reduces to} \qquad \dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta \vdash B}}{\Gamma, \Delta, !_{x<w}C \vdash B}$$

CL!D

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma, C[x := 0] \vdash A\end{array}}{\Gamma, !_{x<1+w}C \vdash A} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, !_{x<1+w}C \vdash B}$$

$$\text{reduces to} \qquad \dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma, C[x := 0] \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, C[x := 0] \vdash B}}{\Gamma, \Delta, !_{x<1+w}C \vdash B}$$

CL!C

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma, !_{x<p}C, !_{y<q}C[x := p+y] \vdash A\end{array}}{\Gamma, !_{x<p+q+w}C \vdash A} \qquad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta, !_{x<p+q+w}C \vdash B}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma, !_{x<p}C, !_{y<q}C[x := p+y] \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, !_{x<p}C\,,\, !_{y<q}C[x := p+y]\,,\, \Delta \vdash B}}{\Gamma, \Delta, !_{x<p+q+w}C \vdash B}$$

CR!W

$$\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Delta, A, !_{x<w}C \vdash B}}{\Gamma, \Delta, !_{x<w}C \vdash B}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A \vdash B\end{array}}{\Gamma, \Delta \vdash B}}{\Gamma, \Delta, !_{x<w}C \vdash B}$$

CR!D

$$\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, A, C[x := 0] \vdash B\end{array}}{\Delta, A, !_{x<1+w}C \vdash B}}{\Gamma, \Delta, !_{x<1+w}C \vdash B}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A, C[x := 0] \vdash B\end{array}}{\Gamma, \Delta, C[x := 0] \vdash B}}{\Gamma, \Delta, !_{x<1+w}C \vdash B}$$

CR!C

$$\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \dfrac{\begin{array}{c}\vdots\ \omega\\ \Delta, A, !_{x<p}C, !_{y<q}C[x := p+y] \vdash B\end{array}}{\Delta, A, !_{x<p+q+w}C \vdash B}}{\Gamma, \Delta, !_{x<p+q+w}C \vdash B}$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \rho\\ \Gamma \vdash A\end{array} \quad \begin{array}{c}\vdots\ \omega\\ \Delta, A, !_{x<p}C, !_{y<q}C[x := p+y] \vdash B\end{array}}{\Gamma, \Delta, !_{x<p}C, !_{y<q}C[x := p+y] \vdash B}}{\Gamma, \Delta, !_{x<p+q+w}C \vdash B}$$

The analog of Theorem 4.4 can be extended to BLL sequent calculus by using cut-size (cf. Section 2.2), in which case all reductions terminate in at most $(\|\nu\|)^3$

steps.

Jean-Yves Girard
Équipe de Logique
UA 753 du CNRS
Mathématiques,
t. 45-55
Univ.de Paris 7
2 Place Jussieu
75251 Paris Cedex 05
France
Internet:
girard@margaux.inria.fr

Andre Scedrov
Dept. of Mathematics
Univ. of Pennsylvania
Philadelphia, PA
19104-6395
U.S.A.
Internet:
andre@cis.upenn.edu

Philip J. Scott
Dept. of Mathematics
Univ. of Ottawa
585 King Edward
Ottawa, Ont.
Canada K1N6N5
Internet:
scpsg@acadvm1.uottawa.ca