A Logical Calculus for Polynomial-time Realizability

J.N. Crossley
Monash University

G.L. Mathai
Monash University

R.A.G. Seely*
McGill University &
John Abbott College

February 14, 1991

Abstract

A logical calculus, not unlike Gentzen's sequent calculus for intuitionist logic, is described which is sound for polynomial-time realizability as defined by Crossley and Remmel. The sequent calculus admits cut elimination, thus giving a decision procedure for the propositional fragment.

0 Introduction

In [4], a restricted notion of realizability is introduced, a special case of which is polynomial-time realizability: this is like Kleene's original realizability, save for three features. First, closed atomic formulæ are realized by realizers that give a measure of the resources required to establish the formula, unlike Kleene's system which only reflects the fact that the formula is provable. Second, open formulæ are treated as the corresponding closed formulæ with all free variables universally quantified simultaneously. (There is a difference between the quantifiers $\forall\langle\xi,\eta\rangle$ and $\forall\xi\forall\eta$.) And third, the realizers code polynomial-time ("p-time") functions, rather than arbitrary recursive functions.

In [4], only the p-time realizability of single formulæ is discussed—in this paper we shall extend these notions to logical rules, to give a calculus that is sound for p-time realizability. This will be a sequent calculus, much like Gentzen's formulation for intuitionist logic, with three main points of difference. First, a sequent of the form $A, B \longrightarrow C$ is interpreted as if it were a formula $A \supset (B \supset C)$, rather than $(A \wedge B) \supset C$ (which would be the Gentzen interpretation). As was shown in [4], these are not equivalent. Indeed, if $\Vdash (A \wedge B) \supset C$ then $\Vdash A \supset (B \supset C)$, but not conversely. ($\Vdash A$ means "A is realizable"; similarly $e \Vdash A$ means "e realizes A".)

Second, among the structure rules we keep thinning, but drop exchange and contraction, (roughly as in Girard's linear logic [7]). Again, it was shown in [4] that one could have $\Vdash A \supset (B \supset C)$ without having $\Vdash B \supset (A \supset C)$. For example, in [4] there is an example that occurs frequently. In the notation of this paper, it can be given in these three flavours:

$$(x:),(y:) \longrightarrow x^{y+1} = x^y \cdot x \tag{1}$$

$$(y:),(x:) \longrightarrow x^{y+1} = x^y \cdot x \tag{2}$$

$$(x,y:) \longrightarrow x^{y+1} = x^y \cdot x \tag{3}$$

(These examples may also be presented as quantified formulæ.) Of course, there is no way that Examples 1, 3 can be realizable in polynomial time, but Example 2 *is* realizable in polynomial time, essentially because

*Partially supported by grants from Le Fonds F.C.A.R. Québec, N.S.E.R.C. Canada, Australian Research Council (Grant no. 20.155.009.288), and the Departments of Mathematics and Computer Science, Monash University. Thanks are particularly due to John and Gerald, and to Dr. and Mrs. L.E. Clemens and their family, for the hospitality the third author received during his visits to Monash.

initially one only need produce in polynomial time (in $|y|$) a *program* which gives the instructions to read the x and compute and compare the given *polynomials*; (at the time the expressions are actually computed they are polynomials since by then y has been fixed.) (There is a need to be careful about the question of realizing the terms here, since exponentiation, as a two-variable term, is not itself realizable—again we refer the reader to [4].)

In this vein, we should remark that the class of functions we shall be able to represent is an extension of the usual class of polynomial-time functions, which we may call "polynomially curried functions". The prototypical example is exponentiation: $exp(x, y) = x^y$ is not p-time as a function of two variables, but if suitably curried (*viz.* as $\lambda y \lambda x.x^y$) then it is p-time at each step of the way. (Note that being polynomially curried depends on the order in which the variables are taken.) These notions coincide for one-variable functions, and so by coding strings of variables as tuples, we recover the complete class of p-time functions as part of the class of polynomially curried functions. We expect to present more details in a separate paper.

An aside for the category-theorist: our structure is essentially a closed multicategory [9, 10] with finite products and coproducts: $\supset$ gives the internal hom but this is not a hom for the product structure given by $\wedge$. We do not have a tensor $\otimes$—the comma in the sequent notation takes that rôle—but if we did, it would not be symmetric. We can find no operational significance for any such tensor—in particular, what might $\Gamma \longrightarrow A \otimes B$ signify?—other than as a notational alternative to the comma. (So the only sequents $\Gamma \longrightarrow A \otimes B$ would be those obtained from $\Gamma_1 \longrightarrow A$, $\Gamma_2 \longrightarrow B$ by "tensoring", where $\Gamma = \Gamma_1\Gamma_2$.) However, it is stronger than ordinary conjunction (which is, of course, symmetric): $\|\!\vdash A, B \longrightarrow A \wedge B$ is valid (take $C = A \wedge B$ in the result quoted above). (See [13] for some details concerning categorical structure for a related system.)

Third, we extend the usual notion of "sequent" by the addition of variable declarations. A variable declaration $(x :)$ delimits the scope of a free variable x in a sequent, and gives finer control over the functional aspect of realizing open formulæ. For example, suppose B is a closed formula and $A(x)$ contains exactly x as free variable. It is *not* the case that $\|\!\vdash B \supset A(x)$ implies $\|\!\vdash B \supset \forall\xi A(\xi)$, according to the definitions of [4]. (Note that we use lower case Greek letters to indicate bound variables.) For if $e \|\!\vdash B \supset A(x)$, then e is the code of a function which, for each standard numeral k, gives $e(k) \|\!\vdash B \supset A(k)$, which in turn is (the code of) a function which for each $b \|\!\vdash B$ gives $e(k)(b) \|\!\vdash A(k)$. The point is that the degree of (the polynomial bounding the run-time of) $\lambda b.e(k)(b)$ depends on k, and so therein lies the problem in finding the realizer for $B \supset \forall\xi A(\xi) : f \|\!\vdash B \supset \forall\xi A(\xi)$ would require (in the above notation) $f(b)(k) \|\!\vdash A(k)$; but $f(b)(k) = e(k)(b)$ cannot be solved for p-time functions. On the other hand, if we can shift the functional dependence so that k only enters when wanted then this problem will disappear. The (valid) rule corresponding to our discussion then becomes

$$(\forall\ \mathrm{R}) \qquad \frac{\Gamma, (x :) \longrightarrow A(x)}{\Gamma \longrightarrow \forall\xi A(\xi)}$$

where $(x :)$ indicates the beginning of the scope of the variable x, (its end being the end of the sequent), Γ being a finite sequence of formulæ (and possibly other variable declarations). Such variable declarations should be thought of as the eponymous statements in computer programmes; as in programming, a variable may only appear locally, and so we wish to capture that locality in the functionality of our realizers. In the above rule, both lines in fact are realized by an e, which is (the code of) a function which to any $\gamma \|\!\vdash \Gamma$ gives (the code of) a function $e(\gamma)$, which to any standard k gives $e(\gamma)(k) \|\!\vdash A(k)$. $(\Gamma, (x :) \longrightarrow A$ becomes in effect $\Gamma \supset (x \supset A)$; since one can think of $\forall\xi A(\xi)$ as $(x \supset A)$, it is easy to see this rule becomes a triviality.)

There is an elaboration of this idea that we shall also need, involving markers for variable declarations that have been "quantified out" (as is $(x :)$ in $(\forall$ R) above). We call these markers "shadow declarations"; they have no effect on the realizers (just ignore them), but are necessary for correct management of the cut rule.

A final remark: the sequent calculus will be more a calculus for p-time realizability than a logical system, as some rules make explicit reference not only to the logical form of the sequents that form the premises of the rules, but also to the nature of the realizers of those sequents. A key case in point is the $(\exists$ L)-rule: traditionally this rule is (roughly) of this form:

$$(\exists\ \mathrm{L}) \qquad \frac{A(x)\Delta \longrightarrow B}{\exists\xi A(\xi)\Delta \longrightarrow B}$$

with the restriction that the free variable x (that becomes quantified) does not occur freely in B nor in any other assumption, *i.e.* nor in Δ. The intention is that x only occur in A, not in any other aspect of the proof of B. For us, this rule omits a crucial other way in which x may affect matters, namely the efficiency of the algorithm. A realizer e of the top sequent is (the code of) a function which to any k assigns (the code of) a function $e(k)$ which to any $a_k \Vdash A(k)$ assigns (the code of) a function $e(k)(a_k)$ which realizes $\Delta \longrightarrow B$. If any k is to be "as good as any other", then the runtime degree of $e(k)$ must not depend on k, or equivalently, must be bound for all k. So k enters the story only in that $e(k)$ is defined on realizers $a_k \Vdash A(k)$, but not in the degree (or "efficiency") of the polynomial bounding the run-time of $\lambda a.e(k)(a)$. Hence e is p-time in the pair of variables $\langle k, a_k\rangle$, given simultaneously. We should note, however, that k may play a rôle in the rest of the sequent: the runtime degree of $e(k)(a_k)$, as a function of $\delta \Vdash \Delta$ may well depend on both k and a_k. (This is, of course, because it may depend on a_k; the rôle of k here is as an "index" placing a_k properly.) The point, then, is that the ($\exists$ L)-rule should only apply to realizers of sequents that have this property, and so the rule is not only dependent on the logical form of the sequent, but also on the particular "derivation tree" in which it occurs. (This restriction is automatic in the "decorated system" of [13], but that system is more restrictive than that of this paper in general.)

(We shall call such declarations (x :) "tame" with respect to the realizer e.)

Restrictions in the same vein are made to various other of the logical and non-logical rules and axioms of our systems; (see especially the treatment of (*PIND*), where we have replaced a restriction based on the logical structure of a formula with a restriction based on properties of the realizers.)

1 Review of polynomial-time realizability

We review informally the basic definitions from [4]—for full details, the reader should refer to that paper.

We assume a formal language $\mathcal{L}$ based on that in [1], with constants including 0, function letters including $+$, $\cdot$, *exp* (exponentiation), S (successor), $\sharp$ (smash product), $\lfloor\ \rfloor$ (greatest integer), and $|\ |$ (length), and predicate letters including $=$ (equality). Realizers are taken as (codes of) pairs $e = \langle e_m, e_t\rangle$, where e_m is (the code of) a Turing machine and e_t is (the code of) a polynomial: e acts as e_m except that it turns off after $e_t(|\mathbf{x}|)$ steps, where $|\mathbf{x}|$ is the length of the input. In the following, "function" means such a realizer, unless otherwise specified. (This includes constants, as functions of 0 variables.)

We shall write $e(f)$ to denote $\{e_m\}(f)$, $\{\ \}$ denoting the Kleene bracket.

For some of the function symbols, in particular, $+$, $\cdot$, but not *exp*, there is associated a realizer e_f with the function symbol f, and likewise for some of the predicate symbols, including $=$, a realizer e_P is associated with P. Terms are realized by this definition:

Definition 1 *1. If t is a constant c and $e = e_c$ is the realizer for c then $e \Vdash t$.*

2. If t is a variable x_i, then $\langle id_i, |\ |\rangle \Vdash t$ (where id_i is the i^{th} projection function and $|\ |$ is the appropriate length-of-input function).

3. If $t = f(t_1, \ldots, t_n)$ is a closed term, $e_i \Vdash t_i$ for $i = 1, \ldots, n$, e_f is the realizer of f, then $e_f(e_1, \ldots, e_n) \Vdash t$.

4. If $t = f(t_1, \ldots, t_n)$ contains free variables $x_1, \ldots, x_m$, then $e \Vdash t$ iff e is (the code of) a function that, for standard numerals $k_1, \ldots, k_m$, gives $e(k_1, \ldots, k_m) =$ (intended value of) $t(k_1, \ldots, k_m)$, within the specified time (given by e_t).

(Some liberties have been taken here for simplicity—a more precise definition may be found in [4].)

We make the convention that if $\mathbf{y} = y_1, \ldots, y_n$ is a string of arguments, and if $n = 0$, then $f(\mathbf{y}) = f$; (recall this $= \{f\}(\mathbf{y})$.)

Observe that if $e \Vdash t(x)$ and $e' \Vdash t'(y)$, then $e \circ e'(y)) \Vdash t(t'(y))$ since composition of functions is associative and e, e' are made up by composition according to clause 3 above.

Each natural number will be regarded as having a normal form which will be the usual binary string representation. $k \in \mathcal{N}$ means k is such a binary string.

Definition 2

- *Closed atomic formulæ are realized as follows: if $e_i \Vdash t_i$ (for $i = 1, \ldots, n$) and e_P is the realizer of P, if $P(t_1, \ldots, t_n)$ is true and if e_P computes that $P(t_1, \ldots, t_n)$ is true using $e_1, \ldots, e_n$, then we set $e_P(e_1, \ldots, e_n) \Vdash P(t_1, \ldots, t_n)$.*

- *Next, we continue as in Kleene's original inductive definition:*

  $e \Vdash A$ in the following situations:

  - *If A is closed:*

    1. *$\perp$ is never realized.*
    2. *A is $B \wedge C$, $e = \langle e_0, e_1 \rangle$, $e_0 \Vdash B$ and $e_1 \Vdash C$.*
    3. *A is $B \vee C$, $e = \langle e_0, e_1 \rangle$, and either e_0 is 0 and $e_1 \Vdash B$ or e_0 is not 0 and $e_1 \Vdash C$.*
    4. *A is $B \supset C$, and, for all f, if $f \Vdash B$ then $e(f)$ is defined and $e(f) \Vdash C$.*
    5. *A is $\forall \xi \leq |t|\, B(\xi)$, $e = \langle e_1, \ldots, e_{|\underline{t}|} \rangle$, and $e_i \Vdash B(x := i)$, for $i = 1, \ldots, |t|$. (Note that here t is closed, and so determines a standard numeral $\underline{t}$.)*
    6. *A is $\forall \xi B(\xi)$, and for all $k \in \mathcal{N}$, $e(k) \Vdash B(x := k)$.*
    7. *A is $\exists \xi B(\xi)$, $e = \langle e_0, e_1 \rangle$, e_0 is a standard numeral $\in \mathcal{N}$ and $e_1 \Vdash B(x := e_0)$, (where $B(x := e_0)$ denotes the substitution of e_0 for x in B.)*

  - *If A is open with free variables exactly $x_1, \ldots, x_n$ and for all $m_1, \ldots, m_n \in \mathcal{N}$,*

$$e(m_1, \ldots, m_n) \Vdash A(x_1 := m_1, \ldots, x_n := m_n).$$

(Note above that $\forall \xi \leq |t|\, A(\xi)$ is well formed only if ξ does not occur in t.)

2 The sequent calculus

In the logical calculus we shall develop, it will be convenient to assume that all formulæ are "homogeneous" in their occurrences of free variables: in forming a compound formula $A \diamond B$, for any connective $\diamond$, we shall suppose A and B have exactly the same free variables, which then also occur in $A \diamond B$. This may be done without loss in expressive power by a liberal use of "dummy free variables"; perhaps the simplest technical way to do this is to add new function symbols to $\mathcal{L}$ corresponding to projections (see [12] for example, where first order logic with equality is handled this way). Note that in quantifying a formula, exactly one free variable disappears—*viz.* the one quantified.

Definition 3 *A sequent $\Gamma \longrightarrow A$ is a pair, where Γ is a finite sequence of formulæ and variable declarations, and A is a formula, where a variable declaration $(x :)$ must precede all formulæ in which x appears. Furthermore, a variable may only be declared once within the sequent.*

Remarks

1. There may be formulæ within the scope of x in which x does not appear—such formulæ will be treated as if they have 'dummy' occurrences of x.

2. We can declare several variables simultaneously via pairing:

   $(\mathbf{x}:)$ will mean $(\langle x_1, \ldots, x_n\rangle:)$, where $\mathbf{x} = x_1, \ldots, x_n$. (Such pairing is treated as unordered.) These variables can also be declared sequentially: $(x_1:), (x_2:), \ldots, (x_n:)$. These forms of declaration are not equivalent.

3. We shall also "decorate" sequents with "shadow declarations" $[x:]$. These are needed for the cut rule (only), but are ignored as far as the realizers are concerned. (In other words, realizability for such "decorated" sequents is just the realizability of the corresponding "undecorated" sequent, with all "shadow declarations" removed.) Shadow declarations in some respects resemble the "discharged assumptions" of natural deduction. Their use is to control the "interleaving" of formulæ in the cut rule—see below. Shadow declarations are introduced by the rules ($|\forall|$ R), ($\forall$ R), and ($\exists$ L). These are the rules that "quantify out" a variable declaration.

Definition 4 *Our theory consists of all sequents generated* [1] *from the following axioms by the following rules:*

Logical Axioms

(L1) $(\mathbf{x}:), A \longrightarrow A$ *(where* $\mathbf{x}$ *lists all free variables of* A*.)*

(L2) $(\mathbf{x}:), \bot \longrightarrow A$ *(where* $\mathbf{x}$ *lists all free variables of* A*.)*

Non-logical Axioms

First we define $x < y =_{def} \exists\zeta(y = x + S\zeta)$, $x \leq y =_{def} x < y \vee x = y$, *and* $x \neq 0 =_{def} \exists\zeta(x = S\zeta)$.

(Z1) $(x:) \longrightarrow x = 0 \vee x \neq 0$

(Z2) $(x:) \longrightarrow \neg(x = 0 \wedge x \neq 0)$

(Z3) $(x:), x = 0 \longrightarrow \neg(x \neq 0)$

(Z4) $(x:), x \neq 0 \longrightarrow \neg(x = 0)$

We also have the following versions of Buss's [1] axioms of arithmetic, or equivalently, those of Cook and Urquhart [3]:

(1) $(x, y:), x < y \vee x = y \longrightarrow x < Sy$

(2) $(x:), x = Sx \longrightarrow \bot$

(3)^B $(x:) \longrightarrow 0 \leq x$

(4) $(x, y:), x \leq y \longrightarrow x = y \vee Sx \leq y$

(5) $(x, y:), x = Sy \longrightarrow 2 \cdot x = SS(2 \cdot y)$

(6) $(x, y:) \longrightarrow x < y \vee x = y \vee y < x$

(7) $(x, y:), x < y \longrightarrow \neg(y < x) \wedge \neg(x = y)$

(8) $(x, y, z:), x < y \wedge y < z \longrightarrow x < z$

(9)^B $\longrightarrow |0| = 0$

(10) $(x:), x \neq 0 \longrightarrow |2 \cdot x| = S|x| \wedge |S(2 \cdot x)| = S|x|$

(11)^B $\longrightarrow |S0| = S0$

(12) $(x, y:), x < y \longrightarrow |x| < |y| \vee |x| = |y|$

[1] As outlined by Definition 6.

(13)^B $(x, y :) \longrightarrow |x \sharp y| = S(|x| \cdot |y|)$

(14)^B $(x :) \longrightarrow 0 \sharp x = S0$

(15) $(x :), x \neq 0 \longrightarrow 1 \sharp (2 \cdot x) = 2 \cdot (1 \sharp x) \wedge 1 \sharp (S(2 \cdot x)) = 2 \cdot (1 \sharp x)$

(16)^B $(x, y :) \longrightarrow x \sharp y = y \sharp x$

(17)^B $(x, y, z :), |x| = |y| \longrightarrow x \sharp z = y \sharp z$

(18)^B $(x, y, u, v :), |x| = |u| + |v| \longrightarrow x \sharp y = (u \sharp y) \cdot (v \sharp y)$

(19)^B $(x, y :) \longrightarrow x \leq x + y$

(20) $(x, y :), x < y \longrightarrow S(2 \cdot x) < 2 \cdot y$ *(see note below)*

(21)^B $(x, y :) \longrightarrow x + y = y + x$

(22)^B $(x :) \longrightarrow x + 0 = x$

(23)^B $(x, y :) \longrightarrow x + Sy = S(x + y)$

(24)^B $(x, y, z :) \longrightarrow (x + y) + z = x + (y + z)$

(25) $\begin{cases} (x, y, z :), x + y = x + z \longrightarrow y = z \\ (x, y, z :), x + y < x + z \longrightarrow y < z \end{cases}$

(26)^B $(x :) \longrightarrow x \cdot 0 = 0$

(27)^B $(x, y :) \longrightarrow x \cdot (Sy) = (x \cdot y) + x$

(28)^B $(x, y :) \longrightarrow x \cdot y = y \cdot x$

(29)^B $(x, y, z :) \longrightarrow x \cdot (y + z) = x \cdot y + x \cdot z$

(30) $\begin{cases} (x, y, z :), (x \neq 0 \wedge y < z) \longrightarrow x \cdot y < x \cdot z \\ (x, y, z :), (x \neq 0 \wedge x \cdot y < x \cdot z) \longrightarrow y < z \end{cases}$

(31) $(x :), x \neq 0 \longrightarrow |x| = S(\lfloor x/2 \rfloor)$

(32) $\begin{cases} (x, y :), x = \lfloor y/2 \rfloor \longrightarrow (2 \cdot x = y \vee S(2 \cdot x) = y) \\ (x, y :), (2 \cdot x = y \vee S(2 \cdot x) = y) \longrightarrow x = \lfloor y/2 \rfloor \end{cases}$

Here (-)^B denotes that the axiom is exactly as in [1]. Axioms (6)-(8) are equivalent to Buss's (6)-(8). Axiom (20) follows easily from (5), (21), (29), and the equality axioms. Similarly the first half of (30) is redundant using (29) and (24). We assume $|x|$ is p-time realized (by the obvious function $\lambda x.|x|$).

(PIND) $(\mathbf{x} :), A(0) \wedge \forall \xi (A(\lfloor \xi/2 \rfloor) \supset A(\xi)), (x :) \longrightarrow A(x)$ *(where* $\mathbf{x}$ *lists all free variables of A other than x.) There is a restriction on this rule, as outlined below. This rule is slightly stronger than the corresponding rule of [1].*

Equality Axioms

(E1) $(x :) \longrightarrow x = x$

(E2) $(x, y :), x = y \longrightarrow y = x$

(E3) $(x_1, \ldots, x_n, y_1, \ldots, y_n :), x_1 = y_1 \wedge \ldots \wedge x_n = y_n \longrightarrow f(x_1, \ldots, \cdot_n) = f(y_1, \ldots, y_n)$ *(where f is a realizable function symbol.)*

(E4) $(x_1, \ldots, x_n, y_1, \ldots, y_n :), x_1 = y_1 \wedge \ldots \wedge x_n = y_n \wedge P(x_1, \ldots, x_n) \longrightarrow P(y_1, \ldots, y_n)$

Structural Rules

(thinning) $\dfrac{\Gamma, \Delta \longrightarrow A}{\Gamma, B, \Delta \longrightarrow A}$

(curry) $\dfrac{\Gamma, (x, y :), \Delta \longrightarrow A}{\Gamma, (x :), (y :), \Delta \longrightarrow A}$

(uncurry) $\dfrac{\Gamma, (x :), (y :), \Delta \longrightarrow A}{\Gamma, (x, y :), \Delta \longrightarrow A}$

(cut) $\dfrac{\Delta, B, \Theta \longrightarrow A \quad \Gamma \longrightarrow B}{\Delta', \Gamma', \Theta \longrightarrow A}$

where Δ', Γ' are Δ, Γ interleaved with some variable declarations omitted (see Remarks following).

Logical Rules

($\supset$ L) $\dfrac{\Delta, B, \Theta \longrightarrow C \quad \Gamma \longrightarrow A}{\Delta', (A \supset B), \Gamma', \Theta \longrightarrow C}$

where Δ', Γ' are as in (cut).

($\supset$ R) $\dfrac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow (A \supset B)}$

($\wedge$ L) $\dfrac{\Gamma, A, \Delta \longrightarrow C}{\Gamma, (A \wedge B), \Delta \longrightarrow C} \qquad \dfrac{\Gamma, B, \Delta \longrightarrow C}{\Gamma, (A \wedge B), \Delta \longrightarrow C} \qquad \dfrac{\Gamma, A, B, \Delta \longrightarrow C}{\Gamma, (A \wedge B), \Delta \longrightarrow C}$

($\wedge$ R) $\dfrac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow B}{\Gamma \longrightarrow A \wedge B}$

($\vee$ L) $\dfrac{\Gamma, A, \Delta \longrightarrow C \quad \Gamma, B, \Delta \longrightarrow C}{\Gamma, (A \vee B), \Delta \longrightarrow C}$

($\vee$ R) $\dfrac{\Gamma \longrightarrow A}{\Gamma \longrightarrow A \vee B} \qquad \dfrac{\Gamma \longrightarrow B}{\Gamma \longrightarrow A \vee B}$

($|\forall|$ L) $\dfrac{\Gamma, A(x), \Delta \longrightarrow B}{\Gamma, \forall \xi \leq |y|\, A(\xi), x \leq |y|, \Delta \longrightarrow B}$

($|\forall|$ R) $\dfrac{\Gamma, (x :), x \leq |y| \longrightarrow A(x)}{\Gamma, [x :] \longrightarrow \forall \xi \leq |y|\, A(\xi)}$

($\forall$ L) $\dfrac{\Gamma, (x :), A, \Delta \longrightarrow B}{\Gamma, \forall \xi A(\xi), (x :), \Delta \longrightarrow B}$

($\forall$ R) $\dfrac{\Gamma, (x :) \longrightarrow A(x)}{\Gamma, [x :] \longrightarrow \forall \xi A(\xi)}$

($\exists$ L) $\dfrac{\Gamma, (x :), A, \Delta \longrightarrow B}{\Gamma, [x :], \exists \xi A(\xi), \Delta \longrightarrow B}$

($\exists$ R) $\dfrac{\Gamma \longrightarrow A}{\Gamma \longrightarrow \exists \xi A(\xi)}$

Remarks: There are several restrictions on these rules. As discussed in the introduction, many of these rules will apply only to certain realizers of the premises and so are not solely determined by the logical form of the sequents involved.

Before we start, however, there is a useful definition (again, this is a somewhat simplified version; a more complete version will appear in the second author's thesis.)

Definition 5 *An occurrence of a formula A [respectively, a declaration of a variable $(x:)$] is* tame *with respect to a realizer e $\|\vdash \Delta, A, \Theta \longrightarrow B$ [respectively $e \|\vdash \Delta, (x:), \Theta \longrightarrow B$] iff, for each $\delta \|\vdash \Delta$, there is $k_\delta \in \mathcal{N}$ so that the runtime degree of $e(\delta)(a)$ is $\leq k_\delta$ for any $a \|\vdash A$ [respectively a realizing x]. (Hence the runtime degree of $e(\delta)(a)$ does not depend on the choice of a.)*

We shall allow a tame declaration to be pushed past following declarations and past formulæ in which it does not occur (except as a dummy free variable), *viz.*:

$$\frac{\Delta, (x:), \Theta \longrightarrow A}{\Delta, \Theta, (x:) \longrightarrow A}$$

if the runtime of $\Theta \longrightarrow A$ does not depend on x, and if x does not occur free in Θ, except possibly as a dummy. (This is sufficient for this paper, but further details will appear in the second author's thesis.)

(PIND) applies to $a_0 \|\vdash A(0), p \|\vdash \forall\xi(A(\lfloor\xi/2\rfloor) \supset A(\xi))$ if there is a p-time function $b(x)$ so that the function α defined by

$$\begin{cases} \alpha(0) &= a_0 \\ \alpha(x) &= p(x)(\alpha(\lfloor x/2\rfloor)) \end{cases}$$

is bounded by b, *i.e.* $\alpha(x) \leq b(x)$. (In fact this shows how to realize $(PIND)$ as well, since then $\alpha = e(a_0, p)$ will make $e \|\vdash (PIND)$.)

(thinning) B must be a formula whose variables are declared in Γ, or may itself be a new variable declaration. (This new formula occurrence or declaration is tame.)

(uncurry) The declaration $(x:)$ must be tame.

(cut) (In the following discussion, "variable declarations" will include shadow declarations as well as ordinary ones.)

Θ and Γ may contain no variable declarations in common. (Actually, this follows from our assumptions on dummy variables, since if Θ and Γ did share a declaration, the two occurrences of the cut formula B would differ in their dummy variables, invalidating the cut.)

If Δ, Γ have a variable declaration $(x:)$ (or $[x:]$) in common, say

$$\Delta_1, (x:), \Delta_2, B, \Theta \longrightarrow A \qquad\qquad \Gamma_1, (x:), \Gamma_2 \longrightarrow B$$

we require Γ_2 to be non-empty, and then the conclusion of the rule is

$$\Delta_1, \Gamma_1, (x:), \Delta_2, \Gamma_2, \Theta \longrightarrow A$$

There is quite a bit of tameness built into this conclusion: it is possible to permute the Δ_1, Γ_1 in any way that preserves the order of the entries of Δ_1 and the order of the entries of Γ_1 (considered separately). So the entries of Γ_1 can be moved through Δ_1. This also applies to the other cases below.

Should Γ_2 be empty, the cut rule would still be valid if the last part of Δ_2—or $(x:)$ if Δ_2 is empty—were tame.

If Γ, Δ have several declarations in common, they must be in the same order, and the conclusion interleaves them in a similar manner. Again, we require that the last $\Gamma_n =_{def} \Gamma'$ be non-empty, unless the last part of Δ is tame. So in the notation of the rule, Δ' is some interleaving of Δ and that part of Γ before the last declaration, and Γ' is the "tail" of Γ.

If Γ, Δ have no variable declarations in common, (whence B must be closed), then we require that Γ be non-empty unless Δ is empty, and the conclusion of the cut rule is just

$$\Delta, \Gamma, \Theta \longrightarrow A$$

($\supset$ L) Similar conditions on variable declarations to those above in (cut).

Aside: The form of the cut rule is somewhat different from what one might expect, by comparison with other logical systems without the exchange rule. Furthermore, the conditions on non-emptiness (and tameness) are somewhat non-standard. The reason for this is that cut is closely related to the S-combinator, and so considerable care must be exercised with these "two-premise" rules (*viz.* the cut rule and the related ($\supset$ L) rule) to avoid the problems associated with the un-realizability of the S-combinator (see [4]). One concession to these concerns is that we do not expect the rule to be p-time itself (in the realizers of the premises), merely that its output is (see Definition 8). Another is that we use the same trick that allows exp to be realizable (when suitably curried): the "function application" implicit in the cut rule is postponed until its execution will be p-time in the current input. (For example, this would be the case if Γ_2 were non-empty, in the notation above.) Finally, we must be sure that the order of the inputs is preserved, unless tameness allows some permutation. For example, consider this attempt at a cut:

$$\frac{(x:), B \longrightarrow B \quad C, (x:) \longrightarrow B}{(x:), C \longrightarrow B}$$

As discussed in the introduction, this is unsound because the C and $(x:)$ have been permuted; it fails to satisfy the restriction that Γ_2 be non-empty, and so is not allowed in our system. However, if the declaration $(x:)$ were tame, then the restriction would no longer be imposed, and the cut (and the permutation) would then become valid.

Note that the interleaving in these two rules provides the means whereby contraction—of variable declarations—and exchange enter the system (without tameness). However, it can be seen that the permutations allowed by cut preserve the order of inputs in the premise sequents; this is controlled by the variable declarations, and it is here that shadow declarations are needed. In some of the quantifier rules a variable declaration is "quantified out", but there remains a "tacit" presence of the variable in the quantified formula. The shadow declaration is there to signal this presence in any subsequent application of cut. An instance of this can be seen in the proof below of the cut elimination theorem.

($\wedge$ L)$''$ (In the third ($\wedge$ L) rule,) the formula (occurrence) A must be tame.

($|\forall|$ L) The declarations $(x:), (y:)$ must appear in Γ, (in any position).

($|\forall|$ R) The usual restriction—x not free in Γ—is built into the syntax. Furthermore, the declaration $(x:)$ must be tame. (Of course, the declaration $(y:)$ must appear in Γ.)

($\forall$ L) The declaration $(x:)$ must be tame.

($\forall$ R) The usual restriction—x not free in Γ—is built into the syntax.

($\exists$ L) The usual restriction—that x occurs free only in A, not in Γ, Δ nor B—is (implicit in the syntax and is) augmented as discussed in the introduction, *viz.* the declaration $(x:)$ must be tame.

Note that the third ($\wedge$ L) rule implies the other (more conventional) two, given thinning. The converse rule

$$\frac{\Gamma, A \wedge B, \Delta \longrightarrow C}{\Gamma, A, B, \Delta \longrightarrow C}$$

can be derived. So in effect $A \wedge B$ in the hypothesis of a sequent amounts to having A, B plus a certain degree of tameness.

Definition 6 *A derivation of a sequent consists of a finite tree such that each branch ends in an axiom and each step is one of the rules of inference given in Definition 4 above, or is a substitution instance of such an axiom or rule.*

We remark here that by a "substitution instance" of a sequent $\Gamma \longrightarrow A$ we mean

- the replacement throughout the sequent of a free variable x, say, occurring in the sequent, by a *realizable* term $t(\mathbf{x})$ which has only new free variables $\mathbf{x}$, not occurring in the sequent, and
- the replacement of the declaration $(x :)$ by the simultaneous declaration $(\mathbf{x} :)$.
- There is one small problem, however, in case the term t is closed; the loss of the indicator provided by the free variable declaration can destroy the realizability unless the formula occurrence or variable declaration (if any) before the declared variable were tame. So we must add this tameness condition when t is closed.

A substitution instance of a rule is defined similarly—take the corresponding substitution instances of the sequents involved in the rule.

Note that we only allow substitution instances of realizable terms.

3 Realizability for sequents and rules

As discussed in the Introduction, we shall treat realizability for sequents as if the sequents consisted of a successive introduction of premises, *i.e.* a nested sequence of implications. Within these successive hypotheses, a variable declaration amounts, in effect, to another such hypothesis. Finally, a rule is realized by a function (not necessarily p-time, however) that assigns a realizer of the conclusion to a simultaneously-presented tuple of realizers of the premises of the rule. These points are summarized in the following definitions:

Definition 7 *For a sequent $\Gamma \longrightarrow A$, we define $e \Vdash \Gamma \longrightarrow A$ inductively:*

1. *$e \Vdash B \longrightarrow A$ iff $e \Vdash B \supset A$.*
2. *$e \Vdash (x :) \longrightarrow A$ iff $e \Vdash \forall \xi A(\xi)$.*
3. *$e \Vdash B, \Gamma \longrightarrow A$ iff e is (the code of) a function which to any $b \Vdash B$, produces an output $e(b) \Vdash \Gamma \longrightarrow A$.*
4. *$e \Vdash (x :), \Gamma \longrightarrow A$ iff e is (the code of) a function which to any standard numeral $m \in \mathcal{N}$, produces an output $e(m) \Vdash \Gamma(x := m) \longrightarrow A(x := m)$.*

Definition 8 *Given a rule of the form*

$$\frac{P_1, \ldots, P_n}{C}$$

where $P_i (i = 1, \ldots, n)$, C, are sequents, we say f realizes the rule iff f is a function (not necessarily p-time) so that for realizers $e_i \Vdash P_i$, $f(e_1, \ldots, e_n) \Vdash C$.

Note that in this definition, although f will be (an index of) a recursive function, we do not insist that f be p-time, only that the value $f(e_1, \ldots, e_n)$ be a polynomial-time realizer (of C.)

Proposition 1 *Each of the rules of Definition 4 is realizable.*

Proof (Sketch only) For each of the rules, we must show how to define the realizer of the conclusion from the realizers of the premises; this means giving both the machine and clock parts e_m, e_t of a realizer e. In this sketch we shall leave the clock part to the reader except for a couple of examples, and shall define the (machine part of the) realizer merely by giving an equation for the "fully evaluated form". Lower case letters will represent realizers of the corresponding formulæ given by upper case letters. We shall write $e(\gamma)$ for $e(a_1)(a_2)\ldots(a_n)$ when $\Gamma = A_1, A_2, \ldots, A_n$. Variable declarations are realized by standard numerals $m \in \mathcal{N}$. Formulæ involving functions ($A \supset B$ or $\forall \xi A(\xi)$, as appropriate) will be realized by p; e, f will denote realizers of the premises of the rule. Finally we shall denote the realizer of the rule by the same name as the rule.

- $(\text{thin})(e)(\gamma)(b)(\delta) = e(\gamma)(\delta)$
- $(\text{curry})(e)(\gamma)(m)(m')(\delta) = e(\gamma)\langle m, m'\rangle(\delta)$
- $(\text{uncurry})(e)(\gamma)\langle m, m'\rangle(\delta) = e(\gamma)(m)(m')(\delta)$

  (Recall here that by hypothesis, the variable declaration $(x :)$ is tame, and so $e(\gamma)$ is p-time in $\langle m, m'\rangle$.)
- $(\text{cut})\langle e, f\rangle(\delta_1)(\gamma_1)(m)(\delta_2)(\gamma_2)(\theta) = e(\delta_1)(m)(\delta_2)(f(\gamma_1)(m)(\gamma_2))(\theta)$ (and similarly if there are more declarations.)

  (Note that the δ's and γ's operate independently. Note also that if γ_2 is non-empty, then we compute the machine $e(\delta_1)(m)(\delta_2)$ before its input $f(\gamma_1)(m)(\gamma_2)$, and so have the run-time bound we need to guarantee we stay within the p-time context when calculating $e(\delta_1)(m)(\delta_2)(f(\gamma_1)(m)(\gamma_2))$. This calculation will be p-time in the length of the last entry in γ_2. Contrast this with the fact that calculating $e(m)(f(m))$ is generally not p-time in $|m|$, unless a tameness condition gives the run-time bound for $\lambda x.e(m)(x)$ before we read m.)
- $(\supset \text{L})\langle e, f\rangle(\delta_1)(\gamma_1)(m)(\delta_2)(p)(\gamma_2)(\theta) = e(\delta_1)(m)(\delta_2)(p(f(\gamma_1)(m)(\gamma_2)))(\theta)$ (and similarly if there are more declarations.)
- $(\supset \text{R})(e)(\gamma)(a) = e(\gamma)(a)$
- $(\wedge \text{L})(e)(\gamma)\langle a, b\rangle(\delta) = e(\gamma)(a)(\delta)$
- $(\wedge \text{L})'(e)(\gamma)\langle a, b\rangle(\delta) = e(\gamma)(b)(\delta)$
- $(\wedge \text{L})''(e)(\gamma)\langle a, b\rangle(\delta) = e(\gamma)(a)(b)(\delta)$

  (Recall here that by hypothesis, the formula A is tame, and so $e(\gamma)$ is p-time in $\langle a, b\rangle$.)
- $(\wedge \text{R})\langle e, f\rangle(\gamma) = \langle e(\gamma), f(\gamma)\rangle$
- $(\vee \text{L})\langle e, f\rangle(\gamma)\langle i, a\rangle(\delta) = \begin{cases} e(\gamma)(a)(\delta) & \text{if } i = 0 \\ f(\gamma)(a)(\delta) & \text{if } i \neq 0 \end{cases}$
- $(\vee \text{R})(e)(\gamma) = \langle 0, e(\gamma)\rangle$
- $(\vee \text{R})'(e)(\gamma) = \langle 1, e(\gamma)\rangle$
- $(|\forall| \text{ L})(e)(\gamma)(p)(q)(\delta) = e(\gamma)(\beta(m, p))(\delta)$, where β is the standard projection function, and m represents the realizer for the variable declaration $(x :)$, (part of Γ.)
- $(|\forall| \text{ R})(e)(\gamma) = \langle e(\gamma)(0)(p(0)), \ldots, e(\gamma)(|m|)(p(|m|))\rangle$, where m realizes the declaration $(y :)$, and $p(k) \mid\!\!\vdash k \leq |m|$, for each $k \leq |m|$.

  (Recall here that by hypothesis, the variable declaration $(x :)$ is tame, and so $e(\gamma)$ has runtime independent of $i, p(i)$.)
- $(\forall \text{ L})(e)(\gamma)(p)(m)(\delta) = e(\gamma)(m)(p(m))(\delta)$

  (Recall here that by the tameness hypothesis, the run-time of $e(\gamma)(m)$ is independent of m.)
- $(\forall \text{ R})(e)(\gamma)(m) = e(\gamma)(m)$
- $(\exists \text{ L})(e)(\gamma)\langle m, a\rangle(\delta) = e(\gamma)(m)(a)(\delta)$

  (Recall here that by the tameness hypothesis, $e(\gamma)$ is p-time in $\langle k, a\rangle$.)
- $(\exists \text{ R})(e)(\gamma) = \langle m, e(\gamma)\rangle$ where m is the realization of the variable declaration $(x :)$ in Γ.

Given the importance of the clock part of our realizers, we ought to illustrate how the rules generate them. Since the details quickly become quite horrid (and rather heavily laden with notation), we shall give just two simple examples for the cut rule.

$$\frac{(x:), B \longrightarrow A \quad (x:), C \longrightarrow B}{(x:), C \longrightarrow A}$$

This is a straight-forward composition, the essence of the cut rule. Set $(\mathrm{cut})\langle e, f\rangle = g = \langle g_m, g_t\rangle$. Also we shall write $g_m(x) = \langle g_m^x, g_t^x\rangle$, $g_m^x(c) = \langle g_m^{xc}, g_t^{xc}\rangle$, letting x represent its own realizer, $c \Vdash C$. (And similarly for e, f.) The clause above gives $g(x)(c) = e(x)(f(x)(c))$; let us see how that works.

g_m takes x, calculates $e(x), f(x)$, and outputs the appropriate code for the machine g_m^x that does the following:

> `read input` c
> `run the program module "`f_m^x`" on the input` c
> `run the program module "`e_m^x`" using the output of "`f_m^x`" as input`
> `produce this as output, together with the constant` g_t^{xc}

Finally g_m also outputs the polynomial g_t^x described below.

When run on input x, g_m takes time $e_t(|x|) + f_t(|x|)$ to produce the required program modules; writing the rest is at most linear in $|x|$, and so we get the required polynomial g_t in $|x|$.

It is now clear that $g_m^x(c)$ will run e_m^x on $f_m^x(c)$ to produce the constant $g_m^{xc} \Vdash A$; this will take time $e_t^x(|f_m^x(c)|) \leq e_t^x(f_t^x(|c|))$, giving the polynomial g_t^x in $|c|$. (Notice that the instructions for producing g_t^x are available as soon as $e(x), f(x)$ are calculated, and so can be produced as part of $g_m(x)$.)

This completes the process, but for the trivial constant g_t^{xc}, which is completely arbitrary.

Now, consider an instance of cut (essentially the S-combinator) which needs some tameness since Γ_2 is empty :

$$\frac{(x:), B \longrightarrow A \quad (x:) \longrightarrow B}{(x:) \longrightarrow A}$$

Here g_m takes x, calculates $e(x), f(x)$, runs e_m^x on f_m^x, outputting the realizer $g_m^x \Vdash A$ (and an arbitrary constant g_t^x). The time required to compute $e_m(x), f_m(x)$ is $e_t(|x|) + f_t(|x|)$; to run $e_m(x)$ on $f_m(x)$ takes $e_t^x(|f_m(x)|) \leq e_t^x(f_t(|x|))$. But here there is a problem: we do not know what e_t^x is until we have x, preventing us from giving the clock polynomial g_t as part of g. However, if $(x:)$ is tame then we have, by definition, a constant k so that the degree of e_t^x is $\leq k$ for any x. So we can replace e_t^x by any polynomial e_t° of degree $\geq k+1$. Then we can bound the run-time by $e_t(|x|) + f_t(|x|) + e_t^\circ(f_t(|x|))$, giving g_t, a polynomial in $|x|$ as required.

Theorem 1 *The sequent calculus given in Definition 4 is sound with respect to realizability.*

Proof All that remains to be shown is that the axioms are realizable. But Axiom (L1) is trivially realized, as is Axiom (L2), since $\perp$ is not realizable. Likewise, the equality axioms are evident from the definition of realizability. For the non-logical axioms, they are shown realizable in [1], except for our version of (*PIND*); this is clearly realizable, as described above.

For completeness we ought to mention the following result, whose proof will appear elsewhere, along with the appropriate definitions; (*cf.* [4] and the second author's thesis.) Of course, it was the belief in the plausibility of such a theorem that motivated the three of us in developing this system. The proof uses Cobham's characterization of polynomial-time functions.

Theorem 2 *Every polynomial-time computable function is realizably representable (in this logic), and conversely every realizably representable function is polynomial-time computable.*

4 Cut-elimination

Theorem 3 *For any derivation-tree of a given sequent in the pure logic part of the calculus of Definition 4, there is a derivation-tree of the same sequent in which the cut rule does not appear. Furthermore, in the full calculus, cuts may be restricted to non-logical formulæ.*

Proof This result is traditional—see, *e.g.* [6] for an account in the classical and intuitionistic cases. Our proof of this result follows the traditional double induction used originally by Gentzen; we illustrate the highlights so the reader can verify that the cut elimination process goes through for our calculus, and that the restrictions in Definition 4 are respected.

We shall generally omit the "primes" in the conclusions of the cuts; for the most part this causes no problem, but the case of implication is a bit special, so we shall be more careful there the first time. We shall also illustrate the rôle of shadow declarations in the case of the quantifier rules.

First, we show that in a derivation that ends with a cut and in which no other cut appears, the cut can be moved to a formula with fewer logical connectives or quantifiers. This is, of course, done by induction on the logical complexity of the cut formula B. (If B is introduced on the left by thinning, the cut may be trivially removed.)

Case $B = B_1 \wedge B_2$: Replace

$$\dfrac{\dfrac{\Delta, B_i, \Theta \longrightarrow A}{\Delta, B_1 \wedge B_2, \Theta \longrightarrow A} \quad \dfrac{\Gamma \longrightarrow B_1 \quad \Gamma \longrightarrow B_2}{\Gamma \longrightarrow B_1 \wedge B_2}}{\Delta, \Gamma, \Theta \longrightarrow A} \qquad (i = 1 \text{ or } 2)$$

with

$$\dfrac{\Delta, B_i, \Theta \longrightarrow A \qquad \Gamma \longrightarrow B_i}{\Delta, \Gamma, \Theta \longrightarrow A}$$

(Note the homogeneity condition on subformulæ guarantees that the condition on variable declarations for (cut) is preserved in the second derivation.)

Replace

$$\dfrac{\dfrac{\Delta, B_1, B_2, \Theta \longrightarrow A}{\Delta, B_1 \wedge B_2, \Theta \longrightarrow A} \quad \dfrac{\Gamma \longrightarrow B_1 \quad \Gamma \longrightarrow B_2}{\Gamma \longrightarrow B_1 \wedge B_2}}{\Delta, \Gamma, \Theta \longrightarrow A}$$

with

$$\dfrac{\dfrac{\dfrac{\Delta, B_1, B_2, \Theta \longrightarrow A \quad \Gamma \longrightarrow B_1}{\Delta, \Gamma, B_2, \Theta \longrightarrow A} \quad \Gamma \longrightarrow B_2}{\Delta, \Gamma, \Gamma, \Theta \longrightarrow A}}{\Delta, \Gamma, \Theta \longrightarrow A}$$

The contraction of the Γ's is allowed by tameness: for all but the last entry of the second copy of Γ, the cut rule itself gives the tameness necessary to move the duplicate entries of Γ together and to contract them, and the last entry of the first copy of Γ inherits the tameness of B_1, and so can be contracted as well.

Case $B = B_1 \vee B_2$: Handled similarly.

Case $B = B_1 \supset B_2$: Replace

$$\dfrac{\dfrac{\Delta_1, (x:), \Delta_2, B_2, \Theta_2 \longrightarrow A \quad \Theta_{11}, (x:), \Theta_{12} \longrightarrow B_1}{\Delta_1, \Theta_{11}, (x:), \Delta_2, B_1 \supset B_2, \Theta_{12}, \Theta_2 \longrightarrow A} \quad \dfrac{\Gamma_1, (x:), \Gamma_2, B_1 \longrightarrow B_2}{\Gamma_1, (x:), \Gamma_2 \longrightarrow B_1 \supset B_2}}{\Delta_1, \Theta_{11}, \Gamma_1, (x:), \Delta_2, \Gamma_2, \Theta_{12}, \Theta_2 \longrightarrow A}$$

with

$$\dfrac{\dfrac{\dfrac{\Delta_1, (x:), \Delta_2, B_2, \Theta_2 \longrightarrow A \quad \Gamma_1, (x:), \Gamma_2, B_1 \longrightarrow B_2}{\Delta_1, \Gamma_1, (x:), \Delta_2, \Gamma_2, B_1, \Theta_2 \longrightarrow A} \quad \Theta_{11}, (x:), \Theta_{12} \longrightarrow B_1}{\Delta_1, \Gamma_1, \Theta_{11}, (x:), \Delta_2, \Gamma_2, \Theta_{12}, \Theta_2 \longrightarrow A}}{\Delta_1, \Theta_{11}, \Gamma_1, (x:), \Delta_2, \Gamma_2, \Theta_{12}, \Theta_2 \longrightarrow A}$$

(where we have explicitly indicated the "shifting" of Γ_1 past Θ_{11} allowed by the cut rule.)

There is a general phenomenon at work here: cuts may be permuted. If one regards ($\supset$ L) as cut plus the "evaluation" axiom $A \supset B, A \longrightarrow B$, then what we have really done here is interchange the order in which two cuts were performed. This may generally be done. (This permutation of cuts is one of the axioms for a multicategory, and we can see here why it is necessary.)

Again, notice that the variable declaration restrictions on (cut) and ($\supset$ L) in the first derivation imply the restrictions are met for each cut in the second.

Case $B = \forall\xi \leq |y|\, B_1(\xi)$: Replace

$$\dfrac{\dfrac{\Delta_1, (y:), \Delta_2, (x:), \Delta_3, B_1, \Theta \longrightarrow A}{\Delta_1, (y:), \Delta_2, (x:), \Delta_3, \forall\xi \leq |y|\, B_1, x \leq |y|, \Theta \longrightarrow A} \qquad \dfrac{\Gamma_1, (y:), \Gamma_2, (x:), x \leq |y| \longrightarrow B_1}{\Gamma_1, (y:), \Gamma_2, [x:] \longrightarrow \forall\xi \leq |y|\, B_1}}{\Delta_1, \Gamma_1, (y:), \Delta_2, \Gamma_2, (x:), \Delta_3, x \leq |y|, \Theta \longrightarrow A}$$

with

$$\dfrac{\Delta_1, (y:), \Delta_2, (x:), \Delta_3, B_1, \Theta \longrightarrow A \qquad \Gamma_1, (y:), \Gamma_2, (x:), x \leq |y| \longrightarrow B_1}{\Delta_1, \Gamma_1, (y:), \Delta_2, \Gamma_2, (x:), \Delta_3, x \leq |y|, \Theta \longrightarrow A}$$

Note here that the shadow declaration $[x:]$ forces the correct interleaving in the first cut; without it the x position would not be noted (x would not be a common variable), and the conclusion would appear (incorrectly) as $\Delta_1, \Gamma_1, (y:), \Delta_2, (x:), \Delta_3, \Gamma_2, x \leq |y|, \Theta \longrightarrow A$.

Another point: it could be the case that the positions of x, y were reversed in one of the sequents, but not the other. Then without shadow declarations we could have this cut:

$$\dfrac{\dfrac{\Delta_1, (x:), \Delta_2, (y:), \Delta_3, B_1, \Theta \longrightarrow A}{\Delta_1, (x:), \Delta_2, (y:), \Delta_3, \forall\xi \leq |y|\, B_1, x \leq |y|, \Theta \longrightarrow A} \qquad \dfrac{\Gamma_1, (y:), \Gamma_2, (x:), x \leq |y| \longrightarrow B_1}{\Gamma_1, (y:), \Gamma_2, \longrightarrow \forall\xi \leq |y|\, B_1}}{\Delta_1, (x:), \Delta_2, \Gamma_1, (y:), \Delta_3, \Gamma_2, x \leq |y|, \Theta \longrightarrow A}$$

Quite apart from the matter of whether or not this is valid, (though our rules would permit it,) consider how to push the cut down to B_1: the cut could not be done on the sequents $\Delta_1, (x:), \Delta_2, (y:), \Delta_3, B_1, \Theta \longrightarrow A$ and $\Gamma_1, (y:), \Gamma_2, (x:), x \leq |y| \longrightarrow B_1$ because of the mis-ordering of the variables. This was missed originally since x was not a variable common to both sequents. The shadow declaration restores x to consideration, and makes the attempt at cutting $\forall\xi \leq |y|\, B_1$ invalid.

Other uses of quantifier rules with shadow declarations behave similarly.

Case $B = \forall\xi B_1$: Replace

$$\dfrac{\dfrac{\Delta, (x:), B_1, \Theta \longrightarrow A}{\Delta, \forall\xi B_1, (x:), \Theta \longrightarrow A} \qquad \dfrac{\Gamma, (x:) \longrightarrow B_1}{\Gamma \longrightarrow \forall\xi B_1}}{\Delta, \Gamma, (x:), \Theta \longrightarrow A}$$

with

$$\dfrac{\Delta, (x:), B_1, \Theta \longrightarrow A \qquad \Gamma, (x:) \longrightarrow B_1}{\Delta, \Gamma, (x:), \Theta \longrightarrow A}$$

Case $B = \exists\xi B_1$: Replace

$$\dfrac{\dfrac{\Delta, (x:), B_1, \Theta \longrightarrow A}{\Delta, \exists\xi B_1, \Theta \longrightarrow A} \qquad \dfrac{\Gamma \longrightarrow B_1}{\Gamma \longrightarrow \exists\xi B_1}}{\Delta, \Gamma, \Theta \longrightarrow A}$$

with

$$\dfrac{\Delta, (x:), B_1, \Theta \longrightarrow A \qquad \Gamma \longrightarrow B_1}{\Delta, \Gamma, \Theta \longrightarrow A}$$

This ends the first part of the induction (Gentzen's induction on "degree".)

Next, we reduce the number of sequents above the left (or right) side of the cut which contain the cut formula (Gentzen's induction on "rank"). First we do the left side, (in our set-up, the left and right are not symmetric). We suppose B is not in Γ; (otherwise the cut can easily be eliminated via thinning.) We label the cases by the rule used to create the left side of the cut.

Case (thinning): Replace

$$\dfrac{\dfrac{\Delta', B, \Theta \longrightarrow A}{\Delta, B, \Theta \longrightarrow A} \qquad \Gamma \longrightarrow B}{\Delta, \Gamma, \Theta \longrightarrow A} \qquad (\text{where } \Delta' \subseteq \Delta)$$

with

$$\dfrac{\dfrac{\Delta', B, \Theta \longrightarrow A \qquad \Gamma \longrightarrow B}{\Delta', \Gamma, \Theta \longrightarrow A}}{\Delta, \Gamma, \Theta \longrightarrow A}$$

(If thinning is used in Θ, proceed similarly.)

Case ($\supset$ L): Replace

$$\dfrac{\dfrac{\Delta_1, D_2, \Delta_3, B, \Theta \longrightarrow A \qquad \Delta_2 \longrightarrow D_1}{\Delta_1, (D_1 \supset D_2), \Delta_2, \Delta_3, B, \Theta \longrightarrow A} \qquad \Gamma \longrightarrow B}{\Delta_1, (D_1 \supset D_2), \Delta_2, \Delta_3, \Gamma, \Theta \longrightarrow A}$$

with

$$\dfrac{\dfrac{\Delta_1, D_2, \Delta_3, B, \Theta \longrightarrow A \qquad \Gamma \longrightarrow B}{\Delta_1, D_2, \Delta_3, \Gamma, \Theta \longrightarrow A} \qquad \Delta_2 \longrightarrow D_1}{\Delta_1, (D_1 \supset D_2), \Delta_2, \Delta_3, \Gamma, \Theta \longrightarrow A}$$

(and similarly if the ($\supset$ L) operates in Θ.)

Case ($\supset$ R): Replace

$$\dfrac{\dfrac{\Delta, B, \Theta, A_1 \longrightarrow A_2}{\Delta, B, \Theta \longrightarrow A_1 \supset A_2} \qquad \Gamma \longrightarrow B}{\Delta, \Gamma, \Theta \longrightarrow A_1 \supset A_2}$$

with

$$\dfrac{\dfrac{\Delta, B, \Theta, A_1 \longrightarrow A_2 \qquad \Gamma \longrightarrow B}{\Delta, \Gamma, \Theta, A_1 \longrightarrow A_2}}{\Delta, \Gamma, \Theta \longrightarrow A_1 \supset A_2}$$

Cases ($\wedge$ L), ($\wedge$ R), ($\vee$ L), ($\vee$ R), ($|\forall|$ L), ($|\forall|$ R), ($\forall$ L), ($\forall$ R), ($\exists$ L), ($\exists$ R) are similar.

Finally, we must show how to reduce the number of formulæ above the right side of the cut which contain the cut formula:

Case (thinning) is handled as above for the left side.

Case ($\supset$ L): Replace

$$\dfrac{\Delta, B, \Theta \longrightarrow A \qquad \dfrac{\Gamma_1, G_2, \Gamma_3 \longrightarrow B \qquad \Gamma_2 \longrightarrow G_1}{\Gamma_1, G_1 \supset G_2, \Gamma_2, \Gamma_3 \longrightarrow B}}{\Delta, \Gamma_1, (G_1 \supset G_2), \Gamma_2, \Gamma_3, \Theta \longrightarrow A}$$

with

$$\dfrac{\dfrac{\Delta, B, \Theta \longrightarrow A \qquad \Gamma_1, G_2, \Gamma_3 \longrightarrow B}{\Delta, \Gamma_1, G_2, \Gamma_3, \Theta \longrightarrow A} \qquad \Gamma_2 \longrightarrow G_1}{\Delta, \Gamma_1, (G_1 \supset G_2), \Gamma_2, \Gamma_3, \Theta \longrightarrow A}$$

Cases ($\wedge$ L), ($\vee$ L), ($|\forall|$ L), ($\forall$ L), ($\exists$ L) are similar.

Cases ($\supset$ R), ($\wedge$ R), ($\vee$ R), ($|\forall|$ R), ($\forall$ R), ($\exists$ R) are in essence part of the content of the first part of the induction process, and do not apply in this context. And so this completes (the sketch of) the proof.

References

[1] S. Buss, *Bounded Arithmetic,* Doctoral dissertation, Princeton University, 1985.

[2] S. Buss, "The polynomial time hierarchy and intuitionistic bounded arithmetic", pp. 77 – 103 in Structure in Complexity Theory, 1986, (Springer Lecture Notes in Computer Science 223, A.L. Selman, ed.)

[3] S. Cook and A. Urquhart, "Functional interpretations of feasibly constructive arithmetic", Technical Report 210/88, Department of Computer Science, University of Toronto, 1988.

[4] J.N. Crossley and J.B. Remmel, "Proofs, programs and run-times", Preprint, Monash University, 1989.

[5] G. Gentzen, "Investigations into logical deductions", in M.E. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland, 1969.

[6] J.-Y. Girard, *Proof Theory and Logical Complexity*, Bibliopolis, 1987.

[7] J.-Y. Girard, "Linear logic", J. Theoretical Computer Science 50 (1987), 1 – 102.

[8] J. Lambek, "Deductive systems and categories I", J. Math. Systems Theory 2 (1968), 278 – 318.

[9] J. Lambek, "Deductive systems and categories II", Springer LNM 86 (1969), 76 – 122.

[10] J. Lambek, "Multicategories revisited", pp. 217 – 239 in Categories in Computer Science and Logic, J.W. Gray and A. Scedrov, eds. (Contemporary Mathematics 92 (1989) American Mathematical Society).

[11] A. Nerode, J.B. Remmel, and A. Scedrov, "Polynomially graded logic", pp. 375 – 385 in Proceedings of the Fourth Symposium on Logic in Computer Science, Asilomar 1989 (IEEE Publications).

[12] R.A.G. Seely, "Hyperdoctrines, natural deduction, and the Beck condition", Zeitsch. f. math. Logik und Grundlagen d. Math. 29 (1983), 505 – 542.

[13] R.A.G. Seely, "Graded multicategories of polynomial-time realizers", pp. 182 – 197 in Proceedings of the Conference on Category Theory and Computer Science, Manchester, UK, September 1989 (Springer Lecture Notes in Computer Science 389, D.H. Pitt *et al* eds.)

Departments of Mathematics
and Computer Science
Monash University
Clayton
Victoria 3168
Australia

Department of Mathematics
McGill University
805 Sherbrooke St. W.
Montréal
Québec H3A 2K6
Canada

Department of Mathematics
John Abbott College
C.P. 2000
Ste. Anne de Bellevue
Québec H9X 3L9
Canada