*Electronic Notes in Theoretical Computer Science 50 No. 2 (2001) { Proc. BOTH 2001 URL:* [*http://www.elsevier.nl/locate/entcs/volume50.html*](http://www.elsevier.nl/locate/entcs/volume50.html) *18 pages*

*Hyperformulae, Parallel Deductions and* Intersection Types

*Beatrice Capitani Dipartimento di Matematica*

*Universit a di Siena*

*Via Del Capitano, 15, Siena, Italy E-mail:* [*capitanib@unisi.it*](mailto:capitanib@unisi.it)

*Michele Loreti Betti Venneri Dipartimento di Sistemi ed Informatica*

*Universita di Firenze*

*Via Lombroso, 6/17, Firenze, Italy*

*E-mail:* [*floreti,vennerig@dsi.unifi.it*](mailto:floreti%2Cvennerig@dsi.unifi.it)

*Abstract*

*We aim at investigating the intersection-type assignment system for lambda cal- culus, with the Curry-Howard approach. We devise a propositional logic, whose notable characteristic is the presence of the hyperformulae denoting parallel com- positions of formulae. As such, this logic formalizes a novel notion of parallel de- ductions, while forming a simple generalization of the standard natural deduction framework.*

*We prove that the logical calculus is isomorphic to the intersection type system, by mapping logical deductions into typed lambda terms, encoding those deductions, and conversely. In this context the intersection type constructor, which comes out to be a proof-theoretic operator, is now interpreted as a standard propositional connective.*

# *1 Introduction*

*Intersection types originated in [6] as an extension of Curry's basic type sys-* tem. Its notable characteristic is the presence of a new type constructor (^) for denoting the intersection on types. The system so devised turns out to be

*? This work was partly supported by MURST COFIN'99 TOSCA Project and CNR- GNSAGA.*

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license.*](http://creativecommons.org/licenses/by-nc-nd/3.0/)

*extremely powerful, since it allows the typing of all (and only) the strongly* normalizing lambda terms.

*However a debated question concerns the logical interpretation of this type* theory, since intersection types do not t into the Curry-Howard paradigm.

*It is well known that, in the Curry-Howard approach, reading formulae* as types, constructive proofs of formulae are mapped into lambda terms hav- ing related types and conversely. Thus, for instance, functional type theory and Girard's system F correspond to implicational and second-order logics, respectively.

*In this perspective, intersection on types seems to be somewhat esoteric,* because of the crucial shape of the introductory rule. Namely, the ^ In- troduction rule says that a term M has type ^ if and only if the same term M has both type and type . Thus, in logical terms, ^ becomes a proof-functional connective, restricting the classical conjunction; the proof of the ^ formula depends in an essential way upon intensional aspects of the component subformulae, namely they must be proved by the same proof.

*It is for giving a logical account of the intersection that a Hilbert-style logic* is proposed in [9], where intersection type inference is, however, investigated in the context of Combinatory Logic instead of lambda calculus. In that paper the ^ Introduction rule is avoided by splitting it into two components, a relevant conjunction and the following inference rule:

*(Sub) \any nite intersection of di erent instances of the same theorem is a* theorem"

*This solution is unsatisfactory for our goal, because the (Sub)-rule prevents* from extending that result to lambda calculus by translating the Hilbert-style logic of [9] into a natural deduction version. In fact, a distinguishing feature of the natural deduction framework is the treatment of assumptions, that are

*xed and could loose their status (by being discharged) but not be modi ed.*

*On the contrary, the (Sub)-rule assumes that any assumption may duplicate* in several di erent instances during the deductive process. On the other hand, the strict relation between natural deduction and lambda calculus is a well- known matter, since the introduction and elimination rules for implication correspond quite naturally to the abstraction and application rules of term formation in assigning types to calculus.

*In the present paper we de ne a natural deduction propositional logic and* we prove that it is isomorphic to the intersection type assignment system for lambda calculus. The novelty of this logic comes up from its syntax, involving hyperfomulae as well as implicative and conjunctive formulae. Hyperformulae are intended as sequences of formulae composed by a parallel operator, so that a notion of parallel deductions is represented inside the logical system without requiring any proof-functional condition in the deduction rules. As a result, we exploit derivability of hyperformulae for giving a logical interpretation of the intersection as a standard truth-functional connective.

*The paper is organized as follows. In section 2 we brie y outline the* Intersection Type Assignment system for calculus (T A^). In section 3

*the logic HL is presented, its main properties are proved and the degree*

*of parallelism represented here is discussed. In section 4 logical proofs are* decorated by lambda terms as a technical tool for proving, in section 5, the isomorphism between HL and T A^.

*2 Intersection Type Assignment for Lambda-calculus This section outlines the intersection type assignment system for calculus.*

*Intersection types have been introduced in [7] and [5] to overcome some weak-* nesses of Curry's basic system, while retaining the normalization property. The arrow-based type language of Curry's system is enriched by a new type constructor, ^, for denoting the intersection of two types, and the inference system is extended with rules for assigning ^-types to terms.

*We remark that there are several formulations of the intersection type* theory in the literature. For instance, the complete system, presented in [5], also considers the universal type ! and a preorder relation on types.

*The system considered here, denoted by T A^, is the simplest one and* only involves the basic rules for introducing and eliminating the ^ type con- structor. The main motivation in choosing the simple system T A^ relies on the fact that the present paper aims at investigating, in logical terms, the

*^ derivability without dealing with constants or others features.*

*We brie y recall that -terms are de ned by the following syntax:*

*M; N ::= xj x:M jMN*

*De nition 2.1 Assume that we have in nitely many type variables ; ; ; :::.* The set T of types is inductively de ned thus:

*type variables are types,*

*if ; are types, then so are ! (arrow type) and ^ (intersection* type).

*Notation 1 Parentheses are omitted from types assuming that ! associates* to right and ^ has precedence over !. Moreover, intersection types 1^:: :^ n are considered equal up to permutations and repetitions of i's.

*De nition 2.2 (The system T A^)*

*A statement is an expression of the form M : where M (subject) is a term and (predicate) is a type.*

*A basis B is a nite set of statements whose subjects are all distinct vari-* ables.

*We will use B; x : for B [ fx : g, where x does not belong to B.*

*A statement M : is derivable from a basis B if B `^ M : can be* proved using the following axioms and inference rules.

*Axioms*

*Rules*

*B; x : `^ x : (V ar)*

*B `^ M : B `^ M :*

*B `^ M : ^*

*(^I)*

*B `^ M : ^ B `^ M : ( )*

*B; x : `^ M : B `^ x:M : !*

*(^E)*

*(! I)*

*B `^ M : ! B `^ N : B `^ MN :*

*(! E)*

*We write : B `^ M : to denote a proof of B `^ M : .*

*The system T A^ enjoys the main property that all the strongly normalizing*

*terms are typeable and viceversa.*

*Further details on Lambda-Calculus and Intersection Type Theory can be* found in [3] and [4], respectively.

# *3 The Logic HL of Hyperformulae*

*In this section the propositional logic HL is de ned in natural deduction-style.* This calculus will be proved to be isomorphic to T A^ in section 5, so providing a logical setting for interpreting intersection-type assignment.

*The novelty of HL comes up from its syntax, involving both formulae* and hyperformulae. As usual, formulae are built from propositional variables by means of connectives, namely ! (implication) and ^ (conjunction) in our case. Then hyperformulae are de ned as nite sequences of formulae, composed by the parallel constructor j. As such, the system HL represents a simple generalization of the standard natural deduction framework, while capturing a novel notion of parallel deductions.

*As formal de nitions will clarify, both the order and the position of each* formula in a hyperformula are signi cant. However, during the derivation, a component of a hyperformula can move to a di erent position for fusing with another component. For this reason, we use a special marker, ", to denote a hole in a parallel composition. In other words " can be considered as a logical constant, whose meaning is just the lack of information. It does not contribute to forming implicative and conjunctive formulae, but to forming hyperformulae.

*In section 3.3 we will discuss the parallel operator j, looking at formulae* as processes and the relation between the method of hyperformulae and that of hypersequents [1,2].

*3.1 Syntax*

*De nition 3.1 (Formulae) Let V be a denumerable set of variables. The* set F of formulae is inductively de ned thus:

*V F*

*" 2 F*

*; 2 F n f"g =) ! ; ^ 2 F*

*As for types, we assume that ^ binds stronger then ! and ! associates to* the right. Furthermore, we assume that ^-formulae are equal up to contraction (i.e. ^ = ), commutative and associative properties; for instance ^ ( ^

*) = ^ .*

*De nition 3.2 (Hyperformulae) A hyperformula ' is a structure of the* form

*' = 1j j n (1 n)*

*where 1 n are formulae and i 6= " for some i (1 i n). Let H denote* the set of hyperformulae.

*Observe that, in the previous de nition, the condition i 6= " is assumed* for simplicity, in order to avoid dealing with totally empty hyperformulae, like "j".

*Notation 2*

*Propositional variables are denoted by ; ; ; Æ. Formulae and hyperformu-* lae are denoted by ; ; and '; , respectively (with or without subscripts).

*Given a hyperformula ' = 1j j n, each i is called a component of ', and this is denoted by i 2 ' (i = 1;::: ; n).*

*If ' = 1j j n then we write 'j for 1j j nj .*

*The function length : H ! N is de ned on the structure of hyperformulae,* that is length(') = 1+(the number of j operators occurring in ').

*We write (')i to denote the selection of the i-th component of '.*

*Namely ( 1j ::: j n)i = i if 1 i n, " otherwise.*

*We observe that 'j" 6= '. However the i th selection function is de ned* as a total function, hence (')i = " if i length('), only for technical reasons concerning de nitions and proofs of section 3.2.

*Let us formalize a special kind of substitution for denoting the replacement* of components of hyperformulae.

*De nition 3.3 (Component substitution)*

*'[i 7! ] = ' if i > length('), otherwise it is the hyperformula such that:*

*length( ) = length(');*

*for every j 6= i, ( )j = (')j;*

*( )i = .*

*'[ 7! ] = where is such that:*

*length( ) = length(');*

*( )i = (')i, if (')i 6=*

*( )i = , if (')i = (1 i length(')).*

*De nition 3.4*

*i) A context of assumptions is any nite multiset of hyperformulae.*

*ii) We generalize the function ()i to the context in the following way:*

*( )i*

*= <8*

*; if = ;*

*1 0*

*:f(') g] ( 0 ) if = f'g]*

*i i*

*Where not ambiguous, we will write i for ( )i.*

*iii) We say that is i j monovalent (i; j 1) if and only if*

*8' 2 : (')i = (')j*

*The Hyperformulae Logic, denoted as HL, is a natural deduction style* logic. The following is the inductive de nition of the HL-consequence relation.

*De nition 3.5 (HL-derivability) The relation ` is de ned by the following* axioms and rules.

*Axiom*

*j Weakening*

*^ Introduction*

*;' ` ' (Ax)*

*` 'j*

*` ' (j w)*

*` '*

*` '[i 7! (')i ^ (')j][j 7! "]*

*(^I) 1 i; j length(')*

*if is i j monovalent.*

*^-Elimination.*

*` ' (')i = ^*

*` '[i 7! ]*

*(^E)*

*` ' (')i = ^*

*` '[i 7! ]*

*(^E)*

*!-Introduction.*

*; 1j ::: j n ` 1j ::: j s (! I) (s n);*

*` 1j ::: j s*

*where i (1 i s) is such that i = i ! i if i 6= ", i = " otherwise.*

*1 ] denotes the standard multiset union*

*!-Elimination.*

*` 1j ::: j n ` 1j ::: j n (! E)*

*` 1j ::: j n*

*where i (1 i n) is such that i = i if i = i ! i, i = " otherwise.*

*We write D : ` ' to denote that ` ' is provable by the proof D using* the axiom and rules de ned above.

*The (j w) rule is the only structural rule on parallel deductions, a kind of* internal weakening that allows to drop a nal component in a hyperformula. During the deductive process, a hyperformula can loose some nal compo- nents (by j w) or render some others inactive, by replacing " to them, but

*it never increases its length. The underlying idea can be rephrased as*

*only what was in parallel, will remain in parallel,* that is formalized by the absence of any introductory rule for j.

*The other rules are logical rules and can be divided into global rules and*

*local rules.*

*The global ones are (! I) and (! E), which involve the whole hyperfor-* mula. By (! I) one may discharge an assumption only if the same activity is done in all the other components at the same time. When saying that the performed action is the same, we also mean that what is really discharged is a component of a parallel composition of assumptions. Namely each compo- nent of the hyperformula-assumption is discharged by each component of the hyperformula-conclusion.

*In the same sense the action of eliminating implication (! E) in one* component of a parallel composition must synchronize with an (! E) action from all the other components in order to occur.

*The local rules are (^I) and (^E) which a ect only some components* inside a hyperformula.

*The (^I) rule is very important because it is the only rule that brings* moments of fusion into parallel deductions. Two formulae 1 and 2, running in parallel in a deduced hyperformula 1j 2, must have the same deductive history (with respect to the applied global rules). If they also depend from equal assumption, that is (')1 = (')2 for every assumption ', then they can fuse in one conjunctive formula. We recall that our main goal was to provide a logical account of the ^ connective as a truth-functional connective. Hence we achieved this goal by de ning the (^I) rule in a such way that it does not involve any proof-functional condition.

*Notation 3 If D : ` ' then we write D to denote the relevant context* containing all and only the assumptions of that are actually used in D.

*Obviously D : ` ' implies D : D ` '.*

*The advantage of considering D instead of is that the relevant*

*context enlightens the ordered dependency between derived parallel formulae* and related parallel assumptions. In fact, by a simple inspection of axiom and rules of De nition 3.5, it is easy to verify that if D : D ` then for any ' belonging to D:

*lenght(') length( );*

*( )i 6= " implies (')i 6= ";*

*that is the deduction of ( )i depends on the i th component of each assump-* tion in D.

*Example 3.6 Let = f( 1 ! 2)^( 1 ! 3)j( 1 ! 2)^( 1 ! 3); 1j 1g, notice that is 1 2 monovalent. The following is a proof in HL.*

*` ( 1 ! 2) ^ ( 1 ! 3)j( 1 ! 2) ^ ( 1 ! 3) (^E)*

*` ( 1 ! 2) ^ ( 1 ! 3)j 1 ! 3 (^E)*

*` 1j 1 ` 1 ! 2j 1 ! 3*

*` 2j 3 (^I)*

*(! E)*

*` 2 ^ 3j" (! I)*

*( 1 ! 2) ^ ( 1 ! 3)j( 1 ! 2) ^ ( 1 ! 3) ` 1 ! 2 ^ 3j" (! I)*

*` ( 1 ! 2) ^ ( 1 ! 3) ! 1 ! 2 ^ 3j" (j w)*

*` ( 1 ! 2) ^ ( 1 ! 3) ! 1 ! 2 ^ 3*

*3.2 Main Syntactic Properties*

*This section is devoted to state some basic properties that clarify how proofs* are constructed in HL and will be used in following sections.

*As far as structural rules are concerned, we rst notice that is a multiset* of assumptions, hence it does not change when its elements are permuted. It is also easy to verify that D : ` ' if and only if D : 0 ` ' when is equal to 0 but for repetition of some assumptions.

*With regards to the weakening property, we have already observed that* (j w) is a kind of structural rule, which says that any assumption ' can be weakened to 'j (for any formula ).

*Moreover a context of a deduction D can be weakened by adding useless* (dummy) assumptions. In this case, however, we have to require the new assumptions to be i j-monovalent if was so, in order to guarantee that possible applications of the (^ I) rule in D still hold.

*This is summarized by the following property.*

*Property 1 (Weakening) If ` ' then ` ' for any , extending ,* such that:

*i) for all ', if ' belongs to then 'j 1j j n belongs to for any 1,:: :, n (n 0).*

*ii) if is i j-monovalent then is i j-monovalent too.*

*De nition 3.7 Let be a context, ' be a hyperformula and p be a permu-* tation of f1;::: ; mg. We de ne the permutation of ' by p, denoted by p('), as the hyperformula such that:

*( )p(i) = (')i (1 i m);*

*length( ) = max(length('); size(p))*

*where size(p) is the greatest i such that p(i) 6= i.*

*We generalize the notion of permutation to contexts, in an obvious way.* Thus p( ) denotes the context obtained by applying p to each hyperformula in .

*For example, let p a permutation of f1; 2; 3g such that p(1) = 2, p(2) = 3 and p(3) = 1, then*

*p( j j 0 j 0 ) = 0 j j j 0 p( j ) = "j j*

*The following lemma shows how deductions running in parallel can be* permuted to obtain them in a di erent order.

*We observe that any permutation of the conclusion requires the same per-* mutation to be applied to the assumptions that belong to the context of the deduction.

*Lemma 3.8 (Commutation Property) Let D be a proof of ` ' and let* p be a permutation of f1;::: ; mg such that m length('). Then there exists a proof D0 of p( ) ` p(').

*Proof. The proof proceeds by induction on D. The base case is trivial. In* the inductive step, the only interesting cases are (j w) and (^I).

*In the rst case we have:*

*` 'j*

*` ' (j w)*

*Since length(') m then length('j ) m. By induction hypothesis there exists a proof of p( ) ` p('j ). Moreover p is a permutation of f1;::: ; mg where m length('), then p('j ) = p(')j . Thus*

*p( ) ` p(')j*

*p( ) ` p(')*

*(j w)*

*For the second case, the thesis follows from the induction hypothesis, be-* cause if is i j monovalent then p( ) is p(i) p(j) monovalent.

*2*

*Lemma 3.9 If D : ; ` ' then there exists a proof D0 such that D0 :*

*; [ 7! ^ ] ` ' for any formula .*

*Proof. By induction on D. If D : ; ` by (Ax), that is ' = , then* construct D0 thus:

*; [ 7! ^ ] ` [ 7! ^ ]*

*; [ 7! ^ ] ` '*

*(^E)*

*In the inductive step, the only interesting case is when the last applied rule* is (^I). Then the thesis follows from the induction hypothesis, because all formulae , such that 2 ', are simultaneously replaced by ^ , so than any i j monovalency of the context is preserved by the component-substitution.

*2*

*The lemma above allows to perform a more careful analysis of the relation-* ship between hyperformulae and ^ formulae, that is stated in the following theorem.

*Theorem 3.10 (From j to ^) If D : ` j then there exists a proof D0*

*D0 : 0 ` ^*

*where 0 is such that, for any ', ' belongs to if and only if '[1 7! (') ^* (') ][2 7! (') ^ (') ] belongs to 0 .

*1*

*2 1 2*

*Proof. Construct a proof of 0 ` j by using Lemma 3.9. Hence, (^I)* applies since 0 is 1 2 monovalent, thus D0 : 0 ` ^ . 2

*To sum up, we observe that assumptions are intended as packets of for-* mulae in parallel, that are used to deduce formulae in parallel, by keeping assumed and deduced formulae in lockstep during the deduction.

*Instead, we need a richer context for deducing an ^ formula. First, all* the assumptions, that are used in all the conclusions, must be composed in one ^ formula. Then several copies of this ^ formula, composed by j in one hyperformula, are available as assumptions for the deductions, which have to go on in a parallel way in order to fuse their conclusions at the nal step. In other words, the main di erence between a deduction of j and a deduction of ^ consists in using di erent resources (assumptions) in a di erent way during the parallel deductions of and , respectively.

*3.3 Hyperformulae, Hypersequents and Parallelism*

*The key idea of the logic HL is the notion of hyperformulae, that allows* handling the (metalogic) concept of packets of parallel deductions by only using logical rules in standard natural deduction style.

*This approach is closely related to the method of Hypersequents, introduced* by Avron for representing the proof theory of non-classical logics (see [1,2]).

*Hypersequents are de ned as nite sequences of Sequents composed by a* parallel operator. Thus they form a generalization of the sequential framework as well as the HL logic generalizes the natural-deduction framework.

*However the main di erence between the two approaches relies on the* degree of parallelism represented by the logical rules. The interpretation of Avron's parallel constructor is disjunctive, since most of the deduction rules treat only one component of the hypersequent, in other words rules can be applied concurrently.

*Instead, in the logic HL the interpretation of j is strongly conjunctive.* From a computational view point, one may look at hyperformulae as com-

*positions of processes running in parallel. In this perspective, each application*

*of a deduction rule can be represented as a labeled event, where a label con-* tains both the name of the applied rule and the related argument, for instance the discharged assumption in the (! I) and the minor premise in the (! E). Then an labeled event in a process of a parallel composition must syn- chronize with labeled events from all the other components in order to form

*a synchronization event labeled by .*

*To sum up, in the logic HL the maximum parallelism is represented by* hyperfomulae, rules cannot be applied asynchronously, and the kind of syn- chronization required in deductions is more close to that of CSP [8] than to the synchronization implicit in Avron's logic.

# *4 Labeling proofs by lambda-terms*

*In this section we de ne a labeling procedure that associates -terms to HL-* proofs. This annotation of proofs by terms is instrumental to the isomorphism between T A^ and HL. Informally speaking, the rst step will consist in la- beling the context of a deduction by associating distinct lambda variables to hyperformulae which are assumptions. Then, once a labeled set of assump- tions is provided, any (! I) and (! E)-rule application will correspond to perform a abstraction and an application on the label, respectively. In- stead, local rules, that is (^I), (^E), and (j w), do not modify the term decorating the proof. Finally, the term associated to the whole proof will encode, by its structure, the deductive history of the proof.

*De nition 4.1 Let bea context. Let be a function associating all distinct*

*variables to the hyperformulae in . Then is the labeled version of such that, for all ' in ,*

*if (') = x then x : ' 2.*

*We extend to all the functions and notations de ned on , in an obvious* way.

*Let us notice that ()i is a set while ( )i is a multiset.*

*De nition 4.2 Let D be a proof of ` '. For any given, the term M* labeling D is de ned by induction on D using the following rules

*Case (Ax)*

*If ` '; where ' 2 ; then ` (') : '*

*Case (j w)*

*Case (^E)*

*` M : 'j ) ` M : '*

*` M : '; (')i = ^ ) ` M : '[i 7! ]*

*Case (^I)*

*` M : ' ) ` M : '[i 7! (')i ^ (')j][j 7! "]*

*Case (! I)*

*;x : 1j ::: j n ` M : 1j ::: j s; (s n) ) ` x:M : 1j ::: j s according to rule (! I) in De nition 3.5*

*Case (! E)*

*` M : 1j ::: j n; ` N : 1j ::: j n ) ` MN : 1j ::: j n according to rule (! E) in De nition 3.5*

*We write D : ` M : ' to denote the labeling, by the term M , of D*

*for a given and we say that D is M -labeled.*

*Notice that, for di erent, di erent terms can be associated to a* deduction D. However all these terms are equal on their structure, they only di er in names of free variables. In fact, the structure of the proof strictly corresponds to the structure of the associated term, but for rules (j w), (^E) and (^I).

*Example 4.3 Let = fx : ( 1 ! 2) ^ ( 1 ! 3)j( 1 ! 2) ^ ( 1 !*

*3); y : 1j 1g then labeling of the proof in Example 3.6 is the following.*

*` x : ( 1 ! 2) ^ ( 1 ! 3)j( 1 ! 2) ^ ( 1 ! 3) (^ )*

*E*

*` x : ( 1 ! 2) ^ ( 1 ! 3)j 1 ! 3 (^ )*

*E*

*` y : 1j 1 ` x : 1 ! 2j 1 ! 3 (! )*

*E*

*` xy : 2j 3*

*(^I)*

*` xy : 2 ^ 3j" (! )*

*I*

*x : ( 1 ! 2) ^ ( 1 ! 3)j( 1 ! 2) ^ ( 1 ! 3) ` y:xy : 1 ! 2 ^ 3j" (! )*

*I*

*` x: y:xy : ( 1 ! 2) ^ ( 1 ! 3) ! 1 ! 2 ^ 3j" (j )*

*w*

*` x: y:xy : ( 1 ! 2) ^ ( 1 ! 3) ! 1 ! 2 ^ 3*

*4.1 Properties of labeled proofs*

*We rst reformulate Lemma 3.8 and Theorem 3.10 for labeled proofs in an* obvious way. Then, we prove the Composition Lemma, which is the main result of the present section.

*Lemma 4.4 (Commutation Property for labeled proofs) Let D be an* M -labeled proof of D : ` M : ', and let p be a permutation of f1;::: ; mg,

*where lenght(') m. Then there exists an M -labeled proof D0 of p( 0 ) `*

*M : p(').*

*Proof. The proof is the same as in Lemma 3.8, but using labels as in De ni-* tion 4.2. 2

*Lemma 4.5 (From j to ^ for labeled proofs) If D : ` M : j then*

*there exists an M labeled proof D0 such that D0*

*: 0*

*` M : ^ , where 0*

*is such that x : ' belongs to if and only if x : '[1 7! (')1 ^ (')2][2 7!*

*(') ^ (') ] belongs to 0 .*

*1 2*

*Proof. The proof is the same as in Theorem 3.10, but using labels as in* De nition 4.2. 2

*We de ne now how to construct a labeled context by a parallel mixing of two* contexts.

*De nition 4.6 Let 0*

*and 00*

*be two labeled contexts, then the context*

*k h 0 ; 00 i (n; m 1) is de ned in the following way:*

*n;m*

*if x : ' belongs to 0 and x : belongs to 00 , length(') = ` and*

*length( ) = k, then*

*x : (')1j j(')nj( )1j j( )mj(')n+1j j(')`j( )m+1j (')k*

*belongs to k h 0 ; 00 i*

*n;m*

*if x : ' belongs to 0 and no x labeled assumption belongs to 00 , then*

*(i) if length(') n then x : ' belongs to k h 0 ; 00 i*

*(ii) if n < length(') then*

*n;m*

*x : (')1j j(')nj "j j" j(')n+1j j(')length(')*

*belongs to k*

*h 0 ; 00 i*

*m|t{izme}s*

*n;m*

*if x : belongs to 00 , where length( ) = k, and no x labeled assumption*

*belongs to 0 , then*

*n| t{imz e}s x : "j j" j*

*belongs to k h 0 ; 00 i*

*n;m*

*nothing else belongs to k h 0 ; 00 i*

*n;m*

*We point out that any context k h 0 ; 00 i is built up in such a way*

*that, for any i and j,*

*n;m*

*if 0*

*is i j monovalent (i; j n) then*

*is i j monovalent;*

*if 00 is i j monovalent (i; j m) then is (n +i) (n +j) monovalent.*

*This is a crucial property in the proof of the next lemma.*

*Lemma 4.7 (Composition Lemma) Let D0 and D00 be two M-labeled proofs*

*of 0*

*` M : j ::: j and 00 ` M : j ::: j*

*respectively. Then there exists*

*1 n 1 m*

*an M -labeled proof*

*D : ` M : 1j ::: j nj 1j ::: j m*

*such that =k h 0 ; 00 i*

*n;m*

*Proof. By induction on the sum of the depths of D0 and D00 .*

*If D0 : 0 ;x : j ::: j*

*` j ::: j*

*and D00 : 00 ;x : j ::: j*

*` j ::: j*

*then*

*1 n 1 n*

*1 m 1 m*

*x : 1j ::: j nj 1j ::: j m 2*

*by de nition of.*

*Let (j w) be the last rule in D0 or in D00 , for example:*

*` M : 1j ::: j nj*

*D*

*:*

*0*

*` M : 1j ::: j n*

*j w*

*By the induction hypothesis there exists a proof of*

*k h 0 ; 00 i ` M : j ::: j j j j ::: j*

*n+1;m 1 n 1 m*

*Let p be a permutation of f1;::: ;m + n + 1g such that:*

*p(i) = i (1 i n);*

*p(n + 1) = n + m + 1;*

*p(n +1+ i) = n + i (1 i m);.*

*Using the Commutation Property for labeled proofs we obtain a proof of*

*p(k*

*h 0 ; 00 i) ` M : j ::: j*

*j j ::: j j*

*n+1;m 1 n 1 m*

*It is easy to verify that*

*p(k*

*h 0 ; 00 i) =k*

*h 0 ; 00 i*

*Then from the proof of*

*n+1;m*

*n;m*

*k h 0 ; 00 i ` M : j ::: j j j ::: j j*

*n;m 1 n 1 m*

*we obtain D by applying the (j w)-rule.*

*Let (^E) be the last rule in D0 or D00 , for example*

*` M : 1j ::: j n i = ^*

*D*

*:*

*0*

*` M : 1j ::: j n[i 7! ]*

*^E.*

*By induction hypothesis there is a proof of ` M : 1j ::: j nj 1j ::: j m,* then the (^E) rule can be applied on the i th component, and the thesis hold.

*For (^I) we proceed as for (^E). First two proofs are mixed, then (^I)* rule is applied.

*Let ! E be the last applied rule in D0 and D00 , that is*

*0 ` M : j ::: j 0 ` N : j ::: j*

*D0 : 1 n 1 n*

*(! E)*

*` MN : 1j ::: j n*

*0*

*00 ` M : 0 j ::: j 0 00 ` N : 0 j ::: j 0*

*D00 : 1 m 1 m (! E)*

*00 ` MN : 0 j ::: j 0*

*1 m*

*By using induction hypothesis we construct the proof D as*

*` M : j ::: j j 0 j ::: j 0*

*` N : j ::: j j 0 j ::: j 0*

*D : 1 n 1 m*

*1 n 1*

*m ! E.*

*` MN : j ::: j j 0 j ::: j 0*

*1 n 1 m*

*If (! I) is the last applied rule then we proceed as for (! E).*

*No other cases are possible.*

*2*

# *5 The Curry-Howard isomorphism between HL and T A^*

*Now we can prove that HL is isomorphic to T A^, which is the main goal of* this paper, using the Curry-Howard approach. The idea of this isomorphism is thus to interpret logical deductions as type-derivations and conversely, through the labeled proofs de ned in the previous section.

*A rst step toward this result is to de ne an interpretation of formulae* and hyperformulae as types. Formulae of De nition 3.1 can be read as types, mapping propositional variables, ! and ^ into type variables, arrow and intersection on types.

*Moreover, we read hyperformulae as types, by mapping the parallel con-* structor j into the ^ type constructor.

*We will write ' to denote the type so associated to '; for instance j ^ =*

*^ ^ . Since " denotes the empty formula, 'j" = '. Extending this mapping* to contexts (labeled contexts) we write () for denoting the set of types

*= f'j' 2 g ( = fx : 'jx : ' 2g).*

*Let us notice that, if ' = j and = ^ then ' = . Analogously, if*

*0 = fx : j g and 00 = fx : ^ g, then 0 = 00 = fx : ^ g*

*This is not surprising, because the treatment of the intersection in T A^ as a* proof-functional type constructor attens too many aspects in the (^I) typ- ing rule.

*Completely di erent, our quest for a logical system interpreting the ^ de-* rivability, without any metalinguistic constraint in the deduction rules, re- quired using two logical operator, namely j and ^. Therefore two assumptions of the form j and ^ , even if interpreted as the same type ^ , give raise to di erent proofs in HL.

*Theorem 5.1 (Types as proofs) If : B `^ M : then there exists an*

*M labeled proof D : ` M : such that = B.* Proof. By induction on .

*If : B; x : `^ x : then we construct the x labeled proof D as:*

*D :;x : ` x :*

*where is the mapping de ned by B [ fx : g.*

*If the last applied rule is (^I), i.e.*

*B `^ M : B `^ M : B `^ M : ^*

*(^I)*

*then, by induction hypothesis, there are two M labeled proofs D0 : 0 `*

*M : and D00 : 00 ` M : . Since both D0 and D00 are M labeled, then*

*we apply Composition-Lemma and we obtain an M labeled proof of*

*k h 0 ; 00 i ` M : j*

*1;2*

*where k1*

*h 0 ; 00 i = B, by construction. Finally*

*` M : ^ follows*

*;2*

*from Lemma 4.5 where = k h 0 ; 00 i = B, since the parallel operator j*

*is mapped into ^.*

*1;2*

*All the other cases, concerning (^E), (! I) and (! E), directly follow from* the induction hypothesis.

*2*

*Lemma 5.2 If D is an M labeled proof*

*D : ` M : '*

*then, for every i such that (')i 6= ", Bi `^ M : (')i, where Bi = ()i (1 i length(')).*

*Proof. By induction on D. The base case is straightforward. In the induc-* tive step, the cases (^E) and (jw) follow from the induction hypothesis.

*Let us consider the case when the last applied rule is the (! E). Then, by* induction hypothesis, Bi `^ M : i ! i and Bi `^ N : i, where Bi = ()i. Hence by using the typing rule (! E) we obtain the thesis Bi `^ MN : i.

*The proof is similar when the last applied rule is (! I) or (^I).*

*2*

*Theorem 5.3 (Proofs to types) If D : ` M : ' then there exists a* type derivation

*: B `^ M : '*

*such that B =.*

*Bi = ()i (1 i length(')). Then construct the basis B = SBi as Proof. By Lemma 5.2, Bi `^ M : (')i, for every i such that (')i 6= ", where*

*[Bi = fx : 1 ^ ^ njx : i 2 Big*

*It is trivial to verify that B `^ M : (')i and B =, hence B `^ M : ' by using the (^I) rule of T A^.*

*2*

*Theorem 5.4 (T A^ and HL are isomorphic) : B `^ M : if and only if D : ` ' where ' = and B = fx : j 2 g.*

*Proof. By Theorems 5.1 and Theorem 5.3. 2*

# *6 Conclusion and Future work*

*We presented a propositional logical calculus which formalizes a (metalogic)* notion of parallel deductions by using hyperformulae as parallel compositions of formulae.

*The main feature of this logic is that deduction rules are in the standard* shape of the natural deduction framework and do not involve any intensional requirement on the subproofs. We showed that this calculus is isomorphic to the intersection type assignment for lambda calculus, reading formulae and hyperformulae as types by the well known Curry-Howard paradigm. Namely, every deduction is associated to a type inference for a lambda term, where the term encodes the \history" of the deduction.

*As a result, the intersection type constructor , which comes out to be* proof-functional in the type theory, is interpreted as a standard propositional connective.

*The most interesting application of the logic we proposed here should be* the de nition of an explicitly typed lambda calculus with intersection types. This would be a very powerful functional language, in which all (and only) strongly normalizing terms-programs have a well-typed version. It is in this context that we are currently investigating such a typed lambda calculus. Deduction rules of HL and labeled proofs suggest a smooth solution to this further issue.

# *Acknowledgments*

*We would like to thank a group of Ph.D. students in Siena, C. Bianchini, D.* Luchi and S. Sadocco, for many helpful discussions and ideas that originated the notion of parallel deductions presented in this paper. We are grateful to F.Barbanera and M.Dezani for useful suggestions.

# *References*

*[1] A. Avron. Hypersequents, logical consequence and intermediate logics for concurrency. Annals of Mathematics and Arti cial Intelligence, 4(3{4):225{248, 1991.*

*[2] A. Avron. The method of hypersequents in the proof theory of propositional non- classical logics. In Wilfrid Hodges, Martin Hyland, Charles Steinhorn, and John Truss, editors, Logic: from foundations to applications. Proc. Logic Colloquium, Keele, UK, 1993, pages 1{32. Oxford University Press, New York, 1996.*

*[3] H. P. Barendregt. The Lambda-Calculus: Its Syntax and Semantics. North- Holland, Amsterdam, 1980.*

*[4] H. P. Barendregt. Lambda calculi with types. In D. M. Gabbai Samson Abramski and T. S. E. Maiboum, editors, Handbook of Logic in Computer Science. Oxford University Press, Oxford, 1992.*

*[5] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A lter lambda model and the completeness of type assignment. The Journal of Symbolic Logic, 48(4):931{940, December 1983.*

*[6] M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. Archive f. math. Logic u. Grundlagenforschung, 19:139{156, 1979.*

*[7] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, 27:45{58, 1981.*

*[8] C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, Englewood Cli s (NJ), USA, 1985.*

*[9] B. Venneri. Intersection types as logical formulae. Journal of Logic and Computation, 4(2):109{124, April 1994.*