Resource-oriented Programming Based on Linear Logic

Valerie Novitzká, Daniel Mihályi

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
valerie.novitzka@tuke.sk, daniel.mihalyi@tuke.sk

Abstract: In our research we consider programming as logical reasoning over types. Linear logic with its resource-oriented features yields a proper means for our approach because it enables to consider about resources as in real life: after their use they are exhausted. Computation then can be regarded as proof search. In our paper we present how space and time can be introduced into this logic and we discuss several programming languages based on linear logic.

Keywords: linear logic, programming languages, resource-oriented programming

1 Introduction

The aim of our research is to describe solving of large scientific problems by computers as constructive logical reasoning in some logical formal system over type theory. Predicates we consider as predicament-forming functors with nomenclative arguments (symbols), where unary predicates express properties and predicates of higher arity express relations. Using logical reasoning in categorical logic over type theory we can get a mathematical solution of a given problem and the existence of a proof in the intuitionistic linear logic provides us a computable solution of a problem [16, 17].

The role of computer program is to execute instructions under whose the computer system is to operate, to perform some required computations [15]. A running program should provide us a desirable solution of a given problem. We consider programming as a logical reasoning over axiomatized mathematical theories needed for a given solved problem. A program is intuitively understood as data structures and algorithms. Data structures are always typed and operations between them can be regarded as algorithms. Results of computations are obtained

by evaluation of typed terms. Due to discovering a connection between linear logic and type theory - a phenomenon of Curry-Howard correspondence [5], we are able to consider types as propositions and proofs as programs. Then we can consider a program as a logical deduction within linear logical system. Precisely, reduction of terms corresponding to proofs in the intuitionistic linear logic can be regarded as computation of programs. Thus computation of any resource-oriented program is some form of goal-oriented proof search in linear logic. One of the main goals of this approach is to avoid eventual mistakes of correctness generated by implementation of a programming language. This approach also keeps us away from potentional problems in verification of programs.

Linear logic (LL) was defined by Girard in [6,7] as resource-oriented logic, where formulae are actions. It enables reasoning similar as in real life, where resources are exhausted after their using. The basic ideas about LL we present in section 2. There are several programming languages based on LL and Section 2.2 contains the short disscussion about them. The rest of this paper presents the most important features of LL that enable to introduce resources as space and time into linear logic.

2 Linear Logic

LL is regarded as a continuation of the constructivisation that began with both classical and intuitionistic logic and it can serve as a suitable interface between logic and computer science [1]. Whereas classical logic treats sentences with stable values depended on Tarsky semantic tradition (i.e. interested in denotation of sentences), values of linear intuitionistic logic sentences depend on an internal state of a dynamic system according to Heyting semantic tradition (i.e. interested in constructing the proof of a given sentence).

2.1 Logical Connectives of LL

LL introduces new logical connectives. *Linear implication* $A \multimap B$ describes that an action A is a cause of action B; a formula A can be regarded as a resource that is exhausted by linear implication so as in real life. It is the most important feature of LL and also of programming based on LL. In clasical logic the truth value of formula A after the implication $A \rightarrow B$ remains the same. Classical implication can be rewritten in linear manner as $(!A) \multimap B$, where '*!*' is *modal operator* (*exponential*) expressing that we can use a resource A repeatebly, as many times as we need. LL defines *multiplicative conjunction* (MC), $A \otimes B$, expressing that both actions A and B will be done. *Additive conjunction* (AC), $A \,\&\, B$, expresses that only one of these actions will be performed and we can choose which one. *Additive disjunction* (AD), $A \oplus B$, also describes that only one action of A and B

will be performed but we do not know which one. And finally, *multiplicative disjunction* (MD), $A \wp B$, expresses: if A is not performed then B is done, or if B is not performed then A is done. MD is similar to disjunction in classical logic. *Neutral element* for MC is *1*, for AC is ⊤, for MD is ⊥, and for AD is *0*. *Linear negation,* $A^{\perp}$, is obtained by analogy with the dual spaces in algebra [9], as it is expressed in (1):

$$A \multimap B = B^{\perp} \multimap A^{\perp} \tag{1}$$

Negation is *involutive*, i.e. $(A^{\perp})^{\perp} = A$. An *entailment* in LL is a sequent of the form

$$\Gamma \vdash A \tag{2}$$

where $\Gamma = (A_1, \ldots, A_n)$ is a finite sequence of linear formulae and A is a linear formula deductible from the premises in Γ. If Γ is empty, then a formula is provable without assumptions. Inference rules of LL have a form

$$\frac{S_1 \ldots S_n}{S} \tag{3}$$

where $S_1, \ldots, S_n$, and S are sequents, $S_1, \ldots, S_n$ are assumptions and S is a conclusion of the rule. A proof in this calculus has the form of (proof-) tree, the direction of a proof is from-bottom-to-up, i.e. from the root to the leaves of the tree, where the root is the proved formula and leaves are axioms. In every proof step we can apply the inference rules of LL in Fig. 1 that introduce logical connectives and negation:

$$\frac{\vdash \Gamma, A, \Sigma}{\vdash \Gamma, A^{\perp}, \Sigma}(\perp - l) \qquad \frac{\vdash \Gamma, A, \Sigma}{\vdash \Gamma, A^{\perp}, \Sigma}(\perp - r)$$

$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \times B}(\otimes)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}(\oplus - l) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}(\oplus - r)$$

$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \,\&\, B}(\&) \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}(\parr)$$

Figure 1
Inference rules for LL connectives and negations

In LL are valid the following De Morgan laws:

$$(A \otimes B)^{\perp} = A^{\perp} \wp B^{\perp} \qquad (A \wp B)^{\perp} = A^{\perp} \otimes B^{\perp} \tag{4}$$

$$(A \,\&\, B)^{\perp} = A^{\perp} \oplus B^{\perp} \qquad (A \oplus B)^{\perp} = A^{\perp} \,\&\, B^{\perp} \tag{5}$$

Due to meaning of linearity, LL refuses *weakening*, i.e. the constant function $F(x)=a$ and *contraction*, i.e. the quadratic function $F(x)=G(x,x)$, but they can be reintroduced as logical rules by using modal operators. Just this restriction gives linear logic it's resource conscious nature. On the other side, due to reintroducing weakening and contraction as logical rules, proof symetry of sequents can be restored again.

2.2 Polarity in LL

Logical connectives of LL can be divided into two classes: *positive* and *negative* connectives. Girard [8] regarded positive connectives of LL as algebraic style and negative connectives of LL as logical style. Positive connectives and constants of LL are $\otimes$, $\oplus$, *1, 0* and negative ones are *&*, $\wp$, $\top$, $\bot$.

A formula of LL is positive if its outermost logical connective is positive; dually it is negative if its outermost logical connective is negative. A rule of LL calculus is *invertible* if it introduces a negative connective. Negative connectives are introduced by just one rule and the decomposition in proof from bottom-to-up is deterministic. From this it follows a very important consequence: we can get together several proof steps as a *single* step if we have a cluster of negative formulae. Dual property to invertibility is *focalisation* [2], it says that if we have a cluster of positive connectives called *synthetic connective,* we can perform corresponding rules simultaneously. The dual properties of invertibility/focalisation express associativity of logic that is the LL analogue of the Church-Rosser property of λ-calculus.

The *polarity* between positive/negative properties is general in LL. A cluster of rules with the same polarity can be performed as a single rule, by invertibility in the negative case, by focalisation in the positive case. So, the change of polarity in a proof means a new step in this proof and it can be consider as a *time incrementation*. In this manner we can introduce *time* into LL. If we forget truth of formulae and their content and we consider only their locations in proofs, then we can introduce *space* into LL and explicitly handle resources in LL. Due to the important property of invertibility and focalisation linear connectives can be organized by polarities. Summary of significant properties of LL connectives is shown in the Table 1.

2.3 Programming Languages Based on Linear Logic

LL forms a base for several functional and logic programming languages. Between functional programming languages [12] based on LL we mention Lilac [13] designed in 1994, but it is not wide-spread language.

Abbrev.	*Symb.*	*Descr.*	*Neutral*	*Polarity*	*NonDeter.*	*Human influence*
MC	$\otimes$	parallel	*1*	+		we know how
AC	&	choice	$\top$	-	internal	we know how
AD	$\oplus$	choice	*0*	+	external	we don't know how
MD	$\wp$	until	$\bot$	-		we know how

Table 1
Overview of linear connectives and constants

Logic programming may be viewed as the interpretation of logical formulae as programs and proof search as computations. Our view to logic programming is that computation is a kind of a proof search: let us have program P and goal G. We are trying to find a proof of the sequent $P \vdash G$. Different goals correspond to different computation sequences. The result of a computation with logic program is a proof that the goal is the logical consequence of the facts and rules. The search strategy is determined by the structure of the goal and the program supplies the context of the proof. The goal is 'active' whereas program is 'pasive' and provides a context in which the goal is executed. This principle is called goal-directed provability.

Whereas classical logic programming is based on the first order logic, the resource-oriented programming is based on LL. Resource-oriented program also consists of facts and rules and user-query starts a calculation that it answers the question whether query result belongs to a given program or not. But the difference is only that for performing any action in this logic a clause in a corresponding resource-oriented program must be used exactly once.

There are numerous programming languages that make use of all resource-oriented benefits of LL. In these languages, it is possible to create and consume resources dynamically as logical formulae. Lolli, Lygon, and Forum are implemented as interpreter systems; Lolli [10] is based on SML and Prolog, Lygon [18] is implemented over Prolog, and Forum [14] is based on SML, λProlog and Prolog. Nowadays we have good experiences with Lygon programming language, its implementation can be viewed as extended module over Prolog with full support of LL.

The programming languages ACL and HACL [11] introduce concurrent paradigm in LL programming. Every formula of LL is regarded as a process in some state, the proceses run concurently with asynchronous communication. Minerva is a commercional language based on Java with own development environment. It enables using LL, but also clasical logic features. The programming language Jinni (Java Inference eNgine and Networked Interactor) enables only a fragment of LL with linear implication, AC and AD and it makes a connection between object-oriented programming and logic programming.

3 Resource Handling in LL

In this section we present how it is possible to introduce the resources of space and time into LL. Our approach follows the famous idea published in [9] and it represents a novel approach to proof theory, where proofs are written in locative structure of Gentzen's sequent calculus.

Objects of linear logic are called *designs* and they play the role of proofs, λ-proofs, etc. in usual syntax. Designs are located somewhere. Therefore we need a concept of *location*. A design represents a cut-free proof of a LL formula *A,* in which all information has been erased, only locations in sequents are kept.

Let *A* be a LL formula. Immediate subformulae (w. r. t. focalisation) are denoted by natural numbers called *biases* written as *i,j,k,...* A finite set of *biases* is called *ramification*, denoted by *I,J,K,...* An address, *locus,* is a finite sequence $<i_1,...,i_n>$ of biases. We denote addresses by $\sigma,\tau,\upsilon,\xi,...$ A locus denotes a place or spatial location of a formula. If we have a formula *A* and its proof, then *A* is in the root of the proof tree. Formulae in this tree are subformulae and they have precise locations (absolute or relative to the root). But if some subformula occurs in the proof tree more times, then its every occurrence need to receive a distinct location. If locus of *A* is σ and *B* is a subformula of *A* with bias *i* then locus of *B* is $\sigma*<i>$. Let σ be a locus. Then $\sigma*\tau$, where symbol *'*'* denotes concatenation, is a sublocus of σ. Sublocus $\sigma*\tau$ is called

- *strict* if τ is non-empty sequence, $\tau \neq <>$;
- *immediate* if τ consists of just one bias, $\tau = <i>$.

We can say that a locus σ is in ordering relation *'≤'* with all its sublocuses, i.e. $\sigma \leq \sigma*<i>$.

Because in sequent calculus we proceed in a proof from its conclusion (root of proof tree), a locus occurs before its subloci w. r. t. time relation. Two loci can be comparable and incomparable (disjunct) w.r.t. relation *'≤'*. When two loci are

- *incomparable,* their relation is spatial, i. e. they are completely independent;
- *comparable,* their relation is timeable.

Ramification is needed for multiplicative rules where are two subformulae (assumptions) at the same time. We assume one-sided sequents of the form $\vdash \Gamma$ if all formulae in Γ are positive. If a sequent $\vdash A,B,C$ consists of two positive formulae *B,C* and a negative formula *A*, then we replace this one-side sequent by two-side one

$$A^{\perp} \vdash B,C \tag{6}$$

that consists only of positive formulae. Focalisation enables restriction to sequents with at most one formula on the left side.

Every formula in a sequent has a locus, i. e. a finite sequence of biases. If we forget everything in a sequent except loci of formulae, we get an expression called *pitchfork:*

$$\{\xi\} \vdash \Lambda \tag{7}$$

where $\{\xi\}$ is singleton containing one locus $\{\xi\}$ that can be empty set and Λ is a finite set of loci. A locus on the left side is incomparable with every locus in Λ. A pitchfork $\{\xi\} \vdash \Lambda$ consists of a *handle* ξ and the *tines* (loci) in Λ. A pitchfork is *positive* if it has no handle and *negative* if it has a handle. A pitchfork is *atomic* if it has a form $\xi \vdash$, or $\vdash \xi$.

These definitions enable to introduce space and time into LL. In the following example we show how it can be done for a LL formula.

Example: Let $A=((P^{\perp} \oplus Q^{\perp}) \otimes R^{\perp})$ be a LL formula with *P, Q, R* positive formulae. We can construct the following three proofs of *A*. Proofs 1 and 2 differs only in using left- or right- rules for introducing AD, in the 3 proof we firstly rewrite *A* using De Morgan rules and then we build its proof.

1.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash \Gamma}(id)}{\vdash P^{\perp}, \Gamma}(\perp - r)}{\vdash P^{\perp} \oplus Q^{\perp}, \Gamma}(\oplus - l) \quad \dfrac{\dfrac{}{R \vdash \Delta}(id)}{\vdash R^{\perp}, \Delta}(\perp - r)}{\vdash ((P^{\perp} \oplus Q^{\perp}) \otimes R^{\perp}), \Gamma, \Delta}(\otimes)$$

(-) (+)

2.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{Q \vdash \Gamma}(id)}{\vdash Q^{\perp}, \Gamma}(\perp - r)}{\vdash P^{\perp} \oplus Q^{\perp}, \Gamma}(\oplus - r) \quad \dfrac{\dfrac{}{R \vdash \Delta}(id)}{\vdash R^{\perp}, \Delta}(\perp - r)}{\vdash ((P^{\perp} \oplus Q^{\perp}) \otimes R^{\perp}), \Gamma, \Delta}(\otimes)$$

(-) (+)

3. Using De Morgan rules we can write

$$A=((P^{\perp} \oplus Q^{\perp}) \otimes R^{\perp}) = ((P \oplus Q)^{\perp} \otimes R^{\perp}) = ((P \oplus Q) \parr R)^{\perp}.$$

Then the proof tree is:

3.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash P, R, \Lambda}(id) \quad \dfrac{}{\vdash Q, R, \Lambda}(id)}{\vdash (P^{\perp} \& Q^{\perp}), R, \Lambda}(\&)}{\vdash ((P^{\perp} \& Q^{\perp}) \parr R^{\perp}), \Lambda}(\parr)}{A \vdash \Lambda}(\perp - l)$$

(+) (-)

The left-side sign '*(-)*' in proofs 1 and 2 denotes the place where polarity changes from positive to negative. In the proof 3 the left-side sign '*(+)*' denotes the place where polarity changes from negative to positive. We can form clusters of the rules with same polarity for these proof trees and perform them as the following (single) rules that express *time incrementation* in proofs:

1. 2.

$$\frac{P \vdash \Gamma \qquad R \vdash \Delta}{\vdash A, \Gamma, \Delta}(r1) \qquad \frac{Q \vdash \Gamma \qquad R \vdash \Delta}{\vdash A, \Gamma, \Delta}(r2)$$

3.

$$\frac{\vdash P, R, \Lambda \qquad \vdash Q, R, \Lambda}{A \vdash \Lambda}(r3)$$

Now we forget everything except locations of formulae above. We assume that the locus of *A* is *ξ*, and biases for *P, Q, R* are *3, 4, 7,* respectively. We note that *Γ, Δ, Λ* are no longer formulae but loci. Then we can rewrite previous rules and we get explicit information about space resources needed for proofs. Every sequent in these proofs is a pitchfork:

1. 2.

$$\frac{\xi 3 \vdash \Gamma \qquad \xi 7 \vdash \Delta}{\vdash \xi, \Gamma, \Delta}(\xi r1) \qquad \frac{\xi 4 \vdash \Gamma \qquad \xi 7 \vdash \Delta}{\vdash \xi, \Gamma, \Delta}(\xi r2)$$

3.

$$\frac{\vdash \xi 3, \xi 7, \Lambda \qquad \vdash \xi 4, \xi 7, \Lambda}{\xi \vdash \Lambda}(\xi r3)$$

□

From this example we can see that we can represent any LL cut-free proof as pitchfork rules. In all last three rules we explicitly treat with space and time. This approach creates the new possibilities in logic programming and enables to see logic as a means how to precisely reason about real life problems and solved them by computers.

Conclusions

In our paper we tried to present foundations of the new approach of logic and logic programming. LL enables to handle resources in explicit way, what is the main disadvantage of classical logic used in logical programming. This idea is very recent and it enables to see logic, namely LL, as a logic for reasoning close to real life. In problem solving we are thinking in notions of space and time and there are the most important notions also in programming computers. These concepts form new criteria also for programming languages based on logic and we beliefe that this approach will create a new age of logical reasoning and logic programming.

Acknowledgement

This work was supported by VEGA Grant No. 1/2181/05: Mathematical Theory of Programming and Its Application in the Methods of Stochastic Programming.

References

[1] Abramsky S.: Computational Interpretations of Linear Logic, in Theoretical Computer Science, 1992, pp. 1-53

[2] Andreoli J. M.: Proposition pour une synthése des paradigmes de la programation logique et de la programmation par objects, Thése d'informatique de l'Université de Paris VI, 1990

[3] Bierman G. M.: On Intuitionistic Linear Logic, Tech. Rep. No. 346, University of Cambridge, 1994

[4] Faggian C., Hyland J.: Designs, Disputes and Strategies, In Proccedings of CSL '02, 2002, pp. 1-21

[5] Girard J. Y.: Proofs and Types, Cambridge University Press, 2003, pp. 1-175

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

[7] Girard J. Y.: Linear Logic: Its Syntax and Semantics, In: J. Y. Girard, Y. Lafont, and L. Regnier, editors, Advances in Linear Logic, Cambridge, 1995, pp. 1-42

[8] Girard J. Y.: Locus Solum. Mathematical Structures in Computer Science, Vol. 11, 2001, pp. 301-506

[9] Girard J. Y.: On the Meaning of Logical Rules I: Syntax vs. Semantics. In Computational Logic, U. Berger and H. Schwichtenberg, Eds., NATO ASI Series 165, Vol. 14. Springer-Verlag, New York, 215-272

[10] Hodas J. S, Miller D.: Logic Programming in a Fragment of Intuitionistic Linear Logic. Information and Computation, Vol. 110, No. 2, 1994, pp. 327-365

[11] Kobayashi N., Yonezawa A.: Typed Higher-Order Concurrent Linear Logic Programming. Technical Report 94-12, University of Tokyo, 1994

[12] Kozsik Tamás: Tutorial on Subtype Marks. Z. Horváth (Ed.) Central European Functional Programming School (The First Central European Summer School, CEFP 2005, Budapest, Hungary, July 4-15, 2005), Rev. Selected Lectures. LNCS 4161. Springer-Verlag, 2006, pp. 191-222

[13] Mackie I.: Lilac – a Functional Programming Language Based on Linear Logic, J. of Functional Programming, Vol. 4, No. 4, 1994, pp. 395-433

[14] Miller D.: A multiple-Conclusion Specification Logic. Theoretical Computer Science, 165(1):201-232, 1996

[15] Novitzká V.: Formal Foundations of Correct Programming, Advances in Linear Logic, Elfa, 1999, pp. 1-70

[16] Novitzká V., Mihályi D., Slodičák V.: Linear Logical Reasoning on Programming, Acta Electrotechnica et Informatica, 6, 3, 2006, pp. 34-39, ISSN 1335-8243

[17] Novitzká V., Mihályi D., Slodičák V.: How to Combine Church's and Linear Types, ECI'2006, Košice - Herľany, September 20-22, 2006, Košice, 2006, 6, pp. 128-133, ISBN 80-7099-879-2

[18] Winikoff M. D.: Logic Programming with Linear Logic, PhD. Thesis, Univ. Melbourne, 1997