

# Limitations of logic programming

Herman Ruge Jervell

`www.uio.no/~herman`  
`herman.jervell@ilf.uio.no`  
University of Oslo  
Norway

**Abstract.** We show that logic programming built upon predicate logic is intimately connected with the Kalmar elementary functions.

## 1

The main limitations of any type of programming is connected with what one can describe in a formal language. A computer can be considered as a syntax machine and a computation as a sequence of syntactical manipulations which the machine does. Now we know as logicians

- not all knowledge can be described syntactically
- even fewer parts of our knowledge can be captured within syntactic calculi
- the parts which can be successfully described using syntactical calculi are often using civilizational constructs — money, transportation, bureaucracies, industrial processes . . .

There is no reason that we can describe fully the limitations of programming in terms of the present and future world we humans live in.

Our theme here is much simpler. We disregard all the problems concerning formal languages and syntactical calculi. We consider purely formal problems and ask whether they can be successfully treated within logical programming built upon predicate logic. Then we shall see that the problems where logical programming help are closely connected with the Kalmar elementary functions. Any problem that can be successfully treated can be described in an elementary way. All problems described in an elementary way can be successfully treated in logical programming, if they can be treated at all. The key to the successful treatment is to give to the logical programs appropriate cuts making the execution feasible relative to the size of input and output.

## 2

Our underlying logic is predicate calculus with equality. Later we shall see that we can avoid equality. But the equality makes the development here more perspicuous. Our calculations are done within a datastructure. Here we take it to be the unary numbers. Then the logical language consists of

- a unary predicate  $\mathcal{N}$
- the constant 0 and the unary successor function  $s$
- a number of other function symbols

As axioms for the datastructure we have

$$0 : \mathcal{N}$$

$$\forall x : \mathcal{N}. sx : \mathcal{N}$$

In addition we may have a number of axioms not involving  $\mathcal{N}$ . We call these notational axioms. As typical examples we have

$$+0y = y$$

$$+sxy = s + xy$$

$$\star 0y = 0$$

$$\star sxy = +y \star xy$$

$$p0 = 0$$

$$psx = x$$

$$\dot{-}x0 = x$$

$$\dot{-}xsy = p\dot{-}xy$$

$$e0y = sy$$

$$esxy = exexy$$

*We require that the notational axioms are true in the standard interpretation.* This means that the notational axioms are true in the unary numbers when the function symbols above are interpreted as addition, multiplication, predecessor, modified subtraction and exponentiation. In addition there may be notational laws. It will turn out later that we need the following

$$\dot{-}0y = 0$$

$$\dot{-}sxy = 0 \vee \dot{-}sxy = s\dot{-}xy$$

### 3

What are the requirements for systems for logical programming? We put down four principles

**Principle 1 (Soundness of execution)** *Every program corresponds to a statement in our logical language, and every execution corresponds to a derivation of the statement from the axioms for the datastructure and for the notations. The derivation is in classical predicate calculus with equality.*

**Principle 2 (Feasibility of execution)** *Given a program. Then there is a feasible function which to the inputs in the program give the corresponding derivation.*

For our purposes here we may be so liberal as to assume that the function is Kalmar elementary. But we may also assume the function to be polytime and get similar results.

**Principle 3 (Soundness of notation)** *All notational axioms are true in the standard interpretation.*

**Principle 4 (Transparency of notation)** *All  $t$  used can be shown to be well defined. That is we should be able to show  $t : \mathcal{N}$ .*

It may seem that there is a restriction in assuming that the underlying data-structure is the unary numbers. This is not so. We can for example get notation for the binary numbers by using

$$\begin{aligned} 1 &= s0 \\ fx &= +xx \\ gx &= s + xx \end{aligned}$$

Then the binary numbers corresponds to  $1, f1, g1, ff1, gf1, fg1, gg1, \dots$  — where we reads the terms as binary numbers with  $f$  as digit 0 and  $g$  as digit 1 and the terms read in the opposite way — from right to left. The old arithmetical operation — adding 1 to a binary number — can be done with

$$\begin{aligned} c1 &= f1 \\ cfx &= gx \\ cgx &= fcx \end{aligned}$$

and binary addition

$$\begin{aligned} ax1 &= cx \\ a1y &= cy \\ afxfy &= faxy \\ afxgy &= gaxy \\ agxfy &= gaxy \\ agxgy &= fcaxy \end{aligned}$$

Similarly for other operations where it seems essential to use other data-structures than unary numbers to get fast programs.

## 4

All the terms that we can use in our notations must be Kalmar elementary [8]. This is a consequence of the cut elimination theorem. Let us spell this out. Assume we have a term

$$txy$$

which we can prove to be well defined. Then by feasibility of execution there must be Kalmar elementary functions

$$Pxy$$

which to numerals  $m, n$  give a derivation  $Pmn$  showing that

$$tmn : \mathcal{N}$$

The usual cut elimination theorem [9] gives new elementary functions  $P^*xy$  which to numerals  $m, n$  give cut free derivation of  $tmn : \mathcal{N}$ . The cut free derivation can only go through equality axioms, the axiom  $0 : \mathcal{N}$  and using  $\forall x : \mathcal{N}. sx : \mathcal{N}$ . Soundness of notations shows that there are only rewriting between numerically equal terms. The only way to have a cut free derivation of  $tmn : \mathcal{N}$  is to use the successor axioms as many times as the numerical value of  $tmn$ . But then we can find in an elementary way the numerical value of  $tmn$  from  $m, n$ .

**Theorem 1.** *All well defined terms in logical programming are Kalmar elementary.*

## 5

We now want to show that all Kalmar elementary terms are well defined in logical programming. This is a new result [3]. The crucial new concepts are

**Definition 1.** *A unary predicate  $\mathcal{P}$  is inductive if it is similar to  $\mathcal{N}$ , i.e. the following is provable —  $0 : \mathcal{P}$  and  $\forall x : \mathcal{P}. sx : \mathcal{P}$ . A term  $txy$  is inductive with  $x$  as induction variable if there are inductive predicates  $\mathcal{P}_0$  and  $\mathcal{P}_1$  making the following inductive*

$$x : \mathcal{P}_0 \wedge \forall y : \mathcal{P}_1. txy : \mathcal{N}$$

*Similar for terms with one or more arguments.*

It is straightforward to prove that the following is inductive

$$\begin{aligned} x : \mathcal{A} &= x : \mathcal{N} \wedge \forall y : \mathcal{N}. +xy : \mathcal{N} \\ x : \mathcal{M} &= x : \mathcal{N} \wedge \forall y : \mathcal{A}. \star xy : \mathcal{N} \\ x : \mathcal{E} &= x : \mathcal{N} \wedge \forall y : \mathcal{N}. exy : \mathcal{N} \end{aligned}$$

This shows that addition, multiplication and exponentiation are inductive. We must be a little careful to show that modified subtraction is inductive. To this we need the following equational axioms

$$\begin{aligned}\dot{-}0y &= 0 \\ \dot{-}sxy &= 0 \vee \dot{-}sxy = s\dot{-}xy\end{aligned}$$

We use  $x$  and not  $y$  as induction variable. This is the only formula involving  $\vee$  in our theory. Everything else involves only  $\wedge, \rightarrow, \forall$ . We then get that modified subtraction is inductive

$$x : \mathcal{S} = x : \mathcal{N} \wedge \forall y : \mathcal{N}. \dot{-}xy : \mathcal{N}$$

Assume we have inductive predicates  $\mathcal{F}$  and  $\mathcal{G}$  with the same induction variable. The conjunction  $\mathcal{F} \wedge \mathcal{G}$  is defined as usual. The composition  $\mathcal{F}[\mathcal{G}]$  is defined by substituting the predicate  $\mathcal{G}$  for  $\mathcal{N}$  in  $\mathcal{F}$ . The inductive predicates are obviously closed under conjunction. Since the only axioms involving  $\mathcal{N}$  are similar to the proved formulas of the inductive predicates, we also get that inductive predicates are closed under composition.

For composition of inductive terms assume we have the following inductive predicates

$$\begin{aligned}x : \mathcal{F} &= x : \mathcal{F}_0 \wedge \forall y : \mathcal{F}_1. fxy : \mathcal{N} \\ u : \mathcal{G} &= u : \mathcal{G}_0 \wedge \forall v : \mathcal{G}_1. guv : \mathcal{N}\end{aligned}$$

Then the following are also inductive

$$\begin{aligned}x : \mathcal{F}_0 \wedge \forall u : \mathcal{G}[\mathcal{F}_1]. \forall v : \mathcal{G}_1[\mathcal{F}_1]. fxguv : \mathcal{N} \\ u : \mathcal{G}_0[\mathcal{F}] \wedge \forall v : \mathcal{G}_1[\mathcal{F}]. \forall y : \mathcal{F}_1. fgvy : \mathcal{N}\end{aligned}$$

For diagonalization we start with the inductive predicate

$$x : \mathcal{F}_0 \wedge \forall y : \mathcal{F}_1. \forall z : \mathcal{F}_2. fxyz : \mathcal{N}$$

Then the following are also inductive

$$\begin{aligned}x : \mathcal{F}_0 \wedge \forall y : \mathcal{F}_1 \wedge \mathcal{F}_2. fxyy : \mathcal{N} \\ x : (\mathcal{F}_0 \wedge \mathcal{F}_1) \wedge \forall z : \mathcal{F}_2. fxxz : \mathcal{N}\end{aligned}$$

For bounded sums and products we start with the inductive predicates  $\mathcal{A}$  and  $\mathcal{M}$  for addition and multiplication and the inductive predicate

$$x : \mathcal{F} = x : \mathcal{F}_0 \wedge \forall y : \mathcal{F}_1. fxy : \mathcal{N}$$

Then the following are also inductive

$$\begin{aligned}z : \mathcal{F}[\mathcal{A}] \wedge \forall y : \mathcal{F}_1[\mathcal{A}]. \Sigma_{x < z} fxy : \mathcal{N} \\ z : \mathcal{F}[\mathcal{M}] \wedge \forall y : \mathcal{F}_1[\mathcal{M}]. \Pi_{x < z} fxy : \mathcal{N} \\ z : \mathcal{F}_1[\mathcal{A}] \wedge \forall x : \mathcal{F}[\mathcal{A}]. \Sigma_{y < z} fxy : \mathcal{N} \\ z : \mathcal{F}_1[\mathcal{M}] \wedge \forall x : \mathcal{F}[\mathcal{M}]. \Pi_{y < z} fxy : \mathcal{N}\end{aligned}$$

**Theorem 2.** *All Kalmar elementary terms are inductive.*

**Theorem 3.** *The Kalmar elementary terms are exactly the feasibly well defined terms.*

Furthermore the inductive formulas are constructed from the basic inductive formulas  $\mathcal{N}$  using conjunction, composition and universal quantification over inductive formulas and terms from the language.

As cut formulas in the proof of feasibly well definedness of a term we use the inductive formulas which we get in the construction of the term.

## 6

Let us look closer at the inductive terms. With two arguments we have defined them as terms  $txy$  where we have inductive formulas  $\mathcal{F}_0, \mathcal{F}_1$  such that the following is inductive

$$x : \mathcal{F}_0 \wedge \forall y : \mathcal{F}. txy : \mathcal{N}$$

Now Gentzen has shown [2] that all provable formulas in Peano arithmetic are closely connected with the inductive formulas. Assume we have a proof in Peano arithmetic of a certain formula  $B$

$$\vdash_{PA} B$$

Then there are a finite number of application of the induction axiom

$$0 : F_i \wedge \forall x : F_i. sx : F_i \text{ where } i \leq N$$

Let us define a new formula

$$Fx = \forall y. \bigwedge_{i \leq N} (y = i \rightarrow x : F_i)$$

We then show in predicate logic using our axioms

- $Fx$  is inductive
- $\vdash \forall x. Fx \rightarrow B$

But here there are no other way to find the appropriate substitutions of  $x$  than looking at the derivation in Peano arithmetic of  $B$ . In the case of the inductive terms the substitutions come from the input arguments.

## 7

Equality can be eliminated at the expense of perspicuity. To do this we

- use relation symbols instead of function symbols
- replace equality with appropriate implications

For addition and multiplication we introduce two ternary relation symbols  $A$  and  $M$  and have as axioms

$$\begin{aligned} & A0yy \\ & Axyz \rightarrow Asxysz \\ & M0y0 \\ & Mxyz \wedge Ayzu \rightarrow Msxyu \end{aligned}$$

Similarly for the other function symbols. The inductive predicates can then be given in the form

$$x : \mathcal{F}_0 \wedge \forall y : \mathcal{F}_1. \forall z : \mathcal{N}. Rxyz$$

where  $x$  is the induction variable,  $x, y$  input variables and  $z$  output variable. We then go through the same proof as before showing that to every Kalmar elementary function there is a corresponding inductive relation.

## 8

The results here can be used in interactive proofs.

**Definition 2.** *An interactive proof system (for predicate logic)  $(A, B)$  is a protocol between Alice and Bob. Alice runs a logical programming system  $A$  following our four principles, while Bob operates a polynomial-time randomized algorithm  $B$ . The input to the protocol is a statement known to both algorithms. The two exchange a sequence of messages  $m_1, m_2, m_3, \dots$  where Alice sends the odd-numbered ones and Bob the even ones. All messages are polynomial in length.*

This definition is similar to the one in [7]. We now let the polynomial randomized Bob propose cut formulas. Alice runs a logical programming system. The randomized Bob does not need to know the recursive definition of the Kalmar elementary terms. He just tries to guess what a cut formula should be. But then it is clear that

**Theorem 4.** *Any Kalmar elementary term can be shown to be well defined in the interactive proof system.*

Conversely it is clear that the only help that Bob can give Alice in an interactive proof system for predicate logic is to propose possible cut formulas. But then we get the converse

**Theorem 5.** *A term which can be shown to be well defined in an interactive system for predicate logic is Kalmar elementary.*

The starting point has been the examples of Orevkov[5] and Zhang[10]. They showed that the terms built up from exponentiation as

$$eeeeeeee000000000$$

could be given a short proof of well definedness using appropriate cut formulas. But if we restrict ourselves to propositional cuts then the proofs must be larger than the universe.

The usual cut elimination theorem gives a super exponential jump in length. We have designed our system such that there are no cut free proof of well definedness of a closed term of length less than the numerical value of the term. If we use propositional cuts we may gain an exponential speed up in the length, but not more.

Bellantoni and Cook[1] has given a system for polytime arithmetic using a restricted form of induction. They differentiated between induction arguments and substitution arguments — calling them normal and safe arguments. Our induction variables are similar to normal arguments — we are not free to substitute arbitrary values in them. Leivant[4] generalized this to a tiered hierarchy and went from the polytime functions to the Kalmar elementary functions. There is also recent work by Ostrin[6]. From their work a theorem like our main theorem was expected.

Our results give concrete feasible proofs using cut formulas. As an example consider the factorial. An inductive formula for the factorial  $f$  is

$$\begin{aligned} & x : \mathcal{M} \wedge fx : \mathcal{N} \\ \Leftrightarrow & \quad \forall y. (y : \mathcal{A} \rightarrow \star xy : \mathcal{N}) \wedge fx : \mathcal{N} \\ \Leftrightarrow & \forall y. (\forall z : \mathcal{N}. + yz : \mathcal{N} \rightarrow \star xy : \mathcal{N}) \wedge fx : \mathcal{N} \end{aligned}$$

And as cutformulas to get a short proof that the factorial is well defined we use subformulas of it. Similar calculations can be done for all Kalmar elementary functions. Our framework can also be used to define the Ackermann function  $a$  and the exponential tower function  $t$

$$\begin{aligned} a0y &= sy \\ asx0 &= axs0 \\ asxsy &= axasxy \\ t0 &= 0 \\ tsx &= etx0 \end{aligned}$$

But there are no short proof in predicate logic that they are well defined.

## References

1. Stephen Bellantoni and Stephen Cook. *A new recursion-theoretic characterization of the poly-time functions*. Computational Complexity, 2: 97-110, 1992.
2. Gerhard Gentzen. *Zusammenfassung von mehreren vollständigen Induktionen zu einer einzigen*. Published posthumously Archiv für mathematische Logik und Grundlagenforschung, 2, pp 1-3. 1954.
3. Herman R Jervell, Wenhui Zhang. *Cut Formulas for Kalmár Elementary Functions*. Submitted LICS2000.
4. Daniel Leivant. *Stratified functional programs and computational complexity*. Proceedings of POPL'93. 1993
5. V P Orevkov. *Lower bounds for lengthening of proofs after cut-elimination* (Russian). Zapiski Nauchnykh Seminarov Leningradskogo Obdeleniya Ordena Lenina Matematicheskogo Instituta imeni V. A. Steklova Akademii Nauk SSSR (LOMI), 88, 137-162, 242-243. Translation Journal of Soviet Mathematics 20 (1982), 2337-2350.
6. Geoffrey Ellis Ostrin. *Proof Theories of Low Sub-Recursive Classes*. Dissertation. Leeds 1999.
7. Christos H Papadimitriou. *Computational Complexity*. Addison-Wesley. 1994.
8. H E Rose. *Subrecursion. Functions and Hierarchies*. Oxford Logic Guides 9. Clarendon Press. Oxford 1984.
9. A S Troelstra, H Schwichtenberg. *Basic Proof Theory*. Cambridge Tracts in Theoretical Computer Science 43. Cambridge University Press. Cambridge 1996.
10. Wenhui Zhang. *Cut elimination and automatic proof procedures*. Theoretical Computer Science 91(2):265-284. 1991.