ON OPTIMIZING PROOF SEARCH IN LINEAR LOGIC BY VALUE ITERATION METHOD

[*]Valerie NOVITZKÁ, [**]Anita VERBOVÁ
[*]Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, tel. 055/602 4182, E-mail: Valerie.Novitzka@tuke.sk
[**] Department of Computers and Informatics, , Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, tel. 055/602 4144, E-mail: Anita.Verbova@tuke.sk

SUMMARY

In this article we are interested in proving of linear logic sequents. In linear sequent calculus one sequent can have more than one proof tree. We choose the best among them satisfying some criterion. There can occur some non-deterministic choices in the process of building the proof of a sequent. We introduce probabilities to these proof constructions. Therefore we can apply one method of stochastic programming (value iteration method) to determine the optimal way in the proof search.

Keywords: *stochastic programming, value iteration method, non-determinism, linear logic, proof search.*

1. INTRODUCTION

Within the context of the construction of correct programs, some applications are based on the following paradigms: *proofs-as-programs* and *proof search-as-computation*. These paradigms are called the Curry-Howard correspondence that associates a λ-term to a proof in intuitionistic logic [7]. From a given specification, expressed in the given logic, we can construct a proof by the first paradigm mentioned above and extract a program from this proof. The second paradigm focuses on the proof construction: the proof search process corresponding to computation. Thus the common problem is to be able to construct proofs in the given logic. Here we consider linear logic (LL) that has some applications to computation, proof construction, for concurrent or functional programming [12].

Many works on linear logic and their applications to computer science [1, 4, 5, 11, 13] involve methods and techniques to deal with the problem of proof construction in linear sequent calculi. We propose algorithms and techniques by the methods of stochastic programming for proof construction in linear logic.

In linear logic proof construction there are non-determinisms like the non-deterministic selection between rules for different connectives and the non-deterministic choice in the case of applying the rule for the $\oplus$ connective, where we cannot affect the choice of the rule $\oplus_1$ or $\oplus_2$ and also for the $\otimes$ connective, where we have 2^n ways of partitioning the context. $\oplus$ is called an external choice. On the other hand there exists an internal choice &, which expresses that we can control this choice. We introduce probability to the linear logic rules. We define the optimal strategy using value iteration method, which is a method of *stochastic programming,* that is a framework for modeling optimization problems that involve uncertainty.

2. PROOF SEARCH

Linear logic has been introduced by Girard as *resource-sensitive* refinement of classical logic. It is a strong system of logic and it has full features of first order and intuitionistic logic and additionally supports concepts of disposable resources and their *consumptions*. Linear logic provides a mechanisms to destroy and construct formulas in the process of proving, where formulas represent actions.

In classical logic, there is one conjunction ($\wedge$) and one disjunction ($\vee$); in linear logic, there are two of each (conjunctions: $\wp$, $\otimes$; disjunctions: &, $\oplus$). We are using one sided sequents to reduce the number of rules considered. The linear logic connectives and rules are presented in [6].

In linear logic proof search Andreoli formulated in [2] the problem of the principal formula. In his procedure SEARCH when an inference rule is to be applied at a given node, two choices must be made:

1. choice of a (non atomic) principal formula in the sequent at that node;
2. choice of an instance of the inference rule associated with the topmost connective of the selected principal formula.

Although all forms of "don‘t know" non-determinism cannot be eliminated in these choices, a definite permutation of inference rules shows that some of these choices are not significant and either need not be considered at all or could be treated deterministically ("don‘t care" non determinism).

Linear connectives are divided into two groups which behave differently with respect to the choice of the principal formula.

- The "asynchronous" connectives:
  - Multiplicative: $\bot$, $\wp$, ?
  - Additive: $\top$, &, $\forall$

- The "synchronous" connectives:
  - Multiplicative: 1, ⊗, !
  - Additive: 0, ⊕, ∃

The dual of an asynchronous connective is synchronous and vice versa. A non-atomic formula whose topmost connective is synchronous (resp. asynchronous) is called a synchronous (resp. asynchronous) formula. The difference in search behavior between these two groups can be characterized as follows.

If the principal formula which has been selected in a sequent is *asynchronous*, then there is *one and only one* applicable instance of the corresponding inference rule, whereas if it is *synchronous*, one among *several* (or sometimes no) instances has to be selected.

Thus, if the synchronous formula $A \otimes B$ is selected as principal formula in the sequent $\vdash \Gamma$, $A \otimes B$, many possible instances of the corresponding inference rule $\otimes$ can be applied, corresponding to the different partitions of Γ along the two branches. Similarly, a principal formula of the form $A \oplus B$ requires the choice between the left $\oplus_1$ and right $\oplus_2$ instances of the corresponding inference rule. On the other hand, when an asynchronous formula is selected as principal formula, there is a unique applicable instance of the corresponding inference rule and its application is therefore deterministic.
Andreoli summarized these properties as:

- Asynchronous → Determinism
- Synchronous → Non-determinism

He proposed a proof normalisation, which can be summarized as follows:

- If the sequent contains some *asynchronous* formulae (at least one), then any one of them can be *immediately* and *randomly* selected as the principal formula ("don‘t care" non-determinism). Furthermore, as the formula thus selected is by hypothesis asynchronous, the instance of inference rule to apply is completely determined. Consequently, as long as the sequent contains an asynchronous formula, the search can be made completely *deterministic*.
- When all the asynchronous formulae have been decomposed, then a principal formula must be selected non deterministically. But, as soon as one formula has been selected, the search can *focus* on it, i.e. subsequently select systematically the subformula stemming from the initial one as principal formula, *and do so as long as this subformula is synchronous*.

Asynchronous formulae are decomposed immediately as soon as they appear in the sequent (hence their name "asynchronous"). Synchronous formulae are delayed until all the asynchronous formulae have been decomposed, and must be non deterministically selected to be processed; in other words, synchronous connectives synchronize the selection process and the decomposition process (hence their name "synchronous"). But once a synchronous formula starts being decomposed, it keeps on being decomposed till a non synchronous (i.e. atomic or asynchronous) formula is reached. This means that in a normal proof, each formula is viewed as a succesion of layers of asynchronous connectives and of synchronous connectives; each synchronous layer is decomposed in a *critical section,* i.e. which cannot be interrupted. It is called a "critical focusing section" of the proof.

"Don‘t know" non-determinism appears in the search only during the critical focusing section, which involve synchronous connectives (asynchronous connectives generate only "don‘t care" non-determinism). However, non-determinism can be considerably reduced by the following condition imposed on normal proofs. Let`s partition arbitrarily the atomic formulae into two dual disjoint classes: positive atoms X and negative atoms $X^{\perp}$. In a normal proof, when a critical focusing section reaches a negative atom, then the inference rule of Identity (id) *have to* be applied. This condition reduces the amount of non-determinism involved in the critical sections.

3. OPTIMAL PROOF SEARCH USING STOCHASTIC PROGRAMMING

In this section we apply stochastic programming to determine the optimal strategy for linear logic proof search. We can use Markov decision process (MDP) because it models decision making in situations where results are partly random and partly influenced by the decision maker. This holds in linear logic proof construction where we can sometimes force the way of building the proof but in the non-deterministic cases as ⊕ and ⊗, we have no permission to select one of some possible ways to continue. Therefore we assign probabilities to these actions, which represent the rules, which we can apply in each stage of the proof construction and we compute the most probable strategy by value iteration method.

A **stochastic process**, or sometimes random process, is the counterpart of a deterministic process (or deterministic system) considered in probability theory. Instead of dealing only with one possible “reality” of how the process might evolve under time, in a random process there is some *indeterminacy* in its future evolution described by probability distributions. This means that even if the initial condition (or starting point) is known, there are more possibilities the process might go to, but some paths *are more probable* and others less.

In the case of *discrete time*, a stochastic process amounts to a sequence of random variables known as a time series (for example Markov chain). In probability theory, a stochastic process has the Markov property if the conditional probability distribution of future states of the process, given the present state and all past states, depends only upon the *present state and not on any past states*, i.e. it is

conditionally independent of the past states (the path of the process) given the present state. A process with the Markov property is usually called a *Markov process*.

Markov decision processes (MDPs) provide a mathematical framework for modelling decision-making in situations where outcomes are *partly random and partly under the control* of the decision maker. MDPs are useful for studying a wide range of optimization problems solved via dynamic programming.

Stochastic programming [9] is a framework for modeling optimization problems that involve uncertainty. Its models are similar in style but take advantage of the fact that probability distributions governing the data are known or can be estimated. The goal here is to find some strategy that is feasible for all (or almost all) the possible data instances and maximizes the expectation of some function of the decisions and the random variables. More generally, such models are formulated, solved analytically or numerically, and analyzed in order to provide useful information to a decision-maker.

3.1. Markov Decision Process

Consider a system being observed over a finite or infinite time horizon split up into periods or stages. At each stage, the *state* of the system is observed, and a *decision* (or an *action*) concerning the system has to be made. The decision influences (*deterministically or stochastically*) the state to be observed at the next stage, and depending on the state and the decision made, an immediate *reward* is gained. The expected total rewards from the present stage until the end of the planning horizon is expressed by a *value function*. The relation between the value function at the present stage and the one at the following stage is expressed by the *functional equation*. Optimal decisions depending on stage and state are determined backwards step by step as those maximizing the right hand side of the functional equation. This way of determining an optimal *strategy* is based on the Bellman principle of optimality which says: "An optimal strategy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal strategy with regard to the state resulting from the first decision".

Markov Decision Process:

1. *Definition:* MDP is defined as a 4-tuple *(S, A, TP, R)*:
   - *S* is a *finite set of states*,
   - *A* is a *finite set of actions* which permits the transition between states. There is generally a discrete number of actions.
   - *TP: S×A×S → <0,1>* is a *transition probability* which encodes the probabilistic effects of actions; *T (s, a, s')* is the probability to go from state *s* to state *s'*, when action *a* is performed.
   - *R: S → R* is the *reward function* used to specify the goal to reach and the dangerous parts. *R(s)* gives the reward or penalty for being in state *s*.
2. *Optimal strategy*: In MDP, we know at each instant the current state. Actions must provide all the information for predicting the next state. Once the set of states *S* has been defined and the goal state chosen, then an *optimal strategy σ: S → A* gives the *optimal action* to execute *in each state* of *S* in order to reach the goal state(s) (according to a given optimality criterion).

The two most important algorithms used to calculate the optimal policy are: *Value Iteration* [3] and *Policy Iteration* [8]. The Value Iteration algorithm proceeds by little improvement at each iteration and requires a lot of iterations. Policy Iteration however, yields greater improvement at each iteration and accordingly needs fewer iterations, but each iteration is very expensive.

3.2. Optimal Proof Tree Planning Method

We specify MDPs approach in our case of searching for an optimal proof in linear sequent calculus.

1. *Definition:* of *(S, A, TP, R)*:
   - $S = \{s_1, s_2, \ldots, s_n\}$ where every state is associated to the set of sequents at each stage.
   - $A = \{\{a_i\}, \{a_{i1}, a_{i2}, \ldots, a_{i1}\}, \ldots, \{a_{i1}, a_{i2}, \ldots, a_{ik}\}\}$ where i is the number of rules required for the proof construction, and each action represents rules which we can apply for sequents at each stage.
   - The *transition probability* remains the same as defined, i.e. *TP: S×A×S → <0,1>* where p_{ij}^a is the probability to go from state s_i to state s_j, when action a is performed.
   - We modify the *reward function R: S×A → R* where r_i^a gives the reward for being in state s_i and performing action *a*.
2. *Optimal strategy:* An *optimal strategy σ: S → A* gives the *optimal action* to execute *in each state* of S in order to reach the goal state(s).

$$\forall\ s_i \in S: \sigma(s_i) \in A$$

3.3. Value Iteration Method

Under finite planning horizon the *value iteration method* is excellent. The optimal strategy is determined sequentially using the *functional equations* [10]:

$$f_i(n) = \max_a \left\{ r_i^a + \sum_{i=1}^{u} p_{ij}^a f_j(n-1) \right\}, \qquad (1)$$

$$i = 1,2,\mathrm{K}\ ,u$$

where the action a maximizes the right hand side, which is optimal for the state s_i at the stage n. The function $f_i(n)$ is the *total expected rewards* from the process when it starts from state s_i and will operate for n stages before termination. Thus $f_i(0)$ is the salvage value of the system when it is in state s_i. At each stage an optimal strategy is chosen using these functional equations.

An optimal strategy is one with maximum expected value.

$$\sigma_i(n) = \arg\max_a \left\{ r_i^a + \sum_{i=1}^{u} p_{ij}^a f_j(n-1) \right\}, \quad i = 1,2,\text{K},u \qquad (2)$$

In this equation in each stage we choose the action which maximizes the function.

4. APPLICATION OF THE VALUE ITERATION METHOD

Suppose we have a sequent $\vdash a \otimes b,\ a^{\perp} \wp b^{\perp}$. We construct the proof trees by Andreoli's proof normalisation. We want to decide which one is optimal. We use the above mentioned value iteration method to compute the optimal strategy.

Possible proof trees look like that on **Fig. 1**:

$$\otimes\ \frac{\dfrac{}{\vdash a, a^{\perp}}\,\mathrm{id}\quad \dfrac{}{\vdash b, b^{\perp}}\,\mathrm{id}}{\wp\ \dfrac{\vdash a \otimes b, a^{\perp}, b^{\perp}}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}}} \qquad \otimes\ \frac{\dfrac{}{\vdash a, b^{\perp}}\,?\quad \dfrac{}{\vdash b, a^{\perp}}\,?}{\wp\ \dfrac{\vdash a \otimes b, a^{\perp}, b^{\perp}}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}}}$$

$$\otimes\ \frac{\dfrac{}{\vdash a, a^{\perp}, b^{\perp}}\,?\quad \dfrac{}{\vdash b}\,?}{\wp\ \dfrac{\vdash a \otimes b, a^{\perp}, b^{\perp}}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}}} \qquad \otimes\ \frac{\dfrac{}{\vdash a}\,?\quad \dfrac{}{\vdash b, a^{\perp}, b^{\perp}}\,?}{\wp\ \dfrac{\vdash a \otimes b, a^{\perp}, b^{\perp}}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}}}$$

Fig. 1 Proof trees obtained by proof normalisation.

There are 2^n ways of partitioning the context $a^{\perp}$, $b^{\perp}$ in the case of the sequent $\vdash a \otimes b,\ a^{\perp},\ b^{\perp}$. Hence we got four proof trees. We have to decide, which of these are optimal in the sense that every leaf is an axiom.

Therefore we apply MDP to these proof trees to compute the optimal strategy. The set of states, actions, the reward function and transition probability function are described below.

$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$
$A = \{a_1, a_2, a_3\}$
where $a_1 = \{\otimes\}$, $a_2 = \{\wp\}$, $a_3 = \{id, id\}$

The *reward function* defines for all states the most suitable action to perform and the *probabilities* of the transition from one state to another are determined by the chance of realizing the defined transition under a given action.

State	Set of sequents
s_1	$\{\vdash a \otimes b, a^{\perp} \wp b^{\perp}\}$
s_2	$\{\vdash a \otimes b, a^{\perp}, b^{\perp}\}$
s_3	$\{\vdash a, a^{\perp}; \vdash b, b^{\perp}\}$
s_4	$\{\vdash a, b^{\perp}; \vdash b, a^{\perp}\}$
s_5	$\{\vdash a, a^{\perp}, b^{\perp} \vdash b\}$
s_6	$\{\vdash a \vdash b, a^{\perp}, b^{\perp}\}$
s_7	$\{\emptyset\}$

$r_{s_i}^{a_l}$			
A \ S	a_1	a_2	a_3
s_1	0	5	0
s_2	5	0	0
s_3	0	0	5
s_4	0	0	0
s_5	0	0	0
s_6	0	0	0
s_7	0	0	0

Fig. 2 The set of states and the reward function.

p_{ij}	a_1							a_2							a_3						
s_i \ s_j	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
s_2	0	0	0.25	0.25	0.25	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
s_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 3 The probability function.

In this example we are proceeding by the following algorithm:

Procedure: *value iteration (TP, R)*
Inputs: TP is transition probability specifying p_{ij}^a
R is a reward function r_i^a
Outputs: $\sigma[s]$ is an optimal strategy
$f[s]$ is a value function

for $\forall s$ **do**
$f_0[s] = 0\ ;\ \sigma[s] = 0$
end for
for $k = 1, k \le n$ **do**
for $\forall s$ **do**

$$f_k[s] = \max_a \left\{ r_s^a + \sum_{i=1}^{u} p_{ss_j}^a f_{k-1}[s_j] \right\}$$

end for
end for
for $\forall s$ **do**

$$\sigma[s] = \arg\max_a \left\{ r_s^a + \sum_{i=1}^{u} p_{ss_j}^a f_{k-1}[s_j] \right\}$$

end for
return f_k, σ

For the initial state s_1 the functional equation is:

$$f_{s_1}(3) = \max_{a_i} \left\{ r_{s_1}^{a_i} + \sum_{j=1}^{5} p_{1j}^{a_i} f_{s_j}(2) \right\}; \forall i = 1,2,3$$

$a_1 : f_{s_1}(3) = 0 + 0$
$a_2 : f_{s_1}(3) = 5 + 1 * f_{s_2}(2) = 15$
$a_3 : f_{s_1}(3) = 0 + 0$

It is a recursive function which values are calculated by the functional equations (1) and these values are presented in the table below.
The optimal strategy for the initial state s_1 is computed by the equation (2) as follows:

$$\sigma(s_1) = \arg\max_{a_i}\left\{ r_{s_1}^{a_i} + \sum_{i=1}^{5} p_{1j}^{a_i} f_{s_j}(2) \right\}; \forall i = 1,2,3$$

$$\sigma(s_1) = 10$$

The total expected rewards and the optimal strategy are presented in the following tables

Total Rewards	Value
f_{s_1}	11.25
f_{s_2}	6.25
f_{s_3}	5
f_{s_4}	0
f_{s_5}	0
f_{s_6}	0
f_{s_7}	0

Opt. Strategy	ArgMax
$\sigma(s_1)$	a_2
$\sigma(s_2)$	a_1
$\sigma(s_3)$	a_3
$\sigma(s_4)$	$\emptyset$
$\sigma(s_5)$	$\emptyset$
$\sigma(s_6)$	$\emptyset$
$\sigma(s_7)$	$\emptyset$

Fig. 3 Total rewards and the optimal strategy.

In this way we got the *optimal strategy* for the proof construction, i.e. for all states we get the set of actions which maximize the total expected rewards.

5. CONCLUSION

In our contribution we proposed a stochastic programming method (value iteration method) in linear logic proof construction. This new approach is used mainly in the case, when a "don‘t know" non-determinism occurs. In the case of such a non-determinism the branch of a proof tree is chosen only with a certain probability. We used Markov decision process because it models decision making in situations where the results are partly random and partly influenced by the decision maker. It is similar to linear logic proof construction, where in some cases we can force the way of building the proof but in non-deterministic case like "⊕" we have no permission to select one of the two possible ways to continue. Also in the case of the synchronous connective "⊗" many possible instances of the appropriate inference rule can be applied, corresponding to different partitions of the context along two branches.

We presented by a concrete example for the above mentioned connective ⊗ finding the most probable paths in searching for the proof of a given linear sequent. In one part of the proof search we applied Andreoli’s proof normalisation. It was in the case of the selection of a principal formula. In such a way we have build the possible proof trees. Then we investigated asynchronous formulas. We assigned probabilities to actions representing inference rules of linear sequent calculus and we have computed the most probable strategy for building correct proofs by maximizing the expected reward of functional equations using the algorithm of the Value Iteration Method.

REFERENCES

[1] Alexiev V. : Applications of Linear Logic to Computation. Bulletin of the IGPL, vol. 2, No. 1, 1994, pp. 77-107.

[2] Andreoli J. M. : Logic programming with focusing proofs in linear logic. Journal of Logic and Computation, vol. 2, No. 3, 1992, pp. 297-347.

[3] Bellman R., Holland J.,Kalaba R. :: On an Application of Dynamic Programming to the Synthesis of Logical Systems. ACM Press 4, 1959, vol. 6.

[4] Galmiche, D., Perrier, G. : A procedure for automatic proof nets construction. Springer-Verlag LNAI 624, 135, 1992, pp.42-53.

[5] Galmiche, D., Perrier, G. : Foundations of Proof Search Strategies Design in Linear Logic. Logical Foundations of Computer Science ({LFCS}'94), No. 813, 135, 1994, pp.101-113.

[6] Girard. J.-Y. : On the meaning of logical rules I : syntax vs. semantics. Note CRAS Paris, January 1998, pp.1-45.

[7] Girard, J.-Y., Lafont Y., Taylor, P. : Proofs and Types, Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press, 1988

[8] Howard. R. A. : Dynamic Programming and Markov Processes. Cambridge, Massachussets, U.S.A.: MIT Press, 1960.

[9] Kall P., Wallace S. W. : Stochastic programming. John Wiley & Sons, New York, second edition, 2003.

[10] Kristensen, A. : Dynamic programming and Markov decision processes. Dina Notat No. 49, 1996.

[11] Lincoln P.,Shankar N. : Proof search in first-order linear logic and other cut-free sequent calculi. Proceedings of the Ninth Annual IEEE Symp. on Logic in Computer Science, 1994, pp.282-291.

[12] Tejfel M., Horváth Z., Kozsik T. : Temporal properties of functional programs proven in Sparkle-T. Z. Horváth (Ed.) Central European Functional Programming School (The First Central European Summer School, CEFP 2005, Budapest, Hungary, July 4--15, 2005), Revised Selected Lectures. LNCS 4161. Springer-Verlag, 2006, pp.168-190.

[13] Verbová A., Novitzká V., Slodičák V. : From Linear Sequent Calculus to Proof Nets. Informatics'07, Bratislava, 2007, pp.100-107.

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

BIOGRAPHIES

Valerie Novitzká defended her PhD Thesis: On semantics of specification languages at Hungarian Academy of Sciences in 1989. She works at Department of Computers and Informatics from 1998, firstly as Assistent Professor, from 2004 as Associated Professor. Her research areas covers category theory, categorical logic, type theory, classical and linear logic and theoretical foundations of program development.

Anita Verbová was born on 15.09.1982. In 2006 she graduated (Ing.) at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. She is working on her PhD. degree at the DCI FEEI, Technical university of Košice, Slovakia. Her scientific research is focusing on interaction categories, which are one form of representation of parallel processes. In addition, she also investigates questions related with proof search in linear sequent calculus.