



**HAL**  
open science

## Parallelism in Soft Linear Logic

Paulin Jacobé de Naurois

► **To cite this version:**

Paulin Jacobé de Naurois. Parallelism in Soft Linear Logic. Computer Science Logic, Feb 2022, Göttingen, Germany. 10.4230/LIPIcs..10 . hal-03374648

**HAL Id: hal-03374648**

**<https://hal.science/hal-03374648>**

Submitted on 12 Oct 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# 1 Parallelism in Soft Linear Logic

2 Paulin Jacobé de Naurois ✉

3 CNRS, Université Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, F-93430 Villetaneuse, France.

## 4 — Abstract —

---

5 We extend the Soft Linear Logic of Lafont with a new kind of modality, called *parallel*. Contractions  
6 on parallel modalities are only allowed in the cut and the left  $\multimap$  rules, in a controlled, uniformly  
7 distributive way. We show that SLL, extended with this parallel modality, is sound and complete  
8 for PSPACE. We propose a corresponding typing discipline for the  $\lambda$ -calculus, extending the STA  
9 typing system of Gaboardi and Ronchi, and establish its PSPACE soundness and completeness. The  
10 use of the parallel modality in the cut-rule drives a polynomial-time, parallel call-by-value evaluation  
11 strategy of the terms.

12 **2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Linear logic; Theory of computation  
13  $\rightarrow$  Complexity theory and logic

14 **Keywords and phrases** Implicit Complexity, Typing, Linear Logic, Functional Programming

15 **Digital Object Identifier** 10.4230/LIPIcs...10

16 **Funding** *Paulin Jacobé de Naurois*: ANR project ELICA - ANR-14-CE25-0005

## 17 Introduction

18 Implicit Complexity aims at providing purely syntactical, machine independent criteria on  
19 programs, in order to ensure they respect some complexity bounds upon execution. In  
20 the context of functional programming, the use of tailored proof systems, and subsequent  
21 type systems for  $\lambda$ -calculus, has been very successful: using subsystems of Linear Logic [8],  
22 several proof systems have been proposed, where cut-elimination has a bounded complexity.  
23 Consequently, under the Curry-Howard isomorphism, type systems for  $\lambda$ -calculus based on  
24 these logics have been proposed, where  $\beta$ -normalization of the typed terms follows the same  
25 complexity bounds. Such results include, among many others, Bounded Linear Logic [10, 14]  
26 and Light Linear Logic [9, 2] for polynomial time computation, and Stratified Bounded Affine  
27 Logic [17, 15] for logarithmic space computations. Our interest in this paper lies in the  
28 Soft Linear Logic of Lafont [13], which proposes a simple and elegant approach for ensuring  
29 polynomial time bounds by controlling contractions on exponential formulas, and in the  
30 subsequent type systems for polynomial time  $\lambda$ -calculus [1, 7, 5].

31 At this point, it is relevant to note that the complexity classes captured thus far are all  
32 sequential, deterministic *in essence*. While Soft Linear Logic type systems have been extended  
33 to express the classes NP and PSPACE [6], it is important to note that the construction relies  
34 on Soft Type Assignment (STA), a deterministic, sequential polynomial time type system, by  
35 extending the  $\lambda$ -calculus with an additional construct (**if then else**), for which an ad hoc,  
36 alternating polynomial time evaluation strategy is imposed - the core of the language retaining  
37 its sequential polynomial time evaluation. While being indeed extensionally complete for  
38 PSPACE, this approach lacks intensionality: many natural algorithms, that are easily  
39 computable in parallel, are hardly expressible in this setting. Let us take as simple example  
40 the numerical evaluation of a balanced, arithmetic expression on bounded integer values. In  
41 order to compute it in alternating polynomial time with the (**if then else**) defined in [6],  
42 one would need to express the value of all bits of the result as boolean expressions on the bits  
43 of the input numbers, and use the alternating evaluation of the (**if then else**) construct  
44 to speed up the parallel computation time - not quite a practical method. Furthermore,  
45 this approach is no longer doable in real world functional programming languages, where



© Paulin Jacobé de Naurois;

licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 10:2 Parallelism in Soft Linear Logic

46 integers are given as a base type, and arithmetical operations as unitary functions of the  
47 language. Our approach, on the other hand, extends very naturally to such programs: indeed,  
48 our complexity bounds still hold in this context, and the encodings used in Lemma 27 and  
49 Theorem 28 can seamlessly be used to encode uniform families of algebraic formulae, or  
50 algebraic circuits, of polynomial depth, provided the base type for numbers (be they integers  
51 or floating numbers, or even real or complex numbers) and for the algebraic basic operations  
52 are given in the typing context.

53 A reason why these approaches are all essentially sequential, deterministic is that they  
54 use the typing discipline to control the *amount* of resources the calculus uses (e.g. by  
55 controlling contractions on exponentials), not the *way* these resources are distributed along  
56 the computation. In order to truly denote parallel computation in a functional programming  
57 language, our proposal here is to use a parallel, call-by-value evaluation strategy for the  
58  $\lambda$ -calculus: in an application, both terms can be normalized in parallel, before the substitution  
59 of the redex takes place. If both terms share the same normalization time bound, the parallel  
60 evaluation strategy is efficient. Note that in first order functional programming, this is already  
61 the approach used by Leivant and Marion [16] with their safe recursion with substitutions:  
62 using sequential resource bounds from Ptime Safe Recursion [3], and a parallel call-by-  
63 value evaluation strategy, the authors characterize the class FPAR (Parallel polynomial  
64 time), which coincides with PSPACE. This approach has also been later on extended to  
65 sub-polynomial complexity classes [12, 4, 11]. For higher order functional programming, we  
66 rely on the Curry-Howard isomorphism: ensuring an homogeneous computation time on  
67 the parallel evaluation of both arguments of an application amounts to ensuring that both  
68 premises of a cut-rule share a homogeneous bound on the resource usage in the corresponding  
69 type system.

70 In order to achieve this, we can no longer rely on the usual linear cut-rule. We propose  
71 therefore a modification of the linear cut-rule, that internalizes a controlled number of  
72 contractions on some formulas, that are uniformly distributed among the premises. These  
73 formulas are decorated with a dedicated modality, called parallel modality. This approach is  
74 applied here on the Soft Type Assignment (STA) of Gaboardi and Ronchi [7], in order to  
75 propose a sound and complete type system for PSPACE, with a truly parallel evaluation  
76 strategy.

77 Of course, breaking linearity in the cut-rule comes with a price: while proof nets for this  
78 logic are still definable, the additional bureaucracy needed to deal with the side condition of  
79 the cut-rule makes them much less meaningful than those for simpler logical systems such as  
80 *MLL* or *SLL*.

81 The paper is organized as follows. Section 1 recalls the Soft Linear Logic rules, introduces  
82 the parallel modality  $//$ , and the modified, parallel (cut) rules, yielding the system PSLL.  
83 Cut-elimination for PSLL is also shown. Section 2 provides a parallel, polynomial time  
84 normalization bound. Section 3 extends STA with the rules of PSLL, yielding PSTA. A  
85 parallel polynomial time call-by-value strategy for PSTA is described. FPAR completeness  
86 of PSTA is proven in Section 4.

## 87 **1** Parallel Soft Linear Logic

### 88 **1.1** Soft Linear Logic

89 Let us recall the SLL rules of Lafont [13], in its intuitionistic fashion. In the following,  $! \Gamma$   
90 stands for a multiset of formulae of the form  $!F$ , and  $(A)^n$  stands for  $n$  copies of a formula  $A$ .

$$\begin{array}{l}
91 \quad \frac{}{U \vdash U} (Id) \qquad \frac{\Gamma \vdash U \quad \Delta, U \vdash V}{\Gamma, \Delta \vdash V} (cut) \qquad \frac{\Gamma, U \vdash V}{\Gamma \vdash U \multimap V} (\multimap R) \\
92 \quad \frac{\Gamma \vdash U \quad V, \Delta \vdash Z}{\Gamma, U \multimap V, \Delta \vdash Z} (\multimap L) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&R) \quad \frac{\Gamma, A \vdash V}{\Gamma, A \& B \vdash V} (\&L_1) \\
93 \quad \frac{\Gamma, B \vdash V}{\Gamma, A \& B \vdash V} (\&L_2) \quad \frac{\Gamma \vdash U}{\Gamma \vdash \forall \alpha U} (\forall R) \quad \frac{\Gamma, U[V/\alpha] \vdash Z}{\Gamma, \forall \alpha U \vdash Z} (\forall L) \\
94 \quad \frac{\Gamma \vdash U}{!\Gamma \vdash !U} (sp) \qquad \frac{\Gamma, (U)^n \vdash V \quad n \geq 0}{\Gamma, !U \vdash V} (m), \text{ of rank } n
\end{array}$$

95 where, in the  $(\forall R)$ -rule, there is no free occurrence of  $\alpha$  in  $\Gamma$ . SLL proofs (of a given  
96 degree) normalize in polynomial time. Let the rank of a proof be the maximal rank of its  
97  $(m)$  rules, and its degree the maximal nesting of its  $(sp)$  rules:

98 ► **Theorem 1** ([13]). *A SLL proof of rank  $n$  and degree  $d$  normalizes in  $n^d$  steps.*

99 SLL is also complete for the class FP: inputs of size  $n$  are encoded with proofs of rank  $n$ ,  
100 degree 1, and programs running in time  $O(n^d)$  by proofs of degree  $d$ . Applying a program  
101 on an input amounts to performing a  $(cut)$  of the two proof derivations.

## 102 1.2 Parallel Modalities

103 PSLL is built upon SLL. An additional modality, called the parallel modality  $//$ , is introduced,  
104 with corresponding elimination rules. Finally, the  $(sp)$ , and the  $(cut)$  and  $(\multimap L)$ -rules are  
105 modified to accommodate this new modality, implementing the controlled contractions and  
106 homogeneous distribution of  $//$  formulas on the premises of the cut, as follows.

### 107 Polarities

108 Let us define as usual inductively the polarity of a sub-formula in an intuitionistic sequent  
109  $\Gamma \vdash V$ . Polarities are either positive or negative, one being the opposite of the other.

- 110 1. in  $\Gamma \vdash V$ , every occurrence of a formula  $F$  in  $\Gamma$  is negative, and  $V$  is positive.
- 111 2. If  $F$  is  $\forall \alpha A$ ,  $!A$  or  $//A$ , the polarity of  $A$  is the polarity of  $F$ .
- 112 3. If  $F$  is  $A \& B$ , the polarity of  $A$  and the polarity of  $B$  are the polarity of  $F$ .
- 113 4. If  $F$  is  $A \multimap B$ , the polarity of  $A$  is the opposite of the polarity of  $F$ , and the polarity of  
114  $B$  is the polarity of  $F$ .

115 In the sequel we only admit  $//A$  sub-formulas with negative polarities in a sequent. An  
116 immediate consequence is that no  $//$  modality can appear in a cut formula, since a cut-formula  
117 has both a positive and a negative occurrence in a proof tree.

### 118 Rules for Parallel Modalities

119  $(//W)$  (weakening) and  $(//D)$  (dereliction) rules eliminate the  $//$  modality,  $(//sp)$  (soft pro-  
120 motion for the  $!$  modality) and  $(//ax)$  replace the linear  $(sp)$  and  $(Id)$  rules. Contraction  
121 for the  $//$  modality is not dealt with a dedicated rule, but is instead internalized in the side  
122 condition of the modified  $(cut)$  rule, as detailed in the next section.

$$123 \quad \frac{\Gamma \vdash B}{\Gamma, //A \vdash B} (//W) \quad \frac{\Gamma, A \vdash B}{\Gamma, //A \vdash B} (//D) \quad \frac{//\Delta, \Gamma, \vdash U}{//\Delta, !\Gamma \vdash !U} (//sp) \quad \frac{}{//\Gamma, A \vdash A} (//ax)$$

124 where  $(//ax)$ , is derivable from  $(Id)$  and  $(//W)$ , and used for convenience only.

125 **1.2.1 ( $\parallel$ Cut) and ( $\parallel \multimap$ ) Rules**

126 Contraction for parallel formulas is internalized into the PSLL ( $\parallel$ cut)-rule and ( $\parallel \multimap L$ )-rule,  
 127 in a controlled fashion. As for the usual ( $cut$ ) and ( $\multimap L$ )-rules in linear logic, linear and  
 128 exponential formulas are linearly distributed among the two premises. Denote by  $\subsetneq$  the strict  
 129 inclusion relation on multisets. The (binary) ( $\parallel$ cut) and ( $\parallel \multimap L$ ) rules are the following.

$$130 \frac{\parallel\Delta_1, \Gamma_1 \vdash A_1 \quad \parallel\Delta_2, \Gamma_2, A_1 \vdash A_2}{\parallel\Delta, \Gamma_1, \Gamma_2 \vdash A_2} (\parallel cut) \quad \frac{\parallel\Delta_1, \Gamma_1 \vdash A_1 \quad \parallel\Delta_2, \Gamma_2, A_2 \vdash A_3}{\parallel\Delta, \Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash A_3} (\parallel \multimap L)$$

131 These two rules hold under the side condition  $\mathcal{S}_P : (\parallel\Delta_1 \subsetneq \parallel\Delta, \parallel\Delta_2 \subsetneq \parallel\Delta)$ . The  
 132 ( $\parallel$ cut)-rule has the principal cut-formula  $A_1$  and the cut-pair of premises the pair  $(\parallel\Delta_1, \Gamma_1 \vdash$   
 133  $A_1) \rightarrow (\parallel\Delta_2, \Gamma_2, A_1 \vdash A_2)$ . The ( $\parallel \multimap L$ ) rule has the principal  $\multimap$ -formula  $A_1 \multimap A_2$  and  
 134 the  $\multimap$ -pair of premises the pair  $(\parallel\Delta_1, \Gamma_1 \vdash A_1) \rightarrow (\parallel\Delta_2, \Gamma_2, A_2 \vdash A_3)$ .

135 In a proof tree consisting only in  $(n - 1)$  binary linear ( $cut$ )-rules, these ( $cut$ )-rules can  
 136 be freely permuted, and a generalized,  $n$ -ary linear ( $cut$ )-rule can be derived. The non-linear  
 137 distribution of parallel modalities in PSLL breaks this isomorphism: permuting two binary  
 138 ( $\parallel$ cut)-rules may come in conflict with the side condition  $\parallel\Delta_i \subsetneq \parallel\Delta$ . A similar remark can  
 139 be made for  $\multimap L$  rules as well. Since we want a *uniform* bound on the parallel normalization  
 140 of the premises, we define a  $n$ -ary parallel ( $cut$ )-rule, exemplified in Example 5, as a parallel  
 141 extension of the linear one, where the side condition for  $\parallel$  modalities is adapted accordingly.

142 **► Definition 2 ( $n$ -ary ( $cut/\multimap L$ ) rule).** We define the following  $n$ -ary ( $cut/\multimap L$ )-rule,  
 143 together with its cut-pairs and  $\multimap$ -pairs, and principal formulae. To each cut-pair (respectively  
 144  $\multimap$ -pair) corresponds one principal cut-formula (resp.  $\multimap$ -formula).

145 The following rule  $\frac{\Gamma_1 \vdash A_1 \quad \Gamma_2 \vdash A_2 \cdots \Gamma_d \vdash A_d}{\Gamma, \Lambda \vdash A_d} R : (cut/\multimap L)$  is either a bin-  
 146 ary ( $\multimap L$ ) or a binary ( $cut$ )-rule, or a  $n$ -ary rule obtained by several of the following proof  
 147 tree ( $cut/\multimap L$ )-merge rewriting steps:

$$148 \frac{T_1 \cdots \frac{T_2 \cdots T_n}{T_t} R_1 \cdots T_m}{\Gamma, \Lambda \vdash A_d} R_2 \quad \rightarrow \quad \frac{T_1 \cdots T_2 \cdots T_n \cdots T_m}{\Gamma, \Lambda \vdash A_d} R$$

149 provided the  $\multimap$  principal formulae of  $R_1$  are not sub-formulae of any principal formulae  
 150 of  $R_2$  corresponding  $T_t$ .

151 The multiset of  $\multimap$  (respectively ( $cut$ )) principal formulae of  $R$  is then the union of those  
 152 of  $R_1$  and  $R_2$ .

153 The cut - and  $\multimap$ -pairs of  $R$  are obtained from the union of those of  $R_1$  and  $R_2$  with the  
 154 following update procedure: whenever  $T_t$  belongs to a  $\multimap$  or cut-pair of premises  $T_t \rightarrow T_w$   
 155 (respectively  $T_w \rightarrow T_t$ ) of  $R_2$ , with corresponding principal formula  $F$  belonging to one of  
 156 the premises  $T_v$  of  $R_1$ , the pair  $T_t \rightarrow T_w$  (resp.  $T_w \rightarrow T_t$ ) is replaced by  $T_v \rightarrow T_w$  (resp.  
 157  $T_w \rightarrow T_v$ ), with the same corresponding principal formula.

158 We derive from this linear  $n$ -ary ( $cut/\multimap L$ ) rule its parallel version ( $\parallel, \multimap cut$ ) as follows.

159 **► Definition 3 ( $n$ -ary ( $\parallel, \multimap cut$ ) rule).** A  $n$ -ary ( $\parallel, \multimap cut$ ) rule is

$$160 \frac{\parallel\Delta_1, \Gamma_1 \vdash A_1 \quad \parallel\Delta_2, \Gamma_2 \vdash A_2 \cdots \parallel\Delta_d, \Gamma_d \vdash A_d}{\parallel\Delta, \Gamma, \Lambda \vdash A_d} (\parallel, \multimap cut)$$

161 where the side condition  $\mathcal{S}_P : \forall i = 1, \dots, d, \parallel\Delta_i \subsetneq \parallel\Delta$  holds, and the linear rule instance

$$162 \frac{\Gamma_1 \vdash A_1 \quad \Gamma_2 \vdash A_2 \cdots \Gamma_d \vdash A_d}{\Lambda, \Gamma \vdash A_d} (cut/\multimap L)$$

163 holds as per Definition 2, with corresponding pairs of premises and principal formulae.

164 The following Lemma follows from the intuitionistic nature of the PSSL sequents, and  
165 will play a role in our elimination strategy.

166 ► **Lemma 4.** *The cut-pairing relation on the premises of a ( $\parallel, \text{cut}/ \multimap L$ ) rule  $R$  defines a*  
167 *forest structure  $F(R)$ , called the pairing forest of the ( $\parallel, \text{cut}/ \multimap L$ ), on the premises of  $R$ ;*  
168 *the edges of the pairing forest are the cut-pairs of the rule.*

169 ► **Example 5.** A tree of linear ( $\multimap L$ ) and ( $\text{cut}$ ) rules is

$$170 \frac{\Gamma_1 \vdash A_1 \quad \frac{\frac{\Gamma_2 \vdash A_2 \quad \Gamma_3, A_2 \vdash U}{\Gamma_2, \Gamma_3 \vdash U} (\text{cut}) \quad \frac{\Gamma_4 \vdash A_3 \quad \Gamma_5, A_1, A_3, V \vdash W}{\Gamma_4, \Gamma_5, A_1, V \vdash W} (\text{cut})}{\Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V, A_1 \vdash W} (\multimap L)}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W} (\text{cut})$$

171 The corresponding 5-ary linear ( $\text{cut}/ \multimap L$ ) rule is

$$172 \frac{\Gamma_1 \vdash A_1 \quad \Gamma_2 \vdash A_2 \quad \Gamma_3, A_2 \vdash U \quad \Gamma_4 \vdash A_3 \quad \Gamma_5, A_1, A_3, V \vdash W}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W} (\text{cut}, \multimap L)$$

173 A corresponding 5-ary ( $\parallel, \multimap \text{cut}$ ) rule, with  $\parallel$ -formulae satisfying the side condition, is

$$174 \frac{\parallel F, \Gamma_1 \vdash A_1 \quad \parallel G, \Gamma_2 \vdash A_2 \quad \Gamma_3, A_2 \vdash U \quad \parallel G, \Gamma_4 \vdash A_3 \quad \parallel F, \Gamma_5, A_1, A_3, V \vdash W}{\parallel F, \parallel G, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}$$

175 The cut-pairs are

- 176 ■  $(\parallel F, \Gamma_1 \vdash A_1) \rightarrow (\parallel F, \Gamma_5, A_1, A_3, V \vdash W)$  with principal formula  $A_1$ ,
- 177 ■  $(\parallel G, \Gamma_2 \vdash A_2) \rightarrow (\Gamma_3, A_2 \vdash U)$  with principal formula  $A_2$ , and
- 178 ■  $(\parallel G, \Gamma_4 \vdash A_3) \rightarrow (\parallel F, \Gamma_5, A_1, A_3, V \vdash W)$  with principal formula  $A_3$ ,

179 which defines the pairing forest, with two roots  $(\parallel F, \Gamma_5, A_1, A_3, V \vdash W)$  and  $(\Gamma_3, A_2 \vdash U)$ .

180 The  $\multimap$ -pair is  $(\Gamma_3, A_2 \vdash U) \rightarrow (\parallel F, \Gamma_5, A_1, A_3, V \vdash W)$  with principal formula  $U \multimap V$ .

181 We now define PSSL by the rules ( $\parallel ax$ ), ( $\multimap R$ ), ( $\forall R$ ), ( $\forall L$ ), ( $\&R$ ), ( $\&L_i$ ), ( $m$ ) ( $\parallel sp$ ),  
182 ( $\parallel W$ ), ( $\parallel D$ ) and ( $\parallel, \multimap \text{cut}$ ).

183 A PSSL proof  $\Pi$  is said to be in *normal form* if it contains no cut: more precisely,  
184 no ( $\parallel, \multimap \text{cut}$ )-rule in  $\Pi$  admits any cut-pair of premises. Cut-elimination in this context  
185 amounts to rewrite the proof into a new equivalent proof in normal form. The cut-elimination  
186 procedure stems on the usual one for SLL, with some refinements.

### 190 1.3 Parallel Cut Elimination

191 ► **Lemma 6.** *Sequent calculus rules preserve the polarities of subformulae.*

192 The proof is straightforward. This allows us to state the following rule commutation result.

193 ► **Lemma 7.**

- 194 1. A ( $\parallel, \multimap \text{cut}$ ) rule ( $R_1$ ), with premise  $\Gamma \vdash V$  commutes with any non ( $\parallel, \multimap \text{cut}$ ), non  
195 ( $\parallel W$ ), non ( $\parallel D$ ), non ( $\parallel sp$ ) rule ( $R_2$ ) with conclusion  $\Gamma \vdash V$ , provided the principal  
196 formula of ( $R_2$ ) is not a sub-formula of any principal formula of ( $R_1$ ) with respect to the  
197 premise  $\Gamma \vdash V$ .
- 198 2. A ( $\parallel W$ ) or a ( $\parallel D$ ) rule ( $R_1$ ), with premise  $\Gamma \vdash V$  commutes with any non ( $\parallel ax$ ) rule  
199 ( $R_2$ ) with conclusion  $\Gamma \vdash V$ , provided the principal formula of ( $R_2$ ) is not a subformula  
200 of the principal formula of ( $R_1$ ).

## 10:6 Parallelism in Soft Linear Logic

201 3. A non ( $\//, \multimap$  cut), non ( $\//sp$ ) rule ( $R_1$ ) with premise  $\Gamma \vdash V$  commutes with any non  
 202 ( $\//, \multimap$  cut), non ( $\//sp$ ) rule ( $R_2$ ) with conclusion  $\Gamma \vdash V$ , provided the principal formula of  
 203 ( $R_2$ ) is not a sub-formula of the principal formula of ( $R_1$ ).

204 ► **Proposition 8.** *PSLL enjoys cut elimination.*

205 **PROOF.** Let  $\Pi$  be a PSLL proof and  $R$  be a ( $\//, \multimap$  cut) rule in  $\Pi$  with cut-pair  
 206 ( $S = \Gamma \vdash A, T = \Lambda, A \vdash V$ ). Since the cut formula  $A$  may not contain any  $\//$  modality, the  
 207 commutation rules of Lemma 7 allow us to rewrite  $\Pi$  into an equivalent proof  $\Pi'$ , where  $S$   
 208 is conclusion of a right rule with principal formula  $A$ , and  $T$  conclusion of a left rule with  
 209 principal formula  $A$ . The cut-elimination cases are then the following, where, for all cases but  
 210 ( $\//Sp$ ), ( $m$ ), the other premises of the rule are left unchanged, and omitted. Side conditions  
 211 as well are omitted, but it is straightforward to see that they are preserved. The modification  
 212 induced by each of the elimination cases below on the pairing forest is also detailed.

213 **Rules** ( $\multimap L$ ), ( $\multimap R$ )

$$214 \frac{\frac{\//\Delta_1, \Gamma, B \vdash C}{\//\Delta_1, \Gamma \vdash B \multimap C} (\multimap R) \quad \frac{\//\Delta_3, \Phi \vdash B \quad \//\Delta_4, \Lambda, C \vdash V}{\//\Delta_2, \Phi, \Lambda, B \multimap C \vdash V} (\// \multimap L)}{\//\Delta, \Gamma, \Phi, \Lambda \vdash V} (\//, \multimap \text{ cut})$$

$$215 \text{ reduces to } \frac{\//\Delta_1, \Gamma, B \vdash C \quad \//\Delta_3, \Phi \vdash B \quad \//\Delta_4, \Lambda, C \vdash V}{\//\Delta, \Gamma, \Phi, \Lambda \vdash V} (\//, \multimap \text{ cut}) .$$

216 In the pairing forest, the premise  $\//\Delta_1, \Gamma \vdash B \multimap C$  is replaced by  $\//\Delta_1, \Gamma, B \vdash C$ ,  
 217 the premise  $\//\Delta_2, \Phi, \Lambda, B \multimap C \vdash V$  by  $\//\Delta_4, \Lambda, C \vdash V$ , and a cut-pair ( $\//\Delta_3, \Phi \vdash B$ )  $\rightarrow$   
 218 ( $\//\Delta_1, \Gamma, B \vdash C$ ) is added.

219 **Rule** ( $\//ax$ )

$$220 \frac{\frac{\//\Delta_1, B \vdash B}{\//\Delta, \Gamma, B \vdash V} (\//ax)}{\//\Delta, \Gamma, B \vdash V} (\//, \multimap \text{ cut})$$

221 when no other premise exists, reduces to  $\frac{\//\Delta_2, \Gamma, B \vdash V}{\//\Delta, \Gamma, B \vdash V} (\//W^*)$ , , Where ( $\//W$ )<sup>\*</sup> stands  
 222 for several applications of the ( $\//W$ ) rule.

223 Similarly,

$$224 \frac{\Pi_1 \cdots \frac{\//\Delta_1, B \vdash B}{\//\Delta, \Gamma, B \vdash V} (\//ax) \quad \Pi_t \cdots \//\Delta_2, \Gamma, B \vdash V \quad \cdots \Pi_n}{\//\Delta, \Gamma, B \vdash V} (\//, \multimap \text{ cut})$$

$$225 \text{ reduces to } \frac{\Pi_1 \cdots \Pi_t \cdots \//\Delta_2, \Gamma, B \vdash V \quad \cdots \Pi_n}{\//\Delta, \Gamma, B \vdash V} (\//, \multimap \text{ cut}) .$$

226 In the pairing forest, the premise  $\//\Delta_1, B \vdash B$  is removed, and the paths in the forest are  
 227 shortened accordingly, if necessary.

228 **Rules** ( $\//sp$ ), ( $m$ )

$$229 \frac{S_1, \dots, S_k \quad \frac{\//\Delta_1, \Gamma \vdash B}{\//\Delta_1, !\Gamma \vdash !B} (\//sp) \quad \frac{\//\Delta_2, \Lambda, B^n \vdash V}{\//\Delta_2, \Lambda, !B \vdash V} (m)}{\//\Delta, !\Gamma, \Lambda \vdash V} (\//, \multimap \text{ cut})$$

$$230 \quad \text{reduces to } \frac{S_1^n, \dots, S_k^n \quad // \Delta_1, \Gamma \vdash B \dots \quad // \Delta_1, \Gamma \vdash B \quad // \Delta_2, \Lambda, B^n \vdash V}{\frac{// \Delta, \Gamma^n, \Lambda, \vdash V}{// \Delta, !\Gamma, \Lambda \vdash V} (m)^*} (//, \multimap \text{ cut})$$

231 Where  $S_1, \dots, S_k$  are the premises of the  $(//, \multimap \text{ cut})$  belonging to the pairing tree rooted in  
 232  $// \Delta_1, !\Gamma \vdash !B$ , and  $S_1^n, \dots, S_k^n$  are  $n$  copies of these premises. Then, in the pairing forest, the  
 233 tree rooted in  $// \Delta_1, !\Gamma \vdash !B$  is copied  $n$  times, and the pair  $(// \Delta_1, !\Gamma \vdash !B) \rightarrow (// \Delta_2, \Lambda, !B \vdash V)$   
 234 is replaced by  $n$  pairs  $(// \Delta_1, \Gamma \vdash B) \rightarrow (// \Delta_2, \Lambda, B^n \vdash V)$ , one connected to each of the copies  
 235 above.

236 **Rules**  $(\forall L)$ ,  $(\forall R)$

$$237 \quad \frac{\frac{// \Delta_1, \Gamma \vdash U}{// \Delta_1, \Gamma \vdash \forall \alpha U} (\forall R) \quad \frac{// \Delta_2, \Lambda, U[V/\alpha] \vdash V}{// \Delta_2, \Lambda, \forall \alpha U \vdash V} (\forall L)}{// \Delta, \Gamma, \Lambda \vdash V} (//, \multimap \text{ cut})$$

$$238 \quad \text{reduces to } \frac{// \Delta_1, \Gamma \vdash U[V/\alpha] \quad // \Delta_2, \Lambda, U[V/\alpha] \vdash V}{// \Delta, \Gamma, \Lambda \vdash V} (//, \multimap \text{ cut}).$$

239 **Rules**  $(\&L)$ ,  $(\&R)$

$$240 \quad \frac{\frac{// \Delta_1, \Gamma \vdash A \quad // \Delta_1 \Gamma \vdash B}{// \Delta_1, \Gamma \vdash A \& B} (\&R) \quad \frac{// \Delta_2, \Lambda, A \vdash V}{// \Delta_2, \Lambda, A \& B \vdash V} (\&L_1)}{// \Delta, \Gamma, \Lambda \vdash V} (//, \multimap \text{ cut})$$

241 reduces to  $\frac{// \Delta_1, \Gamma \vdash A \quad // \Delta_2, \Lambda, A \vdash V}{// \Delta, \Gamma, \Lambda \vdash V} (//, \multimap \text{ cut})$ , and, of course, the cut elimina-  
 242 tion rule for  $(\&L_2)$ ,  $(\&R)$  follows a similar pattern.

243 In each of the cases above, for each path in the pairing forest modified by the elimination  
 244 case, the sum of the sizes of the sequents labelling the vertices along that path decreases  
 245 strictly. As a consequence, the procedure terminates in a finite number of steps.

## 246 **2 Complexity Bounds**

247 Let us now show that the contraction discipline ensures that PSLL admits cut-elimination in  
 248 parallel polynomial time. The bounds are actually more straightforward than for SLL.

- 249 ► **Definition 9.** Let  $\Pi$  be a PSLL proof, with conclusion sequent  $S = \Gamma \vdash V$ . We define
- 250 ■ The size  $|S|$  of  $S$  is the number of connectives in  $S$ .
  - 251 ■ The size  $|\Pi|$  of  $\Pi$  is the number of nodes in the proof-tree.
  - 252 ■ The depth of a node  $R$  in  $\Pi$  is the length of the path from  $S$  to  $R$  in  $\Pi$ ; the depth  $d(\Pi)$  of  
 253  $\Pi$  is the maximal depth of its nodes.
  - 254 ■ The rank  $r(\Pi)$  of  $\Pi$  is the maximal rank of its  $(m)$  rules.
  - 255 ■ The degree  $d(f)$  of a formula  $f$  is the maximal nesting of its  $!$  modalities. The degree  
 256  $d(S)$  of a sequent  $S$  is the maximal degree of its formulas. The degree  $d(\Pi)$  of a proof is  
 257 the maximal degree of its sequents.

258 PSLL proofs have bounded depth, and bounded number of  $(\text{cut})$ -rules.

- 259 ► **Lemma 10.** Let  $\Pi$  be a PSLL cut-free proof, of rank  $n$ , with conclusion sequent  $S$  of degree  
 260  $d$ . The depth of  $\Pi$  is then bounded by  $O(|S|.n^d)$ .



## 10:8 Parallelism in Soft Linear Logic

261 ► **Lemma 11.** *Let  $\Pi$  be a PSSL proof, of rank  $n$ , with conclusion sequent  $S$  of degree  $d$ .  
 262 Then, on any path from  $S$  to an axiom in  $\Pi$ , there are at most  $O(|S|.n^d)$  ( $\parallel, \multimap$  cut)-rules  
 263 with cut-pairs of premises.*

264 Combining these two lemmas, we obtain a bound on the depth of PSSL proofs.

265 ► **Lemma 12.** *Let  $\Pi$  be a PSSL proof, of rank  $n$  and degree  $d$ , with conclusion sequent  $S$ . Let  
 266  $M$  be the maximal size of its cut-formulae. Then, the depth of  $\Pi$  is  $O(M.|S|.n^{2d})$ .*

267 ► **Lemma 13.** *Let  $R$  be a ( $\parallel, \multimap$  cut) rule with cut-pairs  $(S_1 = \Gamma_1 \vdash A_1) \rightarrow S, \dots, (S_t =$   
 268  $\Gamma_t \vdash A_t) \rightarrow S$  and cut-formulae  $A_1, \dots, A_t$ . Assume moreover that*

- 269 ■ *each of the proof trees with conclusion  $S_i$ , for  $i = 1, \dots, t$ , ends with the PSSL rule with*  
 270 *right principal formula  $A_i$ , and*
- 271 ■ *the proof tree with conclusion  $S$  ends with the  $t$  PSSL rules with left principal formula*  
 272  *$A_i$ , for  $i = 1, \dots, t$ .*

273 *Then, the cut-elimination steps of Proposition 8 for the cut-pairs  $(S_1, S), \dots, (S_t, S)$  can be*  
 274 *performed in parallel.*

275 **PROOF.** The cut-elimination steps of Proposition 8 act on distinct left sub-formulae of  $S$ ,  
 276 and distinct premises (other than  $S$ ) of the ( $\parallel, \multimap$  cut) rule  $R$ .

277 ► **Definition 14** (Parallel elimination of an innermost cut). *Let  $\Pi$  be a PSSL proof. A ( $\parallel, \multimap$  cut)  
 278 rule  $R$  with cut-pairs is innermost in  $\Pi$  if there is no other ( $\parallel, \multimap$  cut) rule with cut-pairs  
 279 along any path from  $R$  to the axioms of  $\Pi$ .*

280 *Let  $R$  be an innermost ( $\parallel, \multimap$  cut) rule in  $\Pi$ , and  $F(R)$  be the pairing forest. The parallel  
 281 elimination of  $R$  is then the following procedure:*

- 282 1. *For any premise  $S = \Lambda \vdash B$  of  $R$  root in  $F(R)$ , with cut-pairs  $(S_1 = \Gamma_1 \vdash A_1, S), \dots, (S_t =$   
 283  $\Gamma_t \vdash A_t, S)$ , perform the rule permutations of Lemma 7 such that  $S$  is conclusion of a  
 284 proof tree with deep most rules the left rules with principal formulae  $A_1, \dots, A_t$ , and*
- 285 2. *perform the rule permutations of Lemma 7 such that, for  $i = 1, \dots, t$ ,  $S_i$  is conclusion of*  
 286 *a proof tree ending with a right rule with principal formula  $A_i$ .*
- 287 3. *perform in parallel the cut-elimination steps of Lemma 13 for all cut-pairs  $(S_i, S)$  for all*  
 288 *roots  $S$  in  $F(R)$ .*
- 289 4. *if  $R$  has at least one cut-pair left, go to step 1.*

290 ► **Definition 15** (Innermost parallel cut-elimination). *Let  $\Pi$  be a PSSL proof. The Innermost  
 291 parallel cut-elimination procedure consists in applying in parallel, for all its innermost cuts,  
 292 their parallel elimination, until no ( $\parallel, \multimap$  cut) rule with cut-pairs remains.*

293 The innermost parallel cut-elimination procedure ensures that the blow-up of the ( $\parallel, \multimap$   
 294 cut) rules remains under control:

295 ► **Lemma 16.** *Let  $\Pi$  be a PSSL proof with conclusion  $S$ , degree  $d$ , and rank  $n$ . Let  $M$  be  
 296 the maximal size of its cut-formulae and  $w$  the maximal indegree of its pairing forests. Then,  
 297 the maximal indegree of the pairing forests of any proof  $\Pi'$  derived from  $\Pi$  by an innermost  
 298 parallel partial evaluation is bounded by  $O(w.n^d + M)$ .*

299 ► **Lemma 17.** *Let  $\Pi$  be a PSSL proof with conclusion  $S$ , degree  $d$ , and rank  $n$ . Let  $M$  be  
 300 the maximal size of its cut-formulae, and  $h$  the maximal height of its pairing forests. The  
 301 parallel elimination of an innermost cut takes at most  $O(M.h)$  parallel steps.*

302 **PROOF.** For each of the elimination steps of Proposition 8, for each path in the pairing  
 303 forest containing the cut-pair eliminated by this step, the sum of the sizes of the cut-formulae  
 304 along the path strictly decreases, hence the result.



## 10:10 Parallelism in Soft Linear Logic

339 ► **Definition 20** (Parallel Soft types (PSTA)). *In the following,  $\alpha, \beta$ , etc stand for base type*  
 340 *variables,  $A, B, C$ , etc stand for types with linear output, and  $\sigma, \tau$ , etc stand for PSTA*  
 341 *types. PSTA types are given by the following grammar:*

$$342 \quad A, B, C \quad := \quad \alpha \mid \sigma \multimap A \mid \forall \alpha A \mid A \& B$$

$$343 \quad \sigma, \tau, \rho, \mu, \nu \quad := \quad A \mid !\sigma \mid // \sigma$$

344 *A PSTA Typing context is a set of type assignments  $x : \sigma$ , where  $x$  is a variable and  $\sigma$  a*  
 345 *PSTA type. A PSTA Typing judgment is  $\Gamma \vdash M : \sigma$ , where  $\Gamma$  is a PSTA Typing context,  $M$*   
 346 *is a  $\lambda$ -term, and  $\sigma$  is a PSTA type.*

### 347 3.2 Typing Rules

348 Our PSTA typing rules are the following.

$$349 \quad \frac{}{//\Delta, x : A \vdash x : A} (//Id) \quad \frac{//\Delta, \Gamma \vdash M : \sigma}{//\Delta, !\Gamma \vdash M : !\sigma} (//Sp) \quad \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall \alpha A} (\forall R)$$

$$350 \quad \frac{\Gamma, x_0 : \tau, \dots, x_n : \tau \vdash M : \sigma}{\Gamma, x : !\tau \vdash M[x/x_0, \dots, x/x_n] : \sigma} (m) \quad \frac{\Gamma \vdash M : \sigma_1 \quad \Gamma \vdash M : \sigma_2}{\Gamma \vdash M : \sigma_1 \& \sigma_2} (\&R)$$

$$351 \quad \frac{\Gamma, x : A[B/\alpha] \vdash M : \sigma}{\Gamma, x : \forall \alpha A \vdash M : \sigma} (\forall L) \quad \frac{\Gamma, x_1 : \tau \vdash M : \sigma}{\Gamma, x : //\tau \vdash M[x/x_1] : \sigma} (//D)$$

$$352 \quad \frac{\Gamma, x_1 : \tau_1 \vdash M : \sigma}{\Gamma, x : \tau_1 \& \tau_2 \vdash M[x/x_1] : \sigma} (\&L_1) \quad \frac{\Gamma, x_2 : \tau_2 \vdash M : \sigma}{\Gamma, x : \tau_1 \& \tau_2 \vdash M[x/x_2] : \sigma} (\&L_2)$$

$$353 \quad \frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x. M : \sigma \multimap A} (\multimap R) \quad \frac{//\Delta_1, \Gamma \vdash M : \tau \quad //\Delta_2, \Lambda, x : \tau \vdash N : \sigma}{//\Delta, \Gamma, \Lambda, \vdash N[M/x] : \sigma} (//cut)$$

$$354 \quad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : //\tau \vdash M : \sigma} (//W) \quad \frac{//\Delta_1, \Gamma \vdash M : \tau \quad //\Delta_2, \Lambda, x : A \vdash N : \sigma}{//\Delta, \Gamma, \Lambda, y : \tau \multimap A \vdash N[yM/x] : \sigma} (//\multimap L)$$

355  
 356  
 357 As exemplified in the subject reduction property, typing an application term ( $MN$ ) is  
 358 done with the ( $// \multimap L$ ) rule. In the typing rules above, we also add the following side  
 359 conditions:

- 360 ■ Parallel types occur only with negative polarity in the typing judgments,
- 361 ■ In rules ( $//cut$ ) and ( $// \multimap L$ ), the domain of contexts  $\Gamma$  and  $\Lambda$  are disjoint, and finally
- 362 ■ In rules ( $//cut$ ) and ( $// \multimap L$ ), the side condition  $\mathcal{S}_P : //\Delta_1 \subsetneq //\Delta, //\Delta_2 \subsetneq //\Delta$  holds.

363 Moreover, we also define a generalized ( $//, \multimap cut$ ) rule similar to that of PSELL, with the  
 364 appropriate nesting of substitutions for all ( $cut$ ) and  $\multimap$  pairs of terms.

365 These rules being literal translations of that of PSELL, the rule permutations, and ( $cut$ )-  
 366 elimination steps of PSELL apply to PSTA.

367 The grammar of our types, and the typing rules, together with the side conditions above,  
 368 ensure that sharing does not occur in our typing system. More precisely, we have

369 ► **Proposition 21.** *Let  $\Pi$  be a typing derivation with conclusion  $\Gamma \vdash M : !\sigma$ . Then, the*  
 370 *context  $\Gamma$  is  $//\Delta, !\Lambda$ .*

371 A corollary of Proposition 21 is

372 ► **Corollary 22.** *Any typing derivation with conclusion  $//\Delta, !\Gamma \vdash M : !\sigma$  ends with a ( $//Sp$ ),*  
 373 *( $m$ ), ( $//D$ ), ( $//W$ ) or a ( $//cut$ ). Moreover, in this context, the rules ( $m$ ), ( $//D$ ), and ( $//W$ )*  
 374 *can be commuted to the top (since the premise needs to have a modal context as well), and*  
 375 *the derivation can w.l.o.g. be considered to end with a ( $//Sp$ ) or a ( $//cut$ )-rule.*

376 From the absence of sharing, we derive

377 ► **Proposition 23.** *PSTA enjoys the subject reduction property: if  $\Gamma \vdash M : \sigma$  and  $M \rightarrow_\beta M'$ ,*  
378 *then  $\Gamma \vdash M'$ .*

379 **PROOF.** By structural induction on the cut-type  $\sigma$  of the term  $\lambda y.P$  in the redex  $(\lambda y.P Q)$ .  
380 The terms  $M$  and  $M'$  can be written as  $M = N[(x Q)/z][\lambda y.P/x]$  and  $M' = N[P[Q/y]/z]$ .  
381 Two cases arise:

382 1.  $\sigma = \tau \rightarrow A$ . The derivation is

$$383 \frac{\frac{\frac{\text{//}\Delta_1, \Gamma_1, y : \tau \vdash P : A}{\text{//}\Delta_1, \Gamma_1 \vdash \lambda y.P : \tau \rightarrow A} \quad \frac{\text{//}\Delta_3, \Gamma_2, \vdash Q : \tau \quad \text{//}\Delta_4, \Gamma_3, z : A \vdash N : \sigma}{\text{//}\Delta_2, \Gamma_2, \Gamma_3, x : \tau \rightarrow A \vdash N[(x Q)/z] : \sigma} (\text{// } \rightarrow L)}{\text{//}\Delta, \Gamma_1, \Gamma_2, \Gamma_3 \vdash N[(x Q)/z][\lambda y.P/x] : \sigma} (\text{// cut})$$

384 Cut elimination yields then the following derivation tree

$$385 \frac{\frac{\text{//}\Delta_1, \Gamma_1, y : \tau \vdash P : A \quad \text{//}\Delta_3, \Gamma_2, \vdash Q : \tau \quad \text{//}\Delta_4, \Gamma_3, z : A \vdash N : \sigma}{\text{//}\Delta, \Gamma_1, \Gamma_2, \Gamma_3 \vdash N[P[Q/y]/z] : \sigma} (\text{// cut})$$

386 which proves the subject reduction property. If  $\sigma = \forall \alpha \tau$  or  $\sigma = \tau \& \tau'$ , the cut-elimination  
387 steps eventually reduce to the case above.

388 2.  $\sigma = !\tau$ . By Proposition 21, and modulo rule permutations the derivation is

$$389 \frac{\frac{\frac{\text{//}\Delta_1, \Gamma_1 \vdash \lambda y.P : \tau}{\text{//}\Delta_1, !\Gamma_1 \vdash \lambda y.P : !\tau} (\text{// } Sp) \quad \frac{\text{//}\Delta_2, \Gamma_2, \dots x_i : \tau \dots \vdash N[\dots (x_i Q)/z_i \dots] : \sigma}{\text{//}\Delta_2, \Gamma_2, x : !\tau \vdash N[(x Q)/z_1, \dots, (x Q)/z_n] : \sigma} (m)}{\text{//}\Delta, !\Gamma_1, \Gamma_2 \vdash N[(x Q)/z_1, \dots, (x Q)/z_n][\lambda y.P/x] : \sigma} (\text{// cut})$$

390 Cut elimination yields then the following derivation tree

$$391 \frac{\frac{\overbrace{\dots \text{//}\Delta_1, \Gamma'_i \vdash \lambda y.P_i : \tau \dots}^{n \text{ copies}} \quad \text{//}\Delta_2, \Gamma_2, \dots x_i : \tau \dots \vdash N[\dots (x_i Q)/z_i \dots] : \sigma}{\text{//}\Delta, \Gamma'_1, \dots, \Gamma'_n, \Gamma_2 \vdash N[(x_1 Q)/z_1, \dots, (x_n Q)/z_n][\lambda y.P_1/x_1, \dots, \lambda y.P_n/x_n] : \sigma} (\text{// cut})}{\text{//}\Delta, !\Gamma_1, \Gamma_2 \vdash N[(x_1 Q)/z_1, \dots, (x_n Q)/z_n][\lambda y.P_1/x_1, \dots, \lambda y.P_n/x_n] : \sigma} (m)$$

392 and the induction hypothesis applies to the  $n$  cut-types  $\tau$ .

### 393 3.3 A Parallel, Polynomial Time Evaluation Strategy

394 ► **Theorem 24.** *Let  $T$  be a  $\lambda$ -term, typable in PSTA. Then,  $T$  normalizes in polynomial*  
395 *parallel time.*

396 **PROOF.**

397 The proof follows from Theorem 18: cut-elimination in parallel polynomial time, and  
398 subject-reduction, induce a parallel polynomial number of  $\beta$ -reduction steps for the term.  
399 The overall complexity bound is however a bit more subtle: while PSTA type derivations  
400 have exponential size and polynomial depth, the corresponding right-hand side  $\lambda$ -terms may  
401 have syntactic trees of exponential depth as well. Performing the substitutions for each  
402  $\beta$ -reduction step in parallel polynomial time requires then to use an appropriate, polynomial  
403 space representation of the terms: the explicit representation is clearly unsuitable.

404 Let us first introduce some definitions and observations.

405 Let  $T$  be a  $\lambda$ -term. Its Böhm-like tree  $B(T)$  is defined as follows:

- 406 1. If  $T$  is a variable  $x$ ,  $B(T)$  is a single vertex labelled with  $x$ .  
407 2. If  $T$  is an abstraction  $\lambda x.U$ ,  $B(T)$  is obtained adding  $B(U)$  as a leftmost child of a root  
408 labelled with  $\lambda x$ .

## 10:12 Parallelism in Soft Linear Logic

409 3. If  $T$  is an application  $UV$ ,  $B(T)$  is obtained by adding  $B(V)$  as a new rightmost child of  
 410 the root of  $B(U)$ .

411 Clearly, a Böhm-like tree  $B(T)$  uniquely defines a term  $T$ . Therefore, in the sequel we identify  
 412 the two notions, and focus on the computation of the Böhm-like tree of the normal-form of a  
 413 given term.

414 Let  $T$  be a  $\lambda$ -term, typable in PSTA with a typing derivation  $\Pi$ . We define the *pseudo*-  
 415 derivation  $D(\Pi)$  associated to  $\Pi$  as the tree obtained from  $\Pi$  by removing all right-hand side  
 416  $\lambda$ -terms (while keeping the corresponding type).

417 Then, the following observations hold.

- 418 1. In each typing judgment in  $\Pi$  (and therefore in  $D(\Pi)$ ), the typing context contains type  
 419 assignments for variable terms only.
- 420 2. Erasing the variable names in the contexts of  $D(\Pi)$  (while keeping the corresponding  
 421 types) yields a PSSL proof  $L(\Pi)$ , with types as formulae,
- 422 3. All right-hand side  $\lambda$ -terms in  $\Pi$  are uniquely determined by  $D(\Pi)$ , and finally,
- 423 4. The variable type assignments in  $D(\Pi)$  are preserved by the subject-reduction property:  
 424 If  $T_1$  is a  $\lambda$ -term with PSTA type derivation  $\Pi_1$ ,  $T_1 \rightarrow_\beta T_2$ , and  $\Pi_2$  is the type derivation  
 425 of  $T_2$  obtained by the subject reduction steps of Proposition 23, then the variable type  
 426 assignments in  $D(\Pi_1)$  and  $D(\Pi_2)$  coincide.

427 As a consequence, the following reduction strategy holds: from a  $\lambda$ -term  $T$  with PSTA  
 428 typing derivation  $\Pi$ , perform the innermost parallel cut-elimination strategy on  $L(\Pi)$ , while  
 429 keeping the variable type assignments given by  $D(\Pi)$ . The observations above ensure that  
 430 the pseudo derivation  $D(\Pi')$  thus obtained is that of the typing derivation  $\Pi'$  of the normal  
 431 form  $T'$  of  $T$ . The additional information stored in the contexts of  $D(\Pi')$  (the variable  
 432 names) takes polynomial space (polynomially many variable names among an exponential  
 433 number of possible names), and the reduction can be performed in parallel polynomial time.  
 434 It remains to show how to compute the normal form  $T'$  from its pseudo-derivation  $D(\Pi')$ , in  
 435 parallel polynomial time. We do this by actually computing a succinct representation of its  
 436 Böhm-like tree  $B(T')$ .

437 Let  $D(\Pi)$  be the pseudo-derivation of a PSTA derivation  $\Pi$ , with corresponding term  
 438  $T$  with Böhm-like tree  $B(T)$ . A first observation is the following: For any typing judgment  
 439  $\Gamma \vdash t : \sigma$  in  $\Pi$ , if the explicit substitution  $[M/x]$  (respectively  $[yM/x]$ ) occurs in  $t$ , then there  
 440 exists a judgment  $\Gamma' \vdash M : \sigma'$  above in  $\Pi$ . Since  $\Pi$  has polynomial depth, and polynomial  
 441 indegree by Lemma 16, the substitution term  $M$  can then be described in polynomial space  
 442 by the path from the conclusion of  $\Pi$  to this typing judgment  $\Gamma' \vdash M : \sigma'$ .

443 For each typing judgment  $\Gamma \vdash t : \sigma$  in  $\Pi$ , we associate to the right-hand term the following:

- 444 1. the path  $p$  from the conclusion of  $\Pi$  to this judgment, and
- 445 2. the list  $s(p)$  of explicit substitutions occurring along  $p$ , computed as follows:
  - 446 – assume  $p$  chooses the rightmost premise  $N$  in a ( $//$ ,  $\multimap$  cut) rule  $R$  (i.e. the premise  $p'$   
 447 s.t.  $R$  has no (cut) or  $\multimap$ -pair  $p' \rightarrow p''$ ): this cut-rule introduces a polynomial number  
 448 of substitutions  $[M_i/x_i]$  (or  $[y_i M_i/x_i]$ ) in its conclusion term. Then, we add to  $s(p)$   
 449 the pairs  $(p_i, x_i)$  (or  $(y_i p_i, x_i)$ ), where  $p_i$  is the path to the corresponding premise with  
 450 right hand term  $M_i$ .
  - 451 – assume  $p$  passes through a ( $m$ ) ( $//D$ ) or ( $\&L_i$ ). Then, we add to  $s(p)$  the pairs  $(x, x_i)$ .

452 Clearly, for a path  $p$ , the list  $s(p)$  has polynomial size, and can be computed in polynomial  
 453 time. Now, for a given path  $p$ , the computation of the corresponding vertex  $v(p)$  in  $B(T)$   
 454 proceeds co-recursively on  $D(\Pi)$  as follows:

- 455 – if  $p$  is conclusion of a ( $//ax$ ) rule,  $v(p)$  is a leaf in  $B(T)$ , with label  $x$ .

- 456 ■ if  $p$  is conclusion of a ( $//Sp$ ), ( $\forall R$ ), ( $\forall L$ ), ( $//W$ ) or ( $\&R$ ) rule, with premise  $p'$ , then  $v(p)$   
 457 is  $v(p')$ .
- 458 ■ if  $p$  is conclusion of a ( $m$ ), ( $//D$ ) or ( $\&L_i$ ) rule with premise  $p'$ , two cases arise:
- 459 1.  $v(p')$  is labelled  $x_i$ : then,  $(x/x_i)$  belongs to  $s(p)$ . In that case  $v(p)$  is labelled  $x$ , and  
 460 its successors are those of  $v(p')$ .
- 461 2. otherwise,  $v(p)$  is  $v(p')$ .
- 462 ■ if  $p$  is conclusion of a ( $\neg R$ ) rule with premise  $p'$ ,  $v(p)$  is an inner node labelled  $\lambda x$ , with  
 463 left successor node  $p'$ .
- 464 ■ if  $p$  is conclusion of a ( $//, \neg$  cut)-rule  $R$ : let  $p'$  be its rightmost premise. Then, three  
 465 cases arise:
- 466 1.  $v(p')$  is labelled  $x$ ,  $(p'', x)$  belongs to  $s(p')$ : then,  $v(p)$  is obtained from  $v(p'')$  by adding  
 467 to its root the successor vertices of  $v(p')$ .
- 468 2.  $v(p')$  is labelled  $x$ ,  $(yp'', x)$  belongs to  $s(p')$ : then,  $v(p)$  is a vertex labelled  $y$ , with  
 469 right successor  $v(p'')$ .
- 470 3. otherwise,  $v(p)$  is  $v(p')$ .
- 471 Performing the procedure above in parallel for all paths in  $D(\Pi)$  provides then a succinct  
 472 description of  $B(T)$  in parallel polynomial time.

#### 473 **4** Completeness of PSTA

474 We now prove that PSTA is complete for the class FPAR of functions computable in parallel,  
 475 polynomial time. In order to do so, we first encode parallel, polynomial time recursive  
 476 functions with substitutions, à la Leivant and Marion [16], and then use them to simulate  
 477 the computation of a P-uniform family of boolean circuits of polynomial depth. Extending  
 478 these encodings to the setting of algebraic complexity amounts then simply to replace the  
 479 base type  $\mathbf{B}$  by a base type for the underlying algebraic structure (e.g. real numbers), and  
 480 to provide the type of the algebraic constants and operations in the typing context.

481 First, PSTA captures (obviously) STA.

482 ► **Lemma 25.** *Let  $\Pi$  be a SLL proof of degree  $d$  and rank  $n$ , with conclusion  $\Gamma \vdash A$  of size  $s$ .  
 483 Let  $W_\Pi$  be its weight, as defined in [13]. Then, any path in from the conclusion of  $\Pi$  to an  
 484 axiom contains at most  $s + W_\Pi(1)$  ( $\neg L$ ) rules, and at most  $W_\Pi(1) \cdot n^d$  (cut) rules.*

485 ► **Corollary 26.** *Let  $\Pi$  be a SLL proof of degree  $d$  and rank  $n$ , with conclusion  $\Gamma \vdash A$  of size  
 486  $s$ . Then, there exists a PSELL proof  $\Pi'$  with conclusion  $\Delta, \Gamma \vdash A$ , of degree  $d$  and rank  $n$ .*

487 **PROOF.** Take  $\Delta = !^d // A_1, \dots, !^d // A_k$ , with  $k = W_\Pi(1) + s$ , for any  $A_1, \dots, A_k$ .

488 An immediate consequence is that all  $\lambda$ -terms typable in STA are also typable in PSTA,  
 489 with the same rank and degree. As a consequence, following [7], Theorem 19, we immediately  
 490 have that PSTA is complete for FPTIME. This allows us to prove its FPAR completeness  
 491 more easily. Denote by  $\mathbf{B}$  the STA (hence PSTA) type for booleans,  $\mathbf{L}$  the STA type for  
 492 binary strings, and  $\mathbf{N}$  the STA type for Church Integers.

493 The following lemma allows us to encode some sort of polynomial recursion with substi-  
 494 tutions à la Leivant and Marion [16] in PSTA.

495 ► **Lemma 27.** *Assume we have the following sequential, polynomial time functions, with  
 496 PSTA type derivations:*

- 497 ■  $\text{op}$  with derivation  $\Pi_{\text{op}}$  with conclusion  $\Gamma_{\text{op}} \vdash \text{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L}$ .
- 498 ■  $\mathbf{s}_1$  and  $\mathbf{s}_2$  with derivation  $\Pi_i$ , for  $i \in \{1, 2\}$ , with conclusion  $\Gamma_i \vdash \mathbf{s}_i : \mathbf{L} \multimap \mathbf{L}$ .

## 10:14 Parallelism in Soft Linear Logic

499 ■ and, for any univariate polynomial  $P$  of degree  $d$ , a function  $\bar{P}$ , encoding the church  
500 integer  $P(|L|)$  for a binary list  $L$ , with derivation  $\Pi_{\bar{P}}$  with conclusion  $\Gamma_{\bar{P}} \vdash \bar{P} : !^d \mathbf{L} \multimap \mathbf{N}$ .  
501 We now consider the following recursive function with substitutions, on binary lists:  $f(v) =$   
502  $\text{op}(v, f(\mathbf{s}_1(v)), f(\mathbf{s}_2(v)))$ . Moreover, we assume that on any input  $v$ , the recursive computation  
503 of  $f$  reaches a fixed point after  $P(|v|)$  steps. Then,  $f(v)$  is PSTA definable with degree  $d$ .

504 PROOF. Following a similar encoding in [6], each recursion step in the computation  
505 of  $f$  is encoded by the following function  $\text{Step} = \lambda h. \lambda v. \text{op } v (h (\mathbf{s}_1 v)) (h (\mathbf{s}_2 v))$ . Let  
506  $\mathbf{L2} = (\mathbf{L} \multimap \mathbf{L}) \multimap \mathbf{L} \multimap \mathbf{L}$ , and  $\Gamma = \Gamma_{\text{op}}, \Gamma_1, \Gamma_2, (X : //A)^2, h : //(\mathbf{L} \multimap \mathbf{L}), v : //\mathbf{L}$ . Then,  $\text{Step}$   
507 admits a PSTA proof derivation  $\Pi_{\text{Step}}$  with conclusion  $\Gamma \vdash \text{Step} : \mathbf{L2}$ .

508 Indeed,  $\Pi_{\text{Step}}$  is

$$509 \frac{\frac{A_{\text{op}} \quad A_1 \quad A_2 \quad A_{\mathbf{s}1} \quad A_{\mathbf{s}2} \quad A_{\mathbf{v}1} \quad A_{\mathbf{v}2} \quad A_{\text{step}}}{\Gamma, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \text{op } v (h (\mathbf{s}_1 v)) (h (\mathbf{s}_2 v)) : \mathbf{L}}}{\Gamma \vdash \text{Step} : \mathbf{L2}} (\multimap R)^2 \quad (//cut)$$

510 where  $A_{\text{op}}$  is  $\frac{\Pi_{\text{op}}}{\Gamma_{\text{op}} \vdash \text{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L}}$ ,  $A_i$  is  $\frac{\Pi_i}{\Gamma_i \vdash \mathbf{s}_i : \mathbf{L} \multimap \mathbf{L}}$ ,

511  
512  $A_{\mathbf{s}i}$  is  $\frac{\frac{\frac{v : \mathbf{L} \vdash v : \mathbf{L}}{X : //A, \mathbf{t}_i : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \mathbf{t}_i v : \mathbf{L}}{X : //A, \mathbf{t}_i : \mathbf{L} \multimap \mathbf{L}, v : //\mathbf{L} \vdash \mathbf{t}_i v : \mathbf{L}} (//D)}{\frac{v : \mathbf{L} \vdash v : \mathbf{L}}{X : //A, \mathbf{t}_i : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \mathbf{t}_i v : \mathbf{L}} (//Id)} (// \multimap L)} (//Id)$ ,

513  
514  $A_{\mathbf{v}1}$  is  $\frac{\frac{\frac{v : \mathbf{L} \vdash v : \mathbf{L}}{X : //A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h v : \mathbf{L}}{X : //A, h : //\mathbf{L} \multimap \mathbf{L}, v : //\mathbf{L} \vdash h v : \mathbf{L}} (//D)^2}{\frac{v : \mathbf{L} \vdash v : \mathbf{L}}{X : //A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h v : \mathbf{L}} (//Id)} (// \multimap L)} (//Id)$ ,

515  
516  $A_{\mathbf{v}2}$  is  $\frac{\frac{v : \mathbf{L} \vdash v : \mathbf{L}}{X : //A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h v : \mathbf{L}} (//Id)}{\frac{v : \mathbf{L} \vdash v : \mathbf{L}}{X : //A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h v : \mathbf{L}} (// \multimap L)} (//Id)$ ,

517  
518 and  $A_{\text{step}}$  is

$$519 \frac{\frac{V_1 : \mathbf{L} \vdash V_1 : \mathbf{L}}{X : //A, V_1 : \mathbf{L}, V_2 : \mathbf{L}, V_3 : \mathbf{L}, \text{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \vdash \text{op } V_1 V_2 V_3 : \mathbf{L}} (//Id)}{\frac{V_1 : \mathbf{L} \vdash V_1 : \mathbf{L}}{X : //A, V_1 : \mathbf{L}, V_2 : \mathbf{L}, V_3 : \mathbf{L}, \text{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \vdash \text{op } V_1 V_2 V_3 : \mathbf{L}} (//Id)} (//Id)} \dots \frac{x_3 : \mathbf{L} \vdash x_3 : \mathbf{L}}{X : //A, V_1 : \mathbf{L}, V_2 : \mathbf{L}, V_3 : \mathbf{L}, \text{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \vdash \text{op } V_1 V_2 V_3 : \mathbf{L}} (//Id)} (//, \multimap cut)$$

520 The value  $f(v)$  is reached after  $P(|v|)$  recursion steps. It is given by  $\text{Value } v$ , where  
521  $\text{Value} = \lambda v. ((\bar{P} v) \text{Step } \lambda y. y) v$ . Let  $\Gamma' = \Gamma_{\text{op}}, \Gamma_1, \Gamma_2, (X : //A)^5, h : //(\mathbf{L} \multimap \mathbf{L}), v :$   
522  $//\mathbf{L}, \Gamma_{\bar{P}}, v : !^d \mathbf{L}$ . Then,  $\text{Value}$  admits a PSTA proof derivation  $\Pi_{\text{Value}}$  with conclusion  $\Gamma' \vdash$   
523  $\text{Value} : \mathbf{L} \multimap \mathbf{L}$ . Indeed, let  $\mathbf{L3} = (!\mathbf{L2} \multimap \mathbf{L2})$ . Recall that  $\mathbf{N} = \forall \alpha! (\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$  and  
524 consider the following proof derivations.

$$525 \frac{\frac{\Pi_{\bar{P} v}}{\Gamma_{\bar{P}} \vdash \bar{P} : !^d \mathbf{L} \multimap \mathbf{N}}}{\frac{\frac{\frac{v : !^d \mathbf{L} \vdash v : !^d \mathbf{L}}{x : //A, \bar{P} : !^d \mathbf{L} \multimap \mathbf{N}, v : !^d \mathbf{L} \vdash \bar{P} v : \mathbf{N}}{x : //A, \bar{P} : !^d \mathbf{L} \multimap \mathbf{N}, v : !^d \mathbf{L} \vdash \bar{P} v : \mathbf{N}} (//Id)}{\frac{v : !^d \mathbf{L} \vdash v : !^d \mathbf{L}}{x : //A, \bar{P} : !^d \mathbf{L} \multimap \mathbf{N}, v : !^d \mathbf{L} \vdash \bar{P} v : \mathbf{N}} (// \multimap L)} (//Id)} (//cut)} (//cut)$$

$$527 \frac{\frac{\Pi_{\mathbf{s}}}{\Gamma_{\text{op}}, \Gamma_1, \Gamma_2, (X : //A)^3, h : //(\mathbf{L} \multimap \mathbf{L}), v : //\mathbf{L}, \Gamma_{\bar{P}}, v : !^d \mathbf{L} \vdash \bar{P} v \text{Step} : \mathbf{L2}}{\frac{\frac{\frac{! \text{Step} : !\mathbf{L2} \vdash ! \text{Step} : !\mathbf{L2} \quad x : !\mathbf{L2} \vdash x : !\mathbf{L2}}{X : //A, (\bar{P} v) : \mathbf{L3}, ! \text{Step} : !\mathbf{L2} \vdash \bar{P} v \text{Step} : \mathbf{L2}} (// \multimap L)}{\frac{! \text{Step} : !\mathbf{L2} \vdash ! \text{Step} : !\mathbf{L2} \quad x : !\mathbf{L2} \vdash x : !\mathbf{L2}}{X : //A, (\bar{P} v) : \mathbf{N}, ! \text{Step} : !\mathbf{L2} \vdash \bar{P} v \text{Step} : \mathbf{L2}} (\forall L)} (//cut)} (//cut)} (//cut)$$

$$\begin{array}{l}
529 \quad \Pi_{sl}: \\
530 \quad \frac{\frac{\lambda y.y : \mathbf{L} \multimap \mathbf{L} \vdash \lambda y.y : \mathbf{L} \multimap \mathbf{L} \quad f : \mathbf{L} \multimap \mathbf{L} \vdash f : \mathbf{L} \multimap \mathbf{L}}{\Pi_s \quad X : //A, \overline{P} v \text{ Step} : \mathbf{L2} \vdash \overline{P} v \text{ Step} \lambda y.y : \mathbf{L} \multimap \mathbf{L}} (\// \multimap L)}{\Gamma_{\text{op}}, \Gamma_1, \Gamma_2, (X : //A)^4, h : //(\mathbf{L} \multimap \mathbf{L}), v : //\mathbf{L}, \Gamma_{\overline{P}}, v : !^d \mathbf{L} \vdash \overline{P} v \text{ Step} \lambda y.y : \mathbf{L} \multimap \mathbf{L}} (\// \text{cut}) \\
531 \quad \text{and finally } \Pi_{\text{value}}: \\
532 \quad \frac{\frac{\frac{v : \mathbf{L} \vdash v : \mathbf{L} \quad z : \mathbf{L} \vdash z : \mathbf{L}}{\Pi_{sl} \quad X : //A, \overline{P} v \text{ Step} \lambda y.y : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash (\overline{P} v \text{ Step} \lambda y.y)v : \mathbf{L}} (\// \multimap L)}}{\Gamma', v : \mathbf{L} \vdash (\overline{P} v \text{ Step} \lambda y.y)v : \mathbf{L}} (\// \text{cut})}{\Gamma' \vdash \text{Value} : \mathbf{L} \multimap \mathbf{L}} (\multimap R) \\
533
\end{array}$$

534 ► **Theorem 28.** *PSTA is complete for FPAR.*

535 **PROOF.** Using the usual encodings for binary strings, booleans, integers and pairs, we  
536 use Lemma 27 to prove our completeness result. Let  $g$  be a function computed in FPAR.  
537 For the sake of simplicity let us assume that  $g$  outputs a single boolean. Then, there exists a  
538  $P$ -uniform family  $\mathcal{C}$  of succinctly described boolean circuits, of polynomial depth, computing  
539  $g$ . More precisely, there exist a univariate polynomial  $p$ , and polynomial time functions **and**,  
540 **or**, **node**, **input**,  $\mathbf{s}_1$  and  $\mathbf{s}_2$  such that:

- 541 ■ On any input  $x = x_1, \dots, x_n$  of size  $n$ ,  $g(x_1, \dots, x_n)$  is computed by a boolean circuit
- 542  $C_n$  of depth  $p(n)$ , with output node  $t$ .
- 543 ■ For each node  $s$  in  $C_n$ , there exists a binary list  $n_s$ , encoding a path from  $t$  to  $s$  in  $C_n$ , of
- 544 length less than  $p(n)$ . Each node will be identified by these paths (there may be several
- 545 paths for a given node).
- 546 ■ **and**( $x, y$ ) (respectively **or**( $x, y$ ), resp. **not**( $x, y$ )) is **true** if the path  $y$  encodes a **and** (resp.
- 547 **or**, resp. **not**) node of  $C_{|x|}$ , and **false** otherwise.
- 548 ■ **input**( $x, y$ ) is  $(x_i, \text{true})$  if  $y$  encodes the  $i^{\text{th}}$  input node of  $C_{|x|}$ , and  $(0, \text{false})$  otherwise.
- 549 ■  $\mathbf{s}_1(y) = 0.y$  encodes a path to the left parent of the node encoded by  $y$ , if it exists.
- 550 ■  $\mathbf{s}_2(y) = 1.y$  encodes a path to the right parent of the node encoded by  $y$ , if it exists.

551 Define now  $f(x, y) = \text{op}(x, y, f(x, \mathbf{s}_1(y)), f(x, \mathbf{s}_2(y)))$ , where  $y$  denotes a path in  $C_{|x|}$ , and  
552  $\text{op}(x, y, v_1, v_2)$  computes, using the functions defined above, the boolean value of the node  $y$   
553 in  $C_{|x|}$ , provided  $v_1$  and  $v_2$  are the boolean values of its two parents nodes. Then, Lemma 27  
554 applies:  $f$  is definable in PSTA, and recursively computes the value of all nodes in  $C_{|x|}$ . The  
555 output  $g(x)$  is then given by  $f(x, \epsilon)$ , where  $\epsilon$  is the empty binary list.

## 566 5 Concluding Remarks

557 In this paper we have only investigated one of the many possible choices for the way the  
558 parallel ( $//, \multimap \text{cut}$ ) rule allows contraction on  $//$  formulas, and allows its distribution among  
559 the premises of the cut, and we have applied this approach to one example (STA) of linear  
560 typing system. Among the questions now worth investigating are the following: Is it possible  
561 to tune differently the side condition of the ( $//, \multimap \text{cut}$ )-rule to capture other complexity  
562 classes? Such obvious candidates are the classes  $NC^i$ , which we could hope to capture  
563 by taking a fully linear ( $//, \multimap \text{cut}$ )-rule (for ensuring sequential polynomial time), with an  
564 additional side condition ensuring parallel polylogarithmic time cut-elimination. Is it also  
565 possible to use this approach on type systems capturing other sequential complexity classes,  
566 for instance Logspace [17, 15], and to obtain other interesting results?



- 
- 568 1 Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: A language for polynomial time  
569 computation. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer  
570 Science*, pages 27–41. Springer, 2004.
  - 571 2 Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda  
572 calculus. *Information and Computation*, 207(1):41–62, jan 2009. doi:10.1016/j.ic.2008.08.  
573 005.
  - 574 3 Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the  
575 polytime functions. *Computational Complexity*, 2:97–110, 1992.
  - 576 4 Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Two function  
577 algebras defining functions in  $NC^k$  boolean circuits. *Inf. Comput.*, 248:82–103, 2016. doi:  
578 10.1016/j.ic.2015.12.009.
  - 579 5 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of  
580 pspace. In George C. Necula and Philip Wadler, editors, *POPL*, pages 121–131. ACM, 2008.
  - 581 6 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. An implicit charac-  
582 terization of PSPACE. *ACM Transactions on Computational Logic*, 13(2):1–36, apr 2012.  
583 doi:10.1145/2159531.2159540.
  - 584 7 Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for lambda  
585 -calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture  
586 Notes in Computer Science*, pages 253–267. Springer, 2007.
  - 587 8 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
  - 588 9 Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.
  - 589 10 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular  
590 approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.
  - 591 11 Paulin Jacobé De Naurois. Pointers in recursion: Exploring the tropics. 2019. doi:10.4230/  
592 LIPICS.FSCD.2019.29.
  - 593 12 Satoru Kuroda. Recursion schemata for slowly growing depth circuit classes. *Computational  
594 Complexity*, 13(1-2):69–89, 2004. doi:10.1007/s00037-004-0184-4.
  - 595 13 Yves Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180,  
596 2004.
  - 597 14 Ugo Dal Lago and Martin Hofmann. Bounded linear logic, revisited. *Logical Methods in  
598 Computer Science*, 6(4), dec 2010. doi:10.2168/lmcs-6(4:7)2010.
  - 599 15 Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *Programming  
600 Languages and Systems*, pages 205–225. Springer Berlin Heidelberg, 2010. doi:10.1007/  
601 978-3-642-11957-6\_12.
  - 602 16 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity ii:  
603 Substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL*, volume 933 of  
604 *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994. doi:10.1007/BFb0022242.
  - 605 17 Ulrich Schöpp. Stratified bounded affine logic for logarithmic space. In *LICS*, pages 411–420.  
606 IEEE Computer Society, 2007.