From the proceedings of Trends in Linear Logic and Applications 2021 (TLLA 2021, eds E. Pimentel, Ch. Retoré)
https://hal.archives-ouvertes.fr/TLLA2021

Parallelism in Soft Linear Logic

Paulin Jacobé de Naurois*

Abstract

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

1 Introduction

Implicit Complexity aims at providing purely syntactical, machine independent criteria on programs, in order to ensure they respect some complexity bounds upon execution. In the context of functional programming, the use of tailored proof systems, and subsequent type systems for λ-calculus, has been very successful: using subsystems of Linear Logic [Gir87], several proof systems have been proposed, where cut-elimination has a bounded complexity. Consequently, under the Curry-Howard isomorphism, type systems for λ-calculus based on these logics have been proposed, where β-normalization of the typed terms follows the same complexity bounds. Such results include, among many others, Bounded Linear Logic [GSS92, LH10] and Light Linear Logic [Gir98, BT09] for polynomial time computation, and Stratified Bounded Affine Logic [Sch07, LS10] for logarithmic space computations. Our interest in this paper lies in the Soft Linear Logic of Lafont [Laf04], which proposes a simple and elegant approach for ensuring polynomial time bounds by controlling contractions on exponential formulas, and in the subsequent type systems for polynomial time λ-calculus [BM04, GR07, GMR08].

*CNRS, Université Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, F-93430 Villetaneuse, France. denaurois@lipn.univ-paris13.fr

At this point, it is relevant to note that the complexity classes captured thus far are all sequential, deterministic *in essence*. While Soft Linear Logic type systems have been extended to express the classes NP and PSPACE [GMR12], it is important to note that the construction relies on Soft Type Assignment (STA), a deterministic, sequential polynomial time type system, by extending the λ-calculus with an additional construct (`if then else`), for which an ad hoc, alternating polynomial time evaluation strategy is imposed - the core of the language retaining its sequential polynomial time evaluation. While being indeed extensionally complete for PSPACE, this approach lacks intensionality: many natural algorithms, that are easily computable in parallel, are hardly expressible in this setting. Let us take as simple example the numerical evaluation of a balanced, arithmetic expression on bounded integer values. In order to compute it in alternating polynomial time with the (`if then else`) defined in [GMR12], one would need to express the value of all bits of the result as boolean expressions on the bits of the input numbers, and use the alternating evaluation of the (`if then else`) construct to speed up the parallel computation time - not quite a practical method. Furthermore, this approach is no longer doable in real world functional programming languages, where integers are given as a base type, and arithmetical operations as unitary functions of the language.

A reason why these approaches are all essentially sequential, deterministic is that they use the typing discipline to control the *amount* of resources the calculus uses (e.g. by controlling contractions on exponentials), not the *way* these resources are distributed along the computation. In order to truly denote parallel computation in a functional programming language, our proposal here is to use a parallel, call-by-value evaluation strategy for the λ-calculus: in an application, both terms can be normalized in parallel, before the substitution of the redex takes place. If both terms share the same normalization time bound, the parallel evaluation strategy is efficient. Note that in first order functional programming, this is already the approached used by Leivant and Marion [LM94] with their safe recursion with substitutions: using sequential resource bounds from Ptime Safe Recursion [BC92], and a parallel call-by-value evaluation strategy, the authors characterize the class FPAR (Parallel polynomial time), which coincides with PSPACE. This approach has also been later on extended to sub-polynomial complexity classes [Kur04, BKMO16, JDN19]. For higher order functional programming, we rely on the Curry-Howard isomorphism: ensuring an homogeneous computation time on the parallel evaluation of both arguments of an application amounts to ensuring that both premises of a cut-rule share a homogeneous bound on the resource usage in the corresponding type system.

In order to achieve this, we can no longer rely on the usual linear cut-rule. We propose therefore a modification of the linear cut-rule, that internalizes a controlled number of contractions on some formulas, that are uniformly

distributed among the premises. These formulas are decorated with a dedicated modality, called parallel modality. This approach is applied here on the Soft Type Assignment (STA) of Gaboardi and Ronchi [GR07], in order to propose a sound and complete type system for PSPACE, with a truly parallel evaluation strategy. We believe it could be also successfully adapted to other type systems.

2 Parallel Soft Linear Logic

2.1 Soft Linear Logic

Let us recall the SLL rules of Lafont [Laf04], in its intuitionistic fashion. It the following, $(A)^n$ stands for n copies of A in a sequent.

$$\frac{}{U \vdash U}\,(Id) \qquad \frac{\Gamma \vdash U \qquad \Delta, U \vdash V}{\Gamma, \Delta \vdash V}\,(cut) \qquad \frac{\Gamma, U \vdash V}{\Gamma \vdash U \multimap V}\,(\multimap R)$$

$$\frac{\Gamma \vdash U \qquad V, \Delta \vdash Z}{\Gamma, U \multimap V, \Delta \vdash Z}\,(\multimap L) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \& B}\,(\&R)$$

$$\frac{\Gamma, A \vdash V}{\Gamma, A \& B \vdash V}\,(\&L_1) \qquad \frac{\Gamma, B \vdash V}{\Gamma, A \& B \vdash V}\,(\&L_2) \qquad \frac{\Gamma \vdash U}{\Gamma \vdash \forall \alpha U}\,(\forall R)$$

$$\frac{\Gamma, U[V/\alpha] \vdash Z}{\Gamma, \forall \alpha U \vdash Z}\,(\forall L) \qquad \frac{\Gamma \vdash U}{!\Gamma \vdash !U}\,(sp) \qquad \frac{\Gamma, (U)^n \vdash V \qquad n \geq 0}{\Gamma, !U \vdash V}\,(m)$$

In the (m) rule, n is the *rank*. The *rank* of a proof is the maximal rank of its (m) rules, its *degree* is the maximal nesting of ! modalities.

SLL proofs of rank n and degree d normalize in polynomial time n^d.

2.2 Parallel Modalities

PSLL is built upon SLL. An Additional modality, called the parallel modality $/\!\!/$, is introduced with the additional condition, necessary to ensure (cut)-elimination, that $/\!\!/$ sub-formulas occur only negatively in our sequents. The corresponding rules are

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

2.2.1 ($/\!\!/$Cut) and ($/\!\!/ \multimap$) Rules

Contraction for parallel formulas is internalized into the PSLL $(/\!\!/ cut)$-rule, in a controlled fashion. As for the usual cut-rule in linear logic, linear and exponential formulas are linearly distributed among the two premises. The (binary) $(/\!\!/ cut)$ rule is the following,

$$\frac{/\!\!/ \Delta_1, \Gamma_1 \vdash A_1 \qquad /\!\!/ \Delta_2, \Gamma_2, A_1 \vdash A_2 \qquad /\!\!/ \Delta_1 \subsetneq /\!\!/ \Delta,\ /\!\!/ \Delta_2 \subsetneq /\!\!/ \Delta}{/\!\!/ \Delta, \Gamma_1, \Gamma_2 \vdash A_2}\,(/\!\!/ cut)$$

with cut-formula A_1, cut-pair of premises the two premises, and, similarly, the (binary) $(/\!\!/ \multimap)$ rule is the following,

$$\frac{/\!\!/\Delta_1, \Gamma_1 \vdash A_1 \qquad /\!\!/\Delta_2, \Gamma_2, A_2 \vdash A_3 \qquad /\!\!/\Delta_1 \subsetneq /\!\!/\Delta,\ /\!\!/\Delta_2 \subsetneq /\!\!/\Delta}{/\!\!/\Delta, \Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash A_3}\ (/\!\!/ \multimap)$$

with principal $\multimap$-formula $A_1 \multimap A_2$ and $\multimap$-pair of premises its two premises. In a proof tree consisting only in $(n-1)$ binary linear (cut) rules, these (cut) rules can be freely permuted, and a generalized, $n-ary$ linear cut-rule can be derived from the binary ones. The non-linear distribution of parallel modalities in PSLL breaks this isomorphism: permuting two binary $(/\!\!/cut)$ rules in a binary tree of $(/\!\!/cut)$ rules may come in conflict with the side condition $\Delta_i \subsetneq /\!\!/\Delta$. Under some conditions the same remark can be made for $\multimap L$ rules as well, whose structure is similar to the (cut) rule. Since we want a *uniform* bound on the parallel normalization of the premises, we define a n-ary parallel cut-rule as a parallel extension of the linear one, where the side condition for parallel modalities is adapted accordingly: We define linear n-ary cuts, with (cut)-pairs of premises as follows:

Definition 1 (n-ary $(cut/ \multimap L)$ rule) *The following n-ary $(cut/ \multimap L)$*

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

is either a binary $(\multimap L)$ with $\multimap$-pair the pair of its two premises, or a binary (cut) rule with (cut)-pair the pair of its two premises, or it is a n-ary rule obtained by several of the following proof tree $(cut/ \multimap L)$-merge rewriting steps:

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

where the principal formulae of R_1 are not sub-formulae of those of R_2. The multiset of $\multimap$ (respectively (cut)) principal formulae of R is then the union of those of R_1 and R_2. The cut - and $\multimap$-pairs of R are obtained from the union of those of R_1 and R_2 with the following update procedure: whenever T_t belongs to a $\multimap$ or cut-pair of premises $T_t \rightarrow T_w$ (respectively $T_w \rightarrow T_t$) of R_2, with corresponding principal formula F belonging to one of 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. $T_w \rightarrow T_v$), with the same corresponding principal formula.

Now, provided

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

is a valid $n-ary$ rule in SLL we deem the folowing n-ary $(/\!\!/, \multimap cut)$ rule valid for PSLL

$$\frac{/\!\!/\Delta_1, \Gamma_1 \vdash A_1 \qquad /\!\!/\Delta_2, \Gamma_2 \vdash A_2 \cdots \qquad /\!\!/\Delta_d, \Gamma_d \vdash A_d}{/\!\!/\Delta, \Gamma, \Lambda \vdash A_d} (/\!\!/, \multimap \mathit{cut})$$

where the side condition $\forall i = 1, \cdots, d, /\!\!/\Delta_i \subsetneq /\!\!/\Delta$ holds, with the exact same (*cut*)-pairs. These (*cut*)-pairs play a role in our parallel evaluation strategy, and define a forest structure on the premises of a $(/\!\!/, \multimap \mathit{cut})$-rule, called the *pairing forest*.

The logic PSLL is then given by the rules $(\multimap R)$, $(\forall R)$, $(\forall L)$, $(\& R)$, $(\& L_i)$, (m) $(/\!\!/ sp)$, $(/\!\!/ W)$, $(/\!\!/ D)$ and $(/\!\!/, \multimap \mathit{cut})$.

2.3 Parallel Cut Elimination

Restricting $/\!\!/$ modalities to negative sub-formulae only ensures that PSLL enjoys cut elimination - the elimination cases being almost identical to those of SLL. We moreover define a parallel, call-by-value elimination strategy.

Definition 2 (Parallel elimination of an innermost cut) *Le* Π *be a PSLL proof. A* $(/\!\!/, \multimap \mathit{cut})$ *rule* R *with cut-pairs is innermost if there is no other* $(/\!\!/, \multimap \mathit{cut})$ *rule with cut-pairs along any path from* R *to the axioms.*

Let R *be an innermost* $(/\!\!/, \multimap \mathit{cut})$ *rule in* Π*, and* $F(R)$ *be the pairing forest. The parallel elimination of* R *is then the following procedure:*

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

The Innermost parallel cut-elimination *procedure consists in applying in parallel, for all its innermost cuts, their parallel elimination, until no* $(/\!\!/, \multimap \mathit{cut})$ *rule with cut-pairs remains.*

The innermost parallel cut-elimination procedure ensures that the blow-up of the $(/\!\!/, \multimap \mathit{cut})$ rules remains under control, and can be done in parallel polynomial time:

Theorem 3 *Let* Π *be a PSLL proof, of rank* n *and degree* d*, with conclusion sequent* S*. Let* M *be the maximal size of its cut-formulae, and* h *the maximal height of its pairing forests.Then, an innermost parallel cut-elimination strategy takes* $O(|S|.M.h.n^{2d})$ *steps.*

It follows that PSLL is sound for FPSPACE. Completeness is given by the type assignment of the next section.

3 A Parallel Polynomial Time Type Assigment for λ-calculus

We take insipiration from the STA type assignment of [GR07]. We add the parallel modalities in a restricted way, as follows.

Definition 4 *In the following, α, β, etc stand for base type variables, A, B, C, etc stand for types with linear output, and σ, τ, etc stand for PSTA types. The PSTA grammar is:*

$$\begin{array}{rcl} A, B, C & := & \alpha \mid \sigma \multimap A \mid \forall \alpha A \mid A \& B \\ \sigma, \tau, \rho, \mu, \nu & := & A \mid !\sigma \mid \mathbin{/\!\!/} \sigma \end{array}$$

A PSTA Typing context is a set of type assignments $M : \sigma$, where M is a λ-term and σ a PSTA type. A PSTA Typing judgment is $\Gamma \vdash M : \sigma$, where Γ is a PSTA Typing context, M is a λ-term, and σ is a PSTA type.

The PSTA typing rules are the following.

$$\frac{}{\mathbin{/\!\!/}\Delta,\ x :\ A \vdash x :\ A}\ (\mathbin{/\!\!/} Id)$$

$$\frac{\Gamma \vdash M :\ \sigma}{\Gamma,\ x : \mathbin{/\!\!/}\tau \vdash M :\ \sigma}\ (\mathbin{/\!\!/} W) \qquad \frac{\mathbin{/\!\!/}\Delta,\ \Gamma \vdash M :\ \sigma}{\mathbin{/\!\!/}\Delta,\ !\Gamma \vdash M : !\sigma}\ (\mathbin{/\!\!/} Sp)$$

$$\frac{\Gamma,\ x_0 :\ U, \cdots, x_n :\ \tau \vdash M :\ \sigma}{\Gamma,\ x : !\tau \vdash M[x/x_0, \cdots, x/x_n] :\ \sigma}\ (m) \qquad \frac{\Gamma \vdash M :\ A}{\Gamma \vdash M :\ \forall \alpha A}\ (\forall R)$$

$$\frac{\Gamma \vdash M :\ \sigma_1 \qquad \Gamma \vdash M :\ \sigma_2}{\Gamma \vdash M :\ \sigma_1 \& \sigma_2}\ (\& R) \qquad \frac{\Gamma,\ x :\ A[B/\alpha] \vdash M :\ \sigma}{\Gamma,\ x :\ \forall \alpha A \vdash M :\ \sigma}\ (\forall L)$$

$$\frac{\Gamma,\ x :\ \sigma \vdash M :\ A}{\Gamma \vdash \lambda x.M :\ \sigma \multimap A}\ (\multimap R) \qquad \frac{\Gamma,\ x_2 : \tau_2 \vdash M :\ \sigma}{\Gamma,\ x : \tau_1 \& \tau_2 \vdash M[x/x_2] :\ \sigma}\ (\& L_2)$$

$$\frac{\Gamma,\ x_1 : \tau_1 \vdash M :\ \sigma}{\Gamma,\ x : \tau_1 \& \tau_2 \vdash M[x/x_1] :\ \sigma}\ (\& L_1) \qquad \frac{\Gamma,\ x_1 : \tau \vdash M :\ \sigma}{\Gamma,\ x : \mathbin{/\!\!/}\tau \vdash M[x/x_1] :\ \sigma}\ (\mathbin{/\!\!/} D)$$

$$\frac{\mathbin{/\!\!/}\Delta_1,\ \Gamma \vdash M :\ \tau \qquad \mathbin{/\!\!/}\Delta_2,\ \Lambda,\ x :\ \tau \vdash N :\ \sigma}{\mathbin{/\!\!/}\Delta,\ \Gamma,\ \Lambda, \vdash N[M/x] :\ \sigma}\ (\mathbin{/\!\!/} cut)$$

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

With the following additional side conditions:

- Parallel types occur only with negative polarity in the typing judgments,
- In rules ($\mathbin{/\!\!/} cut$) and ($\mathbin{/\!\!/} \multimap L$), the domain of contexts Γ and Λ are disjoint, and finally
- In rules ($\mathbin{/\!\!/} cut$) and ($\mathbin{/\!\!/} \multimap L$), we have $\mathbin{/\!\!/}\Delta_1 \subsetneq \mathbin{/\!\!/}\Delta$, and $\mathbin{/\!\!/}\Delta_2 \subsetneq \mathbin{/\!\!/}\Delta$.

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

Then, PSTA enjoys the subject-reduction property, can be evaluated in parallel polynomial time, and is complete:

Theorem 5 *PSTA is complete for FPAR.*

4 Concluding Remarks

In this paper we have only investigated one of the many possible choices for the way the parallel ($/\!\!/, \multimap$ *cut*) rule allows contraction on $/\!\!/$ formulas, and allows its distribution among the premises of the cut, and we have applied this approach to one example (STA) of linear typing system. Among the questions now worth investigating are the following: Is it possible to tune differently the ($/\!\!/, \multimap$ *cut*) rule to capture other complexity classes? For instance, does it hold that taking the $/\!\!/\Delta_i$ in the premise to have size half of that of $/\!\!/\Delta$ yields some sort of logarithmic, or polylogarithmic parallel time bound? Is it also possible to use this approach on type systems capturing other sequential complexity classes, for instance Logspace [Sch07, LS10], and to obtain other interesting results?

References

[BC92] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[BKMO16] Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Two function algebras defining functions in NC^{k} boolean circuits. *Inf. Comput.*, 248:82–103, 2016.

[BM04] Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: A language for polynomial time computation. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[BT09] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41–62, jan 2009.

[Gir87] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[Gir98] Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.

[GMR08] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. In George C. Necula and Philip Wadler, editors, *POPL*, pages 121–131. ACM, 2008.

[GMR12] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. An implicit characterization of PSPACE. *ACM Transactions on Computational Logic*, 13(2):1–36, apr 2012.

[GR07] Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for *ambda* -calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007.

[GSS92] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.

[JDN19] Paulin Jacobé De Naurois. Pointers in recursion: Exploring the tropics. 2019.

[Kur04] Satoru Kuroda. Recursion schemata for slowly growing depth circuit classes. *Computational Complexity*, 13(1-2):69–89, 2004.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180, 2004.

[LH10] Ugo Dal Lago and Martin Hofmann. Bounded linear logic, revisited. *Logical Methods in Computer Science*, 6(4), dec 2010.

[LM94] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity ii: Substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994.

[LS10] Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *Programming Languages and Systems*, pages 205–225. Springer Berlin Heidelberg, 2010.

[Sch07] Ulrich Schöpp. Stratified bounded affine logic for logarithmic space. In *LICS*, pages 411–420. IEEE Computer Society, 2007.