Parallelism in Soft Linear Logic

Paulin Jacobé de Naurois ✉
CNRS, Université Paris 13, Sorbonne Paris Cité, LIPN, UMR 7030, F-93430 Villetaneuse, France

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.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Complexity theory and logic

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

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.26

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

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 [8], 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 [10, 14] and Light Linear Logic [9, 2] for polynomial time computation, and Stratified Bounded Affine Logic [17, 15] for logarithmic space computations. Our interest in this paper lies in the Soft Linear Logic of Lafont [13], 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 [1, 7, 5].

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 [6], 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 [6], 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. Our approach, on the other hand, extends very naturally to such programs: indeed, our complexity bounds still hold in this context, and the encodings used in Lemma 27 and Theorem 28 can seamlessly be used to encode uniform families of algebraic formulae, or algebraic circuits, of polynomial depth, provided the base type for numbers (be they integers or floating numbers, or even real or complex numbers) and for the algebraic basic operations are given in the typing context.

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 approach used by Leivant and Marion [16] with their safe recursion with substitutions: using sequential resource bounds from Ptime Safe Recursion [3], 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 [12, 4, 11]. 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 [7], in order to propose a sound and complete type system for PSPACE, with a truly parallel evaluation strategy.

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

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

1 Parallel Soft Linear Logic

1.1 Soft Linear Logic

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

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

$$\frac{\Gamma \vdash U \quad V, \Delta \vdash Z}{\Gamma, U \multimap V, \Delta \vdash Z}\,(\multimap L) \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B}\,(\& R) \qquad \frac{\Gamma, A \vdash V}{\Gamma, A \& B \vdash V}\,(\& L_1)$$

$$\frac{\Gamma, B \vdash V}{\Gamma, A \& B \vdash V}\,(\& L_2) \qquad \frac{\Gamma \vdash U}{\Gamma \vdash \forall \alpha U}\,(\forall R) \qquad \frac{\Gamma, U[V/\alpha] \vdash Z}{\Gamma, \forall \alpha U \vdash Z}\,(\forall L)$$

$$\frac{\Gamma \vdash U}{!\Gamma \vdash !U}\,(sp) \qquad \frac{\Gamma, (U)^n \vdash V \quad n \geq 0}{\Gamma, !U \vdash V}\,(m), \text{ of rank } n$$

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

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

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

1.2 Parallel Modalities

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

Polarities

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

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

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

Rules for Parallel Modalities

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

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

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

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

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

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

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

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. The non-linear distribution of parallel modalities in PSLL breaks this isomorphism: permuting two binary ($/\!\!/cut$)-rules may come in conflict with the side condition $/\!\!/\Delta_i \subsetneq /\!\!/\Delta$. A similar remark can be made for $\multimap L$ rules as well. Since we want a *uniform* bound on the parallel normalization of the premises, we define a n-ary parallel (cut)-rule, exemplified in Example 5, as a parallel extension of the linear one, where the side condition for $/\!\!/$ modalities is adapted accordingly.

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

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

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

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

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 $T_t \to T_w$ (respectively $T_w \to T_t$) of R_2, with corresponding principal formula F belonging to one of the premises T_v of R_1, the pair $T_t \to T_w$ (resp. $T_w \to T_t$) is replaced by $T_v \to T_w$ (resp. $T_w \to T_v$), with the same corresponding principal formula.

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

▶ **Definition 3** (n-ary ($/\!\!/, \multimap cut$) rule)**.** *A n-ary ($/\!\!/, \multimap cut$) rule is*

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

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

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

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

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

▶ **Lemma 4.** *The cut-pairing relation on the premises of a* $(\mathbin{/\!/}, cut/\multimap L)$ *rule* R *defines a forest structure* $F(R)$*, called the* pairing forest *of the* $(\mathbin{/\!/}, cut/\multimap L)$*, on the premises of* R*; the edges of the pairing forest are the cut-pairs of the rule.*

▶ **Example 5.** A tree of linear $(\multimap L)$ and (cut) rules is

$$\dfrac{\Gamma_1 \vdash A_1 \qquad \dfrac{\dfrac{\Gamma_2 \vdash A_2 \qquad \Gamma_3, A_2 \vdash U}{\Gamma_2, \Gamma_3 \vdash U}(cut) \qquad \dfrac{\Gamma_4 \vdash A_3 \qquad \Gamma_5, A_1, A_3, V \vdash W}{\Gamma_4, \Gamma_5, A_1, V \vdash W}(cut)}{\Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V, A_1 \vdash W}(\multimap L)}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}(cut)$$

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

$$\dfrac{\Gamma_1 \vdash A_1 \qquad \Gamma_2 \vdash A_2 \qquad \Gamma_3, A_2 \vdash U \qquad \Gamma_4 \vdash A_3 \qquad \Gamma_5, A_1, A_3, V \vdash W}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}(cut, \multimap L)$$

A corresponding 5-ary $(\mathbin{/\!/}, \multimap cut)$ rule, with $\mathbin{/\!/}$-formulae satisfying the side condition, is

$$\dfrac{\mathbin{/\!/}F, \Gamma_1 \vdash A_1 \qquad \mathbin{/\!/}G, \Gamma_2 \vdash A_2 \qquad \Gamma_3, A_2 \vdash U \qquad \mathbin{/\!/}G, \Gamma_4 \vdash A_3 \qquad \mathbin{/\!/}F, \Gamma_5, A_1, A_3, V \vdash W}{\mathbin{/\!/}F, \mathbin{/\!/}G, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, U \multimap V \vdash W}$$

The cut-pairs are

- $(\mathbin{/\!/}F, \Gamma_1 \vdash A_1) \to (\mathbin{/\!/}F, \Gamma_5, A_1, A_3, V \vdash W)$ with principal formula A_1,
- $(\mathbin{/\!/}G, \Gamma_2 \vdash A_2) \to (\Gamma_3, A_2 \vdash U)$ with principal formula A_2, and
- $(\mathbin{/\!/}G, \Gamma_4 \vdash A_3) \to (\mathbin{/\!/}F, \Gamma_5, A_1, A_3, V \vdash W)$ with principal formula A_3,

which defines the pairing forest, with two roots $(\mathbin{/\!/}F, \Gamma_5, A_1, A_3, V \vdash W)$ and $(\Gamma_3, A_2 \vdash U)$. The $\multimap$-pair is $(\Gamma_3, A_2 \vdash U) \to (\mathbin{/\!/}F, \Gamma_5, A_1, A_3, V \vdash W)$ with principal formula $U \multimap V$.

We now define PSLL by the rules $(\mathbin{/\!/}ax)$, $(\multimap R)$, $(\forall R)$, $(\forall L)$, $(\& R)$, $(\& L_i)$, (m) $(\mathbin{/\!/}sp)$, $(\mathbin{/\!/}W)$, $(\mathbin{/\!/}D)$ and $(\mathbin{/\!/}, \multimap cut)$.

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

1.3 Parallel Cut Elimination

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

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

▶ **Lemma 7.**

1. *A* $(\mathbin{/\!/}, \multimap cut)$ *rule* (R_1)*, with premise* $\Gamma \vdash V$ *commutes with any non* $(\mathbin{/\!/}, \multimap cut)$*, non* $(\mathbin{/\!/}W)$*, non* $(\mathbin{/\!/}D)$*, non* $(\mathbin{/\!/}sp)$ *rule* (R_2) *with conclusion* $\Gamma \vdash V$*, provided the principal formula of* (R_2) *is not a sub-formula of any principal formula of* (R_1) *with respect to the premise* $\Gamma \vdash V$*.*
2. *A* $(\mathbin{/\!/}W)$ *or a* $(\mathbin{/\!/}D)$ *rule* (R_1)*, with premise* $\Gamma \vdash V$ *commutes with any non* $(\mathbin{/\!/}ax)$ *rule* (R_2) *with conclusion* $\Gamma \vdash V$*, provided the principal formula of* (R_2) *is not a subformula of the principal formula of* (R_1)*.*

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

▶ **Proposition 8.** *PSLL enjoys cut elimination.*

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

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

$$\dfrac{\dfrac{\sslash\Delta_1,\Gamma,B\vdash C}{\sslash\Delta_1,\Gamma\vdash B\multimap C}\,(\multimap R)\qquad \dfrac{\sslash\Delta_3,\Phi\vdash B\qquad \sslash\Delta_4,\Lambda,C\vdash V}{\sslash\Delta_2,\Phi,\Lambda,B\multimap C\vdash V}\,(\sslash\multimap L)}{\sslash\Delta,\Gamma,\Phi,\Lambda\vdash V}\,(\sslash,\multimap cut)$$

reduces to $\dfrac{\sslash\Delta_1,\Gamma,B\vdash C\qquad \sslash\Delta_3,\Phi\vdash B\qquad \sslash\Delta_4,\Lambda,C\vdash V}{\sslash\Delta,\Gamma,\Phi,\Lambda\vdash V}\,(\sslash,\multimap cut)$.

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

Rule $(\sslash ax)$

$$\dfrac{\dfrac{}{\sslash\Delta_1,B\vdash B}\,(\sslash ax)\qquad \sslash\Delta_2,\Gamma,B\vdash V}{\sslash\Delta,\Gamma,B\vdash V}\,(\sslash,\multimap cut)$$

when no other premise exists, reduces to $\dfrac{\sslash\Delta_2,\Gamma,B\vdash V}{\sslash\Delta,\Gamma,B\vdash V}\,(\sslash W^*)$, , Where $(\sslash W)^*$ stands for several applications of the $(\sslash W)$ rule.

Similarly,

$$\dfrac{\Pi_1\cdots\qquad \dfrac{}{\sslash\Delta_1,B\vdash B}\,(\sslash ax)\qquad \Pi_t\cdots\qquad \sslash\,\Delta_2,\Gamma,B\vdash V\qquad \cdots\Pi_n}{\sslash\Delta,\Gamma,B\vdash V}\,(\sslash,\multimap cut)$$

reduces to $\dfrac{\Pi_1\cdots\Pi_t\cdots\qquad \sslash\Delta_2,\Gamma,B\vdash V\qquad \cdots\Pi_n}{\sslash\Delta,\Gamma,B\vdash V}\,(\sslash,\multimap cut)$.

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

Rules $(\sslash sp)$, (m)

$$\dfrac{S_1,\cdots,S_k\qquad \dfrac{\sslash\Delta_1,\Gamma\vdash B}{\sslash\Delta_1,!\Gamma\vdash !B}\,(\sslash sp)\qquad \dfrac{\sslash\Delta_2,\Lambda,B^n\vdash V}{\sslash\Delta_2,\Lambda,!B\vdash V}\,(m)}{\sslash\Delta,!\Gamma,\Lambda\vdash V}\,(\sslash,\multimap cut)$$

reduces to
$$\dfrac{\dfrac{S_1^n,\cdots,S_k^n\qquad \sslash\,\Delta_1,\Gamma\vdash B\cdots\qquad \sslash\Delta_1,\Gamma\vdash B\qquad \sslash\Delta_2,\Lambda,B^n\vdash V}{\sslash\Delta,\Gamma^n,\Lambda,\vdash V}\,(\sslash,\multimap cut)}{\sslash\Delta,!\Gamma,\Lambda\vdash V}\,(m)^*$$

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

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

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

reduces to $\dfrac{/\!\!/\Delta_1, \Gamma \vdash U[V/\alpha] \qquad /\!\!/\Delta_2, \Lambda, U[V/\alpha] \vdash V}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}(/\!\!/, \multimap cut)$.

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

$$\dfrac{\dfrac{/\!\!/\Delta_1, \Gamma \vdash A \qquad /\!\!/\Delta_1 \Gamma \vdash B}{/\!\!/\Delta_1, \Gamma \vdash A\& B}(\& R) \quad \dfrac{/\!\!/\Delta_2, \Lambda, A \vdash V}{/\!\!/\Delta_2, \Lambda, A\& B \vdash V}(\& L_1)}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}(/\!\!/, \multimap cut)$$

reduces to $\dfrac{/\!\!/\Delta_1, \Gamma \vdash A \qquad /\!\!/\Delta_2, \Lambda, A \vdash V}{/\!\!/\Delta, \Gamma, \Lambda \vdash V}(/\!\!/, \multimap cut)$, and, of course, the cut elimination rule for $(\& L_2)$, $(\& R)$ follows a similar pattern.

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

2 Complexity Bounds

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

▶ **Definition 9.** *Let Π be a PSLL proof, with conclusion sequent $S = \Gamma \vdash V$. We define*

- *The size $|S|$ of S is the number of connectives in S.*
- *The size $|\Pi|$ of Π is the number of nodes in the proof-tree.*
- *The depth of a node R in Π is the length of the path from S to R in Π; the depth $d(\Pi)$ of Π is the maximal depth of its nodes.*
- *The rank $r(\Pi)$ of Π is the maximal rank of its (m) rules.*
- *The degree $d(f)$ of a formula f is the maximal nesting of its $!$ modalities. The degree $d(S)$ of a sequent S is the maximal degree of its formulas. The degree $d(\Pi)$ of a proof is the maximal degree of its sequents.*

PSLL proofs have bounded depth, and bounded number of (cut)-rules.

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

▶ **Lemma 11.** *Let Π be a PSLL proof, of rank n, with conclusion sequent S of degree d. Then, on any path from S to an axiom in Π, there are at most $O(|S|.n^d)$ $(/\!\!/, \multimap cut)$-rules with cut-pairs of premises.*

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

▶ **Lemma 12.** *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-formulas. Then, the depth of Π is $O(M.|S|.n^{2d})$.*

▶ **Lemma 13.** *Let R be a $(/\!\!/, \multimap cut)$ rule with cut-pairs $(S_1 = \Gamma_1 \vdash A_1) \to S, \cdots, (S_t = \Gamma_t \vdash A_t) \to S$ and cut-formulae $A_1, \cdots, A_t$. Assume moreover that*

- *each of the proof trees with conclusion S_i, for $i = 1, \cdots, t$, ends with the PSLL rule with right principal formula A_i, and*
- *the proof tree with conclusion S ends with the t PSLL rules with left principal formula A_i, for $i = 1, \cdots, t$.*

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

Proof. The cut-elimination steps of Proposition 8 act on distinct left sub-formulae of S, and distinct premises (other than S) of the $(/\!\!/, \multimap cut)$ rule R. ◀

▶ **Definition 14** (Parallel elimination of an innermost cut). *Le Π be a PSLL proof. A $(/\!\!/, \multimap cut)$ rule R with cut-pairs is innermost in Π if there is no other $(/\!\!/, \multimap cut)$ rule with cut-pairs along any path from R to the axioms of Π.*

Let R be an innermost $(/\!\!/, \multimap cut)$ 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 of Lemma 7 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 of Lemma 7 such that, 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 Lemma 13 for 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.*

▶ **Definition 15** (Innermost parallel cut-elimination). *Let Π be a PSLL proof. The* Innermost parallel cut-elimination *procedure consists in applying in parallel, for all its innermost cuts, their parallel elimination, until no $(/\!\!/, \multimap cut)$ rule with cut-pairs remains.*

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

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

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

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

We now have a parallel, polynomial time cut-elimination procedure:

▶ **Theorem 18.** *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.*

Proof. By Lemma 12, the depth of the proof-tree is at most $O(M.|S|.n^{2d})$: this bounds applies therefore for the overall parallel time needed to parse the proof-tree and reach all innermost $(\sslash, \multimap cut)$-rules. These innermost $(\sslash, \multimap cut)$ rules belong to different branches of the proof tree: they can therefore be eliminated safely in parallel. Each of these parallel elimination steps takes at most $O(M.h)$ steps.

By Lemma 11, the number of $(\sslash, \multimap cut)$-rules with cut-pairs on any path in the proof tree is bounded by $O(|S|.n^d)$: this bounds the number of times one needs to fully eliminate the innermost $(\sslash, \multimap cut)$-rules, hence the overall bound. ◀

Note that, in Theorem 18, we have only counted the number of parallel cut-elimination steps. Lemma 16 ensures moreover that, for each of these cut-elimination steps, the number of rule permutations needed to compute it is also polynomially bounded.

▶ **Example 19.** Let us consider the following derivation proof, corresponding to the application of a function to two arguments, of types A and B respectively, in a curryfied fashion, with atomic resulting type C. Since the strategy is innermost, the four premises in the tree are conclusions of (cut)-free derivation trees.

$$\dfrac{\dfrac{\dfrac{\sslash\Delta_1,\Gamma,A,B\vdash C}{\sslash\Delta_1,\Gamma,A\vdash B\multimap C}\,(\multimap R)}{\sslash\Delta_1,\Gamma\vdash A\multimap B\multimap C}\,(\multimap R) \qquad \dfrac{\sslash\Delta_3,\Phi\vdash A \qquad \dfrac{\sslash\Delta_5,\Lambda\vdash B \qquad \sslash\Delta_6,\Theta,C\vdash C}{\sslash\Delta_4,\Lambda,\Theta,B\multimap C\vdash C}\,(\sslash\multimap L)}{\sslash\Delta_2,\Phi,\Lambda,\Theta,A\multimap B\multimap C\vdash C}\,(\sslash\multimap L)}{\sslash\Delta,\Gamma,\Phi,\Lambda,\Theta\vdash C}\,(\sslash cut)$$

One parallel $(\sslash cut)$ elimination step exhibits the application of the first argument, of type A,

$$\dfrac{\dfrac{\sslash\Delta_1,\Gamma,A,B\vdash C}{\sslash\Delta_1,\Gamma,A\vdash B\multimap C}\,(\multimap R) \qquad \sslash\Delta_3,\Phi\vdash A \qquad \dfrac{\sslash\Delta_5,\Lambda\vdash B \qquad \sslash\Delta_6,\Theta,C\vdash C}{\sslash\Delta_4,\Lambda,\Theta,B\multimap C\vdash C}\,(\sslash\multimap L)}{\sslash\Delta,\Gamma,\Phi,\Lambda,\Theta\vdash C}\,(\sslash cut)$$

And a second one exhibits the application of the second argument, of type B.

$$\dfrac{\sslash\Delta_1,\Gamma,A,B\vdash C \qquad \sslash\Delta_3,\Phi\vdash A \qquad \sslash\Delta_5,\Lambda\vdash B \qquad \sslash\,\Delta_6,\Theta,C\vdash C}{\sslash\Delta,\Gamma,\Phi,\Lambda,\Theta\vdash C}\,(\sslash cut)$$

The premise $\sslash\Delta_6,\Theta,C\vdash C$ is the root of the pairing forest, and the atomic type C is eliminated first.

$$\dfrac{\sslash\Delta_1,\Gamma,A,B\vdash C \qquad \sslash\Delta_3,\Phi\vdash A \qquad \sslash\Delta_5,\Lambda\vdash B}{\sslash\Delta,\Gamma,\Phi,\Lambda,\Theta\vdash C}\,(\sslash cut)$$

Finally, the two arguments types A and B are then eliminated in parallel, with elimination steps corresponding to the substitution of the corresponding values in the function term in the Curry-Howard isomorphism, as detailed in the next section.

3 A Parallel Polynomial Time Type Assignment for λ-calculus

3.1 Parallel Soft Types

We take insipiration from the STA type assignment of Gaboardi and Ronchi [7]. We add the parallel modalities in a restricted way, as follows.

▶ **Definition 20** (Parallel Soft types (PSTA)). *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. PSTA types are given by the following grammar:*

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

A PSTA Typing context is a set of type assignments $x:\sigma$, where x is a variable 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.

3.2 Typing Rules

Our PSTA typing rules are the following.

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

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

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

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

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

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

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

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

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.

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

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

▶ **Proposition 21.** *Let Π be a typing derivation with conclusion $\Gamma \vdash M : !\sigma$. Then, the context Γ is $/\!\!/\Delta,\ !\Lambda$.*

A corollary of Proposition 21 is

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

From the absence of sharing, we derive

▶ **Proposition 23.** *PSTA enjoys the subject reduction property: if $\Gamma \vdash M : \sigma$ and $M \to_\beta M'$, then $\Gamma \vdash M'$.*

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

1. $\sigma = \tau \to A$. The derivation is

$$\dfrac{\dfrac{/\!\!/\Delta_1,\ \Gamma_1,\ y:\ \tau \vdash P:\ A}{/\!\!/\Delta_1,\ \Gamma_1 \vdash \lambda y.P:\ \tau \to A} \qquad \dfrac{/\!\!/\Delta_3,\ \Gamma_2,\ \vdash Q:\ \tau \qquad /\!\!/\Delta_4,\ \Gamma_3,\ z:\ A \vdash N:\ \sigma}{/\!\!/\Delta_2,\ \Gamma_2,\ \Gamma_3,\ x:\ \tau \to A \vdash N[(x\ Q)/z]:\ \sigma}\,(/\!\!/ \multimap L)}{/\!\!/\Delta,\ \Gamma_1,\ \Gamma_2,\ \Gamma_3 \vdash N[(x\ Q)/z][\lambda y.P/x]:\ \sigma}\,(/\!\!/cut)$$

   Cut elimination yields then the following derivation tree

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

   which proves the subject reduction property. If $\sigma = \forall\alpha\tau$ or $\sigma = \tau\&\tau'$, the cut-elimination steps eventually reduce to the case above.
2. $\sigma = !\tau$. By Proposition 21, and modulo rule permutations the derivation is

$$\dfrac{\dfrac{/\!\!/\Delta_1,\ \Gamma_1 \vdash \lambda y.P:\ \tau}{/\!\!/\Delta_1,\ !\Gamma_1 \vdash \lambda y.P : !\tau}\,(/\!\!/Sp) \qquad \dfrac{/\!\!/\Delta_2,\ \Gamma_2,\ \cdots x_i:\ \tau \cdots \vdash N[\cdots (x_i\ Q)/z_i \cdots]:\ \sigma}{/\!\!/\Delta_2,\ \Gamma_2,\ x : !\tau \vdash N[(x\ Q)/z_1, \cdots, (x\ Q)/z_n]:\ \sigma}\,(m)}{/\!\!/\Delta,\ !\Gamma_{1,;}\, \Gamma_2 \vdash N[(x\ Q)/z_1, \cdots, (x\ Q)/z_n][\lambda y.P/x]:\ \sigma}\,(/\!\!/cut)$$

   Cut elimination yields then the following derivation tree

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

   and the induction hypothesis applies to the n cut-types τ. ◀

3.3 A Parallel, Polynomial Time Evaluation Strategy

▶ **Theorem 24.** *Let T be a λ-term, typable in PSTA.Then, T normalizes in polynomial parallel time.*

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

Let us first introduce some definitions and observations.

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

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

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

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

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

Then, the following observations hold.

1. In each typing judgment in Π (and therefore in $D(\Pi)$), the typing context contains type assignments for variable terms only.
2. Erasing the variable names in the contexts of $D(\Pi)$ (while keeping the corresponding types) yields a PSLL proof $L(\Pi)$, with types as formulae,
3. All right-hand side λ-terms in Π are uniquely determined by $D(\Pi)$, and finally,
4. The variable type assignments in $D(\Pi)$ are preserved by the subject-reduction property: If T_1 is a λ-term with PSTA type derivation Π_1 , $T_1 \to_\beta T_2$, and Π_2 is the type derivation of T_2 obtained by the subject reduction steps of Proposition 23, then the variable type assignments in $D(\Pi_1)$ and $D(\Pi_2)$ coincide.

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

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

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

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

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

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

- if p is conclusion of a $(\S Sp)$, $(\forall R)$, $(\forall L)$, $(\S W)$ or $(\& R)$ rule, with premise p', then $v(p)$ is $v(p')$.
- if p is conclusion of a (m), $(\S D)$ or $(\& L_i)$ rule with premise p', two cases arise:
  1. $v(p')$ is labelled x_i: then, (x/x_i) belongs to $s(p)$. In that case $v(p)$ is labelled x, and its successors are those of $v(p')$.
  2. otherwise, $v(p)$ is $v(p')$.
- if p is conclusion of a $(\multimap R)$ rule with premise p', $v(p)$ is an inner node labelled λx, with left successor node p'.
- if p is conclusion of a $(\S, \multimap\, cut)$-rule R: let p' be its rightmost premise. Then, three cases arise:
  1. $v(p')$ is labelled x, (p'', x) belongs to $s(p')$: then, $v(p)$ is obtained from $v(p'')$ by adding to its root the successor vertices of $v(p')$.
  2. $v(p')$ is labelled x, (yp'', x) belongs to $s(p')$: then, $v(p)$ is a vertex labelled y, with right successor $v(p'')$.
  3. otherwise, $v(p)$ is $v(p')$.

Performing the procedure above in parallel for all paths in $D(\Pi)$ provides then a succinct description of $B(T)$ in parallel polynomial time. ◀

4 Completeness of PSTA

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

First, PSTA captures (obviously) STA.

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

▶ **Corollary 26.** *Let Π be a SLL proof of degree d and rank n, with conclusion $\Gamma \vdash A$ of size s. Then, there exists a PSLL proof Π' with conclusion $\Delta, \Gamma \vdash A$, of degree d and rank n.*

Proof. Take $\Delta = !^d \S A_1, \ldots, !^d \S A_k$, with $k = W_\Pi(1) + s$, for any $A_1, \cdots, A_k$. ◀

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

The following lemma allows us to encode some sort of polynomial recursion with substitutions a la Leivant and Marion [16] in PSTA.

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

- $\mathtt{op}$ *with derivation* $\Pi_{\mathtt{op}}$ *with conclusion* $\Gamma_{\mathtt{op}} \vdash \mathtt{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L}$.
- $\mathtt{s}_1$ *and* $\mathtt{s}_2$ *with derivation* Π_i, *for* $i \in \{1, 2\}$, *with conclusion* $\Gamma_i \vdash \mathtt{s}_i : \mathbf{L} \multimap \mathbf{L}$.

- *and, for any univariate polynomial P of degree d, a function $\overline{P}$, encoding the church integer $P(|L|)$ for a binary list L, with derivation $\Pi_{\overline{P}}$ with conclusion $\Gamma_{\overline{P}} \vdash \overline{P} :!^d\mathbf{L} \multimap \mathbf{N}$.*

We now consider the following recursive function with substitutions, on binary lists: $f(v) = \mathtt{op}(v, f(\mathtt{s}_1(v)), f(\mathtt{s}_2(v)))$. Moreover, we assume that on any input v, the recursive computation of f reaches a fixed point after $P(|v|)$ steps. Then, $f(v)$ is PSTA definable with degree d.

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

Indeed, $\Pi_{\mathtt{Step}}$ is

$$\dfrac{\dfrac{A_{\mathtt{op}} \quad A_1 \quad A_2 \quad A_{\mathtt{s}}1 \quad A_{\mathtt{s}}2 \quad A_{\mathtt{V}}1 \quad A_{\mathtt{V}}2 \quad A_{\mathtt{step}}}{\Gamma, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \mathtt{op}\ v\ (h\ (\mathtt{s}_1\ v))\ (h\ (\mathtt{s}_2\ v)) : \mathbf{L}}\,(\sslash cut)}{\Gamma \vdash \mathtt{Step} : \mathbf{L2}}\,(\multimap R)^2$$

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

$A_{\mathtt{s}}i$ is
$$\dfrac{\dfrac{\dfrac{}{v : \mathbf{L} \vdash v : \mathbf{L}}\,(\sslash Id) \qquad \dfrac{}{x : \mathbf{L} \vdash x : \mathbf{L}}\,(\sslash Id)}{X : \sslash A, \mathtt{t}_i : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash \mathtt{t}_i\ v : \mathbf{L}}\,(\sslash \multimap L)}{X : \sslash A, \mathtt{t}_i : \mathbf{L} \multimap \mathbf{L}, v : \sslash\mathbf{L} \vdash \mathtt{t}_i\ v : \mathbf{L}}\,(\sslash D),$$

$A_{\mathtt{V}}1$ is
$$\dfrac{\dfrac{\dfrac{}{v : \mathbf{L} \vdash v : \mathbf{L}}\,(\sslash Id) \qquad \dfrac{}{x : \mathbf{L} \vdash x : \mathbf{L}}\,(\sslash Id)}{X : \sslash A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h\ v : \mathbf{L}}\,(\sslash \multimap L)}{X : \sslash A, h : \sslash\mathbf{L} \multimap \mathbf{L}, v : \sslash\mathbf{L} \vdash h\ v : \mathbf{L}}\,(\sslash D)^2,$$

$A_{\mathtt{V}}2$ is
$$\dfrac{\dfrac{}{v : \mathbf{L} \vdash v : \mathbf{L}}\,(\sslash Id) \qquad \dfrac{}{x : \mathbf{L} \vdash x : \mathbf{L}}\,(\sslash Id)}{X : \sslash A, h : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash h\ v : \mathbf{L}}\,(\sslash \multimap L),$$

and $A_{\mathtt{step}}$ is

$$\dfrac{\dfrac{}{V_1 : \mathbf{L} \vdash V_1 : \mathbf{L}}\,(\sslash Id) \qquad \dfrac{}{x_1 : \mathbf{L} \vdash x_1 : \mathbf{L}}\,(\sslash Id) \cdots \qquad \dfrac{}{x_3 : \mathbf{L} \vdash x_3 : \mathbf{L}}\,(\sslash Id)}{X : \sslash A, V_1 : \mathbf{L}, V_2 : \mathbf{L}, V_3 : \mathbf{L}, \mathtt{op} : \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \multimap \mathbf{L} \vdash \mathtt{op}\ V_1\ V_2\ V_3 : \mathbf{L}}\,(\sslash, \multimap cut)$$

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

$\Pi_{\overline{P}\ v}$:
$$\dfrac{\dfrac{\Pi_{\overline{P}}}{\Gamma_{\overline{P}} \vdash \overline{P} :!^d\mathbf{L} \multimap \mathbf{N}} \qquad \dfrac{\dfrac{}{v :!^d\mathbf{L} \vdash v :!^d\mathbf{L}}\,(\sslash Id) \qquad \dfrac{}{x :!^d\mathbf{L} \vdash x :!^d\mathbf{L}}\,(\sslash Id)}{x : \sslash A, \overline{P} :!^d\mathbf{L} \multimap \mathbf{N}, v :!^d\mathbf{L} \vdash \overline{P}\ v : \mathbf{N}}\,(\sslash \multimap L)}{(x : \sslash A)^2, \Gamma_{\overline{P}}, v :!^d\mathbf{L} \vdash \overline{P}\ v : \mathbf{N}}\,(\sslash cut)$$

Π_s:

$$\dfrac{\Pi_{\overline{P}\ v} \qquad !\Pi_{\mathtt{Step}} \qquad \dfrac{\dfrac{!\mathtt{Step} :!\mathbf{L2} \vdash !\mathtt{Step} :!\mathbf{L2} \qquad x :!\mathbf{L2} \vdash x :!\mathbf{L2}}{X : \sslash A, (\overline{P}\ v) : \mathbf{L3}, !\mathtt{Step} :!\mathbf{L2} \vdash \overline{P}\ v\ \mathtt{Step} : \mathbf{L2}}\,(\sslash \multimap L)}{X : \sslash A, (\overline{P}\ v) : \mathbf{N}, !\mathtt{Step} :!\mathbf{L2} \vdash \overline{P}\ v\ \mathtt{Step} : \mathbf{L2}}\,(\forall L)}{\Gamma_{\mathtt{op}}, \Gamma_1, \Gamma_2, (X : \sslash A)^3, h : \sslash(\mathbf{L} \multimap \mathbf{L}), v : \sslash \mathbf{L}, \Gamma_{\overline{P}}, v :!^d\mathbf{L} \vdash \overline{P}\ v\ \mathtt{Step} : \mathbf{L2}}\,(\sslash cut)$$

Π_{sl}:

$$\dfrac{\Pi_s \qquad \dfrac{\lambda y.y : \mathbf{L} \multimap \mathbf{L} \vdash \lambda y.y : \mathbf{L} \multimap \mathbf{L} \qquad f : \mathbf{L} \multimap \mathbf{L} \vdash f : \mathbf{L} \multimap \mathbf{L}}{X : \sslash A, \overline{P}\ v\ \mathtt{Step} : \mathbf{L2} \vdash \overline{P}\ v\ \mathtt{Step}\ \lambda y.y : \mathbf{L} \multimap \mathbf{L}}\,(\sslash \multimap L)}{\Gamma_{\mathtt{op}}, \Gamma_1, \Gamma_2, (X : \sslash A)^4, h : \sslash(\mathbf{L} \multimap \mathbf{L}), v : \sslash \mathbf{L}, \Gamma_{\overline{P}}, v :!^d\mathbf{L} \vdash \overline{P}\ v\ \mathtt{Step}\ \lambda y.y : \mathbf{L} \multimap \mathbf{L}}\,(\sslash cut)$$

and finally $\Pi_{\mathtt{Value}}$:

$$\dfrac{\dfrac{\Pi_{sl} \qquad \dfrac{v : \mathbf{L} \vdash v : \mathbf{L} \qquad z : \mathbf{L} \vdash z : \mathbf{L}}{X : \sslash A, \overline{P}\ v\ \mathtt{Step}\ \lambda y.y : \mathbf{L} \multimap \mathbf{L}, v : \mathbf{L} \vdash (\overline{P}\ v\ \mathtt{Step}\ \lambda y.y)v : \mathbf{L}}\,(\sslash \multimap L)}{\Gamma', v : \mathbf{L} \vdash (\overline{P}\ v\ \mathtt{Step}\ \lambda y.y)v : \mathbf{L}}\,(\sslash cut)}{\Gamma' \vdash \mathtt{Value} : \mathbf{L} \multimap \mathbf{L}}\,(\multimap R)$$

◀

▶ **Theorem 28.** *PSTA is complete for FPAR.*

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

- On any input $x = x_1, \cdots, x_n$ of size n, $g(x_1, \cdots, x_n)$ is computed by a boolean circuit C_n of depth $p(n)$, with output node t.
- 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 length less than $p(n)$. Each node will be identified by these paths (there may be several paths for a given node).
- $\mathtt{and}(x, y)$ (respectively $\mathtt{or}(x, y)$, resp. $\mathtt{not}(x, y)$) is `true` if the path y encodes a `and` (resp. `or`, resp. `not`) node of $C_{|x|}$, and `false` otherwise.
- $\mathtt{input}(x, y)$ is $(x_i, \mathtt{true})$ if y encodes the i^{th} input node of $C_{|x|}$, and $(0, \mathtt{false})$ otherwise.
- $\mathtt{s}_1(y) = 0.y$ encodes a path to the left parent of the node encoded by y, if it exists.
- $\mathtt{s}_2(y) = 1.y$ encodes a path to the right parent of the node encoded by y, if it exists.

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

5 Concluding Remarks

In this paper we have only investigated one of the many possible choices for the way the parallel $(\sslash, \multimap cut)$ rule allows contraction on $\sslash$ 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 side condition of the $(\sslash, \multimap cut)$-rule to capture other complexity classes? Such obvious candidates are the classes NC^i, which we could hope to capture by taking a fully linear ($/\!\!/, \multimap$ *cut*)-rule (for ensuring sequential polynomial time), with an additional side condition ensuring parallel polylogarithmic time cut-elimination. Is it also possible to use this approach on type systems capturing other sequential complexity classes, for instance Logspace [17, 15], and to obtain other interesting results?

References

1 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. doi:10.1007/978-3-540-24727-2_4.

2 Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41–62, January 2009. doi:10.1016/j.ic.2008.08.005.

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

4 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. doi:10.1016/j.ic.2015.12.009.

5 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. doi:10.1145/1328438.1328456.

6 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. An implicit characterization of PSPACE. *ACM Transactions on Computational Logic*, 13(2):1–36, April 2012. doi:10.1145/2159531.2159540.

7 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. doi:10.1007/978-3-540-74915-8_21.

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

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

10 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.

11 Paulin Jacobé De Naurois. Pointers in recursion: Exploring the tropics. In *FSCD*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2019. doi:10.4230/LIPICS.FSCD.2019.29.

12 Satoru Kuroda. Recursion schemata for slowly growing depth circuit classes. *Computational Complexity*, 13(1-2):69–89, 2004. doi:10.1007/s00037-004-0184-4.

13 Yves Lafont. Soft linear logic and polynomial time. *Theor. Comput. Sci.*, 318(1-2):163–180, 2004. doi:10.1016/j.tcs.2003.10.018.

14 Ugo Dal Lago and Martin Hofmann. Bounded linear logic, revisited. *Logical Methods in Computer Science*, 6(4), December 2010. doi:10.2168/lmcs-6(4:7)2010.

15 Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *Programming Languages and Systems*, pages 205–225. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-11957-6_12.

16 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. doi:10.1007/BFb0022242.

17 Ulrich Schöpp. Stratified bounded affine logic for logarithmic space. In *LICS*, pages 411–420. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.45.