Clément Aubert
2 rue de l'Averse
94 000 Créteil, France
(+33/0)6.16.92.81.58
(+33/0)1.45.17.65.89
clement.aubert@lacl.fr
lacl.fr/~caubert/

M. Olivier Baudon
Labri
Université de Bordeaux
351, cours de la Liberation,
33 405 Talence

Le 5 décembre 2014

Sujet : Candidature à la qualification aux fonctions
de maître de conférences

M. Olivier Baudon,

vous avez été désigné pour être l'un des rapporteurs de mon dossier de candidature à la qualification aux fonctions de maître de conférences pour la section Informatique (27) du C.N.U. Je vous prie de bien vouloir trouver ci-inclut les pièces de ce dossier, et ci-dessous un récapitulatif de ces pièces.

Je ne vous ai pas joint mon mémoire de thèse de doctorat, mais vous pourrez facilement télécharger celui-ci à `https://lacl.fr/~caubert/recherche/these/these_c_aubert.pdf`. Vous retrouverez de façon générale sur ma page, à `https://lacl.fr/~caubert/`, des liens vers mes publications, des explicitations des acronymes et divers autres renseignements complémentaires (liste complète des exposés, transparents, etc).

Je me tiens bien évidemment à votre disposition pour tout renseignement complémentaire.

Bien cordialement,

Clément Aubert

Je vous prie de trouver ci-joint :

- *La déclaration de candidature à la qualification (p. 2-3),*
- *Un curriculum vitæ (p. 4-5),*
- *Mon diplôme de Doctorat de l'Université de Paris 13 (p. 6),*
- *Les deux rapports de pré-soutenance (p. 7-11),*
- *Le procès-verbal de ma soutenance de thèse (p. 12-14),*
- *Un descriptif de mes activités de recherche et d'enseignement (p. 15-18),*
- *Trois publications,*
  - Sublogarithmic uniform Boolean proof nets *(p. 19-31)*
  - Unification and Logarithmic Space *(en collaboration avec M. Bagnol, p. 32-47)*
  - Logic Programming and Logarithmic Space *(en collaboration avec M. Bagnol, P. Pistone et Th. Seiller, p. 48-66)*

Je me permets de plus de joindre à ce dossier des lettres de recommandation de

- *Laure Petrucci, Directrice du LIPN et ancienne Chef de département R&T (p. 67),*
- *Emmanuel Viennet, Chef de département R&T (p. 68),*
- *Camille Coti, Maître de Conférences à l'Université de Paris 13 (p. 69),*
- *Stefano Guerrini, Professeur à l'Université de Paris 13 (p. **??**).*

Il est conseillé de joindre ce document au dossier transmis
aux rapporteurs désignés par le Conseil National des Universités

DÉCLARATION DE CANDIDATURE À LA QUALIFICATION
AUX FONCTIONS DE MAÎTRE DE CONFÉRENCES,
POUR LA SECTION 27-Informatique
(Campagne 2015)
1ère demande

Je soussigné(e) M.
Nom de famille : AUBERT
Nom d'usage : AUBERT
Prénom : CLEMENT
Date et lieu de naissance : 14/09/1984 - REIMS
Nationalité : Française

Adresse postale et électronique à laquelle seront acheminées toutes les correspondances
2 RUE DE L'AVERSE

Code postal : 94000 Ville :CRETEIL Pays : FRANCE
Téléphone : 0148993139 Télécopie :
Adresse électronique : clement.aubert@lacl.fr

Date de création de la candidature
30/09/2014 à 11:09

Date de dernière modification de la candidature
30/09/2014 à 11:09

Titres universitaires français :
Doctorat

Diplôme au titre duquel la qualification est demandée : Doctorat
Titre : Linear Logic and Subpolynomial Classes of Complexity
Date de soutenance : 26/11/2013
Lieu de la soutenance : UNIVERSITE PARIS 13 (PARIS SORBONNE CITE)
Mention : Très honorable
Directeur : STEFANO GUERRINI
Composition du jury : PATRICK BAILLOT
ARNAUD DURAND
CLAUDIA FAGGIAN
UGO DAL LAGO
JEAN-YVES MARION
PAUL-ANDRE MELLIES

Virgile Mogbil

Liste des établs et labos d'exercice :

L.I.P.N. UMR CNRS 7030 - Institut Galilée - Université Paris 13

L.D.P. Institut de Mathématiques de Luminy (FRE 3529) - CNRS/Aix-Marseille Université

Université Paris-Est, LACL (EA 4219), UPEC, F-94010 Créteil, France

Activités en matière d'enseignement :

Moniteur IUT Réseaux & Télécommunications de Villetaneuse, étudiants de 1ère année, Base de données (26h), administration de système (52h), Programmation (52h), Projets tutorés (59h)

Thème de recherche et mots clés :

Théorie de la démonstration, Logique Linéaire, Complexité, Réseaux de Preuves, Automates, Complexité implicite, Géométrie de l'interaction, pointeurs, correspondance preuve-programme

Activités en matière d'administration et autres responsabilités collectives :

déclare faire acte de candidature à la qualification.

Fait à Créteil **le** 2/12/14

Signature

Clément Aubert

Curriculum vitæ

2 rue de l'Averse
94 000 Créteil, France
(+33/0)6.16.92.81.58
(+33/0)1.45.17.65.89
clement.aubert@lacl.fr
lacl.fr/~caubert/

Sujets de recherche

Théorie de la démonstration, Complexité implicite, Correspondance Preuves–Programmes, Logique Linéaire, Géométrie de l'interaction, Automates, Calcul de processus

Positions & parcours de formation

2014–présent **Post-doctorat**, *INRIA, SPADES – Université Paris-Est Créteil (UPEC, Paris 12), LACL — EA 4219*, sur contrat ANR REVER.

2013–2014 **Post-doctorat**, *CNRS – Aix-Marseille Université, I2M — UMR 7373 CNRS, équipe Logique de la Programmation (L.D.P.)*, sur contrat ANR Recré.

2010–2013 **Doctorat en Informatique (Mention très honorable)**, *ÉD Galilée (146) — Université Paris 13 — LIPN, UMR 7030, section 27*, Logique linéaire et classes de complexité sous-polynomiales.
Thèse soutenue le 26 novembre 2013 à l'Institut Galilée devant le jury composé de

M.	Patrick Baillot	C.N.R.S., E.N.S. Lyon	(Rapporteur)
M.	Arnaud Durand	Université Denis Diderot - Paris 7	(Président)
M.	Ugo Dal Lago	INRIA, Università degli Studi di Bologna	(Rapporteur)
Mme.	Claudia Faggian	CNRS, Université Paris Diderot - Paris 7	
M.	Stefano Guerrini	Institut Galilée - Université Paris 13	(Directeur)
M.	Jean-Yves Marion	LoRIA, Université de Lorraine	
M.	Paul-André Melliès	CNRS, Université Paris Diderot - Paris 7	
M.	Virgile Mogbil	Institut Galilée - Université Paris 13	(Co-encadrant)

2009–2010 **Master Recherche spécialité « Mathématiques » (Mention Bien)**, *Université Paris 7 — Denis Diderot*, Parcours « Logique mathématique et fondements de l'informatique » (LMFI).

2007–2010 **Licence puis Master 1 Recherche de Philosophie (Mention Assez Bien)**, *Université Paris 1 — Panthéon-Sorbonne*, « Logique, Philosophie, Histoire et Sociologie des Sciences ».

2005–2006 **Licence d'Histoire et Licence 2 de Philosophie**, *Université de Reims Champagne-Ardenne*.

2002–2005 **Khâgne (spécialité Histoire-Géographie) — Hypokhâgne — Bac S Européenne Allemand**, *Lycée Jean-Jaurès — Reims*.

Participation à des comités & activités de relecture

Membre du comité de programme de DICE 2015, *reviewer* pour l' *Italian Conference on Theoretical Computer Science* (ICTCS), le journal « Information & Computation » et FoSSaCS 2015.

Enseignements

Moniteur à l'I.U.T. de Villetaneuse (Paris 13), département Réseaux et Télécommunications (2010–2013), vacataires à l'UPEC (Paris 12), département Info. (2014)

54 h. I3 (semestre 1) **Algorithmique et programmation** — T.P. et T.D.

52 h. R3 (semestre 2) **Administration des systèmes d'exploitation réseaux** — T.P.

26 h. I4 (semestre 2) **Bases de données** — T.P. et T.D.

59 h. AA1, AA2 (semestres 1 et 2) **Apprendre autrement** — C.M., T.P. et T.D., responsable

35 h. L2 Info (semestre 3) **Initiation à l'algorithmique et à la complexité** — T.P. et T.D.

Publications

2014 **Clément Aubert** et Marc Bagnol. « Unification and Logarithmic Space ». In : *Rewriting and Typed Lambda Calculi*. Sous la dir. de Gilles Dowek. T. 8650. Lecture Notes in Computer Science. Springer, p. 77–92. ISBN : 978-3-319-08917-1. DOI : 10.1007/978-3-319-08918-8_6.

Clément Aubert, Marc Bagnol, Paolo Pistone et Thomas Seiller. « Logic Programming and Logarithmic Space ». In : *Asian Symposium on Programming Languages and Systems*. Sous la dir. de Jacques Garrigue. T. 8858. Lecture Notes in Computer Science. Springer, p. 39–57. DOI : 10.1007/978-3-319-12736-1_3.

Clément Aubert et Thomas Seiller. « Characterizing co-NL by a group action ». In : *Mathematical Structures in Computer Science* FirstView, p. 1–33. ISSN : 1469-8072. DOI : 10.1017/S0960129514000267.

2013 **Clément Aubert**. « Linear Logic and Sub-polynomial Classes of Complexity ». Thèse de doct. Université Paris 13–Sorbonne Paris Cité. Sous la dir. de Stefano Guerrini, Virgile Mogbil (UMR CNRS 7030 — Paris 13), 184 p.

Clément Aubert et Thomas Seiller. « Logarithmic Space and Permutations ». In : *Arxiv preprint* abs/1301.3189. arXiv : 1301.3189 [cs.LO]. Soumis à Information & Computation — Special Issue on Implicit Computational Complexity, 26 p.

2011 **Clément Aubert**. « Sublogarithmic uniform Boolean proof nets ». In : *Developments in Implicit Computational Complexity*. Sous la dir. de Jean-Yves Marion. T. 75. Electronic Proceedings in Theoretical Computer Science, p. 15–27. DOI : 10.4204/EPTCS.75.2.

2010 **Clément Aubert**. « Réseaux de preuves booléens sous-logarithmiques ». Mém.de mast. L.I.P.N. : L.M.F.I., Paris VII. Sous la dir. de Virgile Mogbil, Paulin Jacobé de Naurois (UMR CNRS 7030 — Paris 13), 29 p.

2009 **Clément Aubert**. « L'élimination des coupures dans la Logique des Domaines Constants ». Mém.de mast. Paris 1. Sous la dir. de Jean-Baptiste Joinet (UMR CNRS 7126 — Paris 7), 29 p.

Exposés de recherche (sélection)

2014 Séminaire du Laboratoire d'Algorithmique, Complexité et Logique, U-PEC
Séminaire de l'équipe L.I.M.D. — L.A.M.A., Université de Savoie
Séminaire de l'équipe L.D.P. — I.2.M., Aix-Marseille Université
Séminaire du département « Méthodes formelles » — Loria, Université de Lorraine

2013 *Workshop Logic and Computational Complexity* — Turin, CSL 2013
Séminaire CLI — Équipe de Logique Mathématique, Université Paris 7
Séminaire Jeunes Chercheurs — LIPN, Université Paris 13

2012 9e réunion de l'ANR-08-BLANC-0211-01 Complice — LIPN, Université Paris 13
Séminaire LDP et groupe de travail — IML, Aix-Marseille Université
FoCUS Meeting — Università Di Bologna
Séminaire LCR — LIPN, Université Paris 13
Logic and interactions 2012 — Centre International de Rencontres Mathématiques

2011 *Workshop Developments in Implicit Computational Complexity* — Saarebruck, ETAPS
Groupe de recherche pluridisciplinaire *Vérité et preuves* — Université Paris 1
16e rencontres LAC — PPS, Université Paris 7

Langues

Français — Langue maternelle, intérêt pour l'orthotypographie
Anglais — Parfaitement lu et compris, à l'aise à l'oral (TOIEC : 975)
Allemand — À réactiver (*Zertifikat Deutsch* du Goethe Institut obtenu en 2002)

R É P U B L I Q U E F R A N Ç A I S E

MINISTÈRE DE L'ÉDUCATION NATIONALE, DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE

UNIVERSITÉ PARIS 13 – SORBONNE PARIS CITÉ

DOCTORAT

Vu le code de l'éducation, notamment son article L. 612–7 ;

Vu le code de la recherche, notamment son article L. 412–1 ;

Vu le décret n° 2002–481 du 8 avril 2002 relatif aux grades et titres universitaires et aux diplômes nationaux ;

Vu l'arrêté du 3 septembre 1998 relatif à la charte des thèses ;

Vu l'arrêté du 7 août 2006 relatif à la formation doctorale ;

Vu l'avis conforme du ministère de l'enseignement supérieur et de la recherche ;

Vu les pièces justificatives produites par M. CLEMENT AUBERT, né le 14 septembre 1984 à REIMS (051), en vue de son inscription au doctorat ;

Vu le procès–verbal du jury attestant que l'intéressé a soutenu, le 26 novembre 2013 une thèse portant sur le sujet suivant : Logique linéaire et classes de complexité sous–polynomiales,
préparée au sein de l'école doctorale sciences, technologie, santé GALILEE, devant un jury présidé par ARNAUD DURAND, PROFESSEUR DES UNIVERSITES et composé de PATRICK BAILLOT, CHARGE(E) DE RECHERCHE, UGO DAL LAGO, CHARGE(E) DE RECHERCHE, CLAUDIA FAGGIAN, CHARGE(E) DE RECHERCHE, STEFANO GUERRINI, PROFESSEUR DES UNIVERSITES, JEAN–YVES MARION, PROFESSEUR DES UNIVERSITES, PAUL–ANDRE MELLIES, PROFESSEUR DES UNIVERSITES, VIRGILE MOGBIL, MAITRE DE CONFERENCES ;

Vu la délibération du jury ;

Le **DIPLÔME DE DOCTEUR** en INFORMATIQUE, *mention très honorable*

est délivré à **M. CLEMENT AUBERT**

et confère **le grade de docteur**,
pour en jouir avec les droits et prérogatives qui y sont attachés.

Fait à Créteil, le 4 juin 2014

Le titulaire

Le Président

Jean–Loup SALZMANN

Le Recteur d'Académie,
Chancelier des universités

Béatrice GILLE

6 / 69

N° PARXIII 10636863
/2014201205726

Rapport sur la thèse de Clément Aubert
Logique Linéaire et Classes de Complexité sous-polynomiales

Patrick Baillot

12 novembre 2013

– Spécialité : Informatique,
– Date de soutenance : mardi 26 novembre 2013
– Directeur : Stefano Guerrini. Co-encadrant : Virgile Mogbil.

La thèse de Clément Aubert se positionne à la croisée des chemins de la logique, des modèles de calcul et de la théorie de la complexité. Plus précisément elle s'inscrit dans un champ de recherche explorant l'expressivité de la logique en tant que moyen de calcul, par le biais de la *correspondance preuves-programmes*. Rappelons que cette dernière met en relation les preuves formelles d'une part et les programmes fonctionnels d'autre part, avec les deux dynamiques de la normalisation des preuves et de l'exécution des programmes. Cette correspondance est à l'origine de l'introduction, à la fin des années 1980, de la *logique linéaire*, une logique particulièrement adaptée à la notion de ressource, puis de ses généralisations algébriques, sous le terme de *géométrie de l'interaction*, où les preuves sont remplacées par des matrices ou des opérateurs d'espaces fonctionnels. Une approche naturelle pour explorer cette expressivité est d'analyser quelles classes de complexité peuvent être caractérisées. Cette thèse apporte ainsi des contributions aux domaines de la logique et de la *complexité implicite*, qui vise à donner des caractérisations de classes de complexité indépendantes des machines et ne faisant pas référence à des bornes explicites de temps ou d'espace.

L'auteur s'attache à deux formalismes particuliers, et met en évidence les classes de complexité qui peuvent être capturées, qui sont dans les deux cas des classes sous-polynomiales. Les deux formalismes sont dérivés de la logique linéaire, mais avec des points de vue différents. Le premier est un modèle non uniforme, celui des réseaux de preuves de la logique linéaire, plus précisément de la logique linéaire multiplicative (MLL), qui sont ici utilisés à la manière des circuits booléens. Le second formalisme est lui un modèle uniforme, défini par des opérateurs d'algèbres de von Neumann. Nous commencerons tout d'abord par résumer brièvement le contenu du manuscrit.

1 Contenu de la thèse

Le chapitre préliminaire 1 introduit le modèle des machines de Turing alternantes (ATM) et les classes de complexité qu'il permet de définir, puis le système de la logique linéaire avec ses preuves en calcul des séquents. Le Chapitre 2 est alors consacré aux réseaux de preuves de la logique linéaire, une représentation des preuves sous forme de graphes. On rappelle qu'il s'agit ici du noyau de la logique linéaire, le fragment multiplicatif (MLL), qui n'autorise pas la réutilisation de formule. Terui avait défini une correspondance entre réseaux de preuves et circuits booléens. L'auteur revisite cette correspondance en définissant une nouvelle classe de réseaux de preuves, les *proof circuits*. Pour cela il définit d'une part une traduction des circuits booléens dans les proof circuits, et d'autre part un circuit booléen permettant de normaliser les proof-circuits. Ceci lui permet de démontrer une relation étroite entre les classes de complexité PCC^i définies par les familles uniformes (et de taille polynomiale) de proof circuits de profondeur $\log^i n$, et celles, bien connue dans la littérature, des classes AC^i définies de manière analogue avec les circuits booléens : $\mathrm{AC}^i \subseteq \mathrm{PCC}^i \subseteq \mathrm{AC}^{i+1}$, pour $i \geq 0$. Ces résultats ont fait l'objet d'une publication dans les actes du workshop DICE 2011. Enfin

ce chapitre est complété par une correspondance entre les machines de Turing alternantes (ATM) et les circuits booléens, qui permet d'affiner les relations entre classes de complexité de proof circuits et classes de complexité de réseaux booléens : $\mathrm{bPCC}^i \subseteq \mathrm{AC}^i$ pour $i > 1$, où bPCC^i est définie comme PCC^i mais avec fan-in borné.

Le chapitre 3 aborde alors le deuxième thème de la thèse. Suivant le programme de la géométrie de l'interaction l'idée est ici d'explorer un modèle de calcul inspiré par la normalisation des preuves de la logique linéaire mais permettant de s'abstraire de la syntaxe avec un cadre plus mathématique. Dans le cas présent le modèle considéré est celui des algèbres d'opérateurs, et plus précisément des algèbres de von Neumann. L'idée de base est qu'un programme (dont on attend un résultat booléen) et une donnée (un mot) sont chacun représentés par un opérateur, et que l'exécution du programme sur cette donnée va être obtenue en itérant la composée des deux opérateurs. On dit alors que le programme *accepte* ce mot si cette composée est nilpotente. En s'appuyant sur des travaux en collaborations avec T. Seiller, l'auteur montre d'abord que cette condition d'acceptation, décider si un tel opérateur est nilpotent, peut être testée en espace logarithmique. Suivant la classe d'opérateurs considérée, définie par une condition de norme (cas général ou norme 1), le prédicat représenté est alors dans la classe non déterministe logspace NL (dans le cas général) ou la classe déterministe L (dans le cas de norme 1).

Enfin le chap. 4 s'attache à établir les résultats réciproques. L'auteur définit pour cela une nouvelle notion de machine à pointeurs (PM), proche des automates finis bidirectionnels, mais avec des choix de construction adaptés à ses objectifs, en particulier un ruban cyclique et une condition d'acceptation spécifique. Les langages reconnus par les PM déterministes (resp. non-déterministes) sont ceux de la classe L (resp. coNL) (rappelons qu'on sait que coNL=NL). Clément Aubert définit alors un encodage de ces machines à pointeurs dans les algèbres de von Neumann. Un point délicat ici consiste à définir cet encodage de manière à ce que les deux notions d'acceptation coïncident. Ceci établit alors que tous les langages de la classe NL sont représentables par des opérateurs. Dans le cas particulier des machines déterministes l'auteur montre alors que la condition de norme 1 est satisfaite. Au final il en déduit ainsi une caractérisation exacte de NL (dans le cas général) et de L (dans le cas de norme 1).

La thèse est complétée par deux appendices techniques. Le premier est une introduction aux algèbres de von Neumann, qui est un complément bienvenu ! Le second présente une preuve alternative du fait que tous les langages de coNL peuvent être reconnus par des PM non-déterministes, basée sur la définition d'une machine résolvant un problème coNL complet.

2 Commentaires et discussion

Il faut tout d'abord souligner le large spectre de ce travail, qui a requis de la part de l'auteur de maîtriser à la fois des notions de théorie de la démonstration, de modèles de calculs pour la complexité, et d'algèbres d'opérateurs. Qui plus est, l'introduction des algèbres de von Neumann en géométrie de l'interaction par J.-Y. Girard est très récente, et cette thèse fait partie des tout premiers travaux sur ce sujet.

J'ai particulièrement apprécié le fait que ce travail présente à la fois un versant syntaxique, avec l'étude des liens entre réseaux de preuves et circuits booléens, et un versant sémantique avec l'investigation détaillée d'un modèle de géométrie de l'interaction. Pour autant la thèse a une unité thématique car elle traite essentiellement de deux points de vue sur une même question, le traitement de données de taille arbitraire en logique : une première possibilité est l'approche non uniforme, illustrée par les circuits booléens, une seconde est de calculer dans des espaces permettant de représenter une forme d'infini, ce qui est discuté ici avec les algèbres d'opérateurs.

Concernant la première partie sur les proof circuits, une contribution notable est la traduction plus modulaire et directe qu'ils permettent des circuits booléens dans les réseaux, par rapport à la traduction antérieure de Terui, ainsi que le fait que les relations obtenues portent sur les classes DLOGTIME-uniformes. La gestion du *garbage* dans la traduction est aussi simplifiée. En revanche on aurait pu apprécier une discussion plus approfondie des différences entre la notion de proof circuit et celle a priori plus générale de réseau booléen. En effet comme ceux-ci sont définis de manière inductive, ne sont-ils pas, par construction même, plus proches des circuits booléens ? Cette question n'entame pas l'intérêt des constructions présentées,

mais pourrait être examinée pour des travaux futurs sur le sujet.

Dans la deuxième partie on peut noter tout d'abord l'approche très progressive adoptée par l'auteur pour présenter la géométrie de l'interaction dans les algèbres de von Neumann, s'appuyant sur les représentations par des matrices pour rendre l'intuition du calcul plus concrète. Il faut aussi souligner le soin tout particulier apporté à la démonstration de la borne de complexité logspace de la procédure de décision de la propriété de nilpotence (Thme 3.5.2 dans le cas non déterministe, et Thme 3.5.3 dans le cas déterministe). Ces résultats sont techniques et font partie des apports les plus remarquables de cette thèse. Ils s'appuient sur des énoncés intermédiaires non-triviaux, en particulier pour montrer comment se ramener de la question de la nilpotence d'un opérateur à celle d'une matrice.

Ensuite la démonstration de la borne inférieure d'expressivité, avec l'encodage d'une machine à pointeurs est traitée avec beaucoup de soin. Au lieu de se lancer dans un encodage fastidieux et malaisé à vérifier, d'une machine logspace standard, l'auteur choisit de commencer par un travail préliminaire consistant à définir une notion de machine bien adaptée à cet encodage. Ceci permet de décomposer notablement la difficulté. La thèse se conclut ainsi avec deux résultats très satisfaisants, puisqu'ils permettent de caractériser le pouvoir expressif des opérateurs représentant des prédicats dans ce modèle.

Une question tentante à ce stade est de se demander si, outre les prédicats, ce modèle pourrait aussi permettre de représenter des *fonctions* sur les mots, et ainsi de caractériser la classe FLOGSPACE des fonctions calculables en espace logarithmique. Ceci n'est a priori pas une extension évidente vue la notion spécifique d'acceptation qui est employée ici. Cette question pourrait peut-être apporter un éclairage intéressant sur la composition des fonctions FLOGSPACE, pour laquelle le mode de calcul de la géométrie de l'interaction semble bien adapté, ainsi que le suggèrent aussi par exemple les travaux de Dal Lago et Schöpp sur le langage de programmation logspace IntML. Mais il s'agit là d'un nouveau problème, et il est tout à fait normal que la thèse n'épuise pas le sujet.

3 Conclusion

Clément Aubert présente un ensemble de résultats très intéressants sur d'une part les liens entre logique linéaire et circuits booléens, et d'autre part une approche du calcul dans les algèbres d'opérateurs basée sur la logique. Soulignons que le document suit une progression très pédagogique : les notions sont bien choisies et amenées, avec des explications sur les subtilités et des exemples. Pour autant l'auteur fait preuve de précision et de rigueur dans les développements et les démonstrations. Cette thèse sera certainement une référence très utile pour qui souhaite approfondir le sujet de la géométrie de l'interaction dans les algèbres de von Neumann.

Pour toutes ces raisons je considère que Clément Aubert mérite le diplôme de Docteur de l'Université Paris 13 et je suis donc très favorable à la soutenance de sa thèse.

Patrick Baillot
Chargé de recherche au CNRS, Habilité à diriger les recherches.

ALMA MATER STUDIORUM — UNIVERSITA' DI BOLOGNA
DIPARTIMENTO DI INFORMATICA — SCIENZA E INGEGNERIA
MURA ANTEO ZAMBONI N. 7 40126 BOLOGNA
Tel. +39 051 20 94873 – Fax +39 051 20 94510

Bologna, November 4, 2013

Report on the PhD thesis entitled "Linear Logic and Sub-polynomial Classes of Complexity", by Clément Aubert.

Computational complexity is one of the most active branches of theoretical computer science. Its objective is the classification of computational problems into classes as for their inherent difficulty, measured by the amount of resources algorithms need when solving the problem at hand. Despite the enormous effort the scientific community is directing towards the study of complexity classes, not much is known about them. As an example, none of the many inclusions holding between classes amid logarithmic space computable functions and polynomial time computable functions is known to be strict or not. This shows that computational complexity is still in its infancy, but also that the aforementioned separation problems are hard.

This thesis is about implicit computational complexity (ICC), a research area which aims at characterizing complexity classes by tools drawn from mathematical logic and programming language theory, without explicit reference neither to machine models nor to combinatorial concepts like polynomial or logarithmic functions. More precisely, the thesis brings contributions to the understanding of the relations existing between fragments of linear logic and complexity classes smaller than the one of polynomial time computable functions.

A natural place where to start looking for simple and natural characterizations of complexity classes is functional computation, given its simplicity and elegance. Once features like higher-order functions come into play, however, keeping the complexity of the underlying program under control is notoriously more difficult. This problem has been alleviated by proof theory through the introduction of linear logic, a refinement of intuitionistic and classical logic in which what makes cut-elimination intractable, namely duplication, is confined to the so-called exponential modalities. This makes the logic more malleable — its expressive power can be tuned by just restricting the rules governing the exponential modality, without touching the other connectives. This is the path Girard, Lafont and others have followed when proposing so-called light logics. The latter are systems which can be proved to precisely capture relatively small classes of functions, like the one of polynomial time computable functions.

But what if we want to shed light on even smaller classes of complexity, like the plethora of classes lying between $\mathbf{L}$, the class of problems solvable in logarithmic space and $\mathbf{P}$, the class of problems solvable in polynomial time, the latter excluded? Is it that linear logic can be useful also in this context? Positive answers to this questions have been given in the last ten years, thanks to the work by Terui (later followed by Mogbil and Rahli), who showed that a deep relation exists between proof-nets of multiplicative linear logic and boolean circuits. Moreover, Girard showed recently that problems in $\mathbf{NL}$ can be characterized in a purely interactive way by operator algebras. Up to now, however, the contributions along this axis have been scattered.

This thesis shed some further light on the relations between small fragments of linear logic and sub-

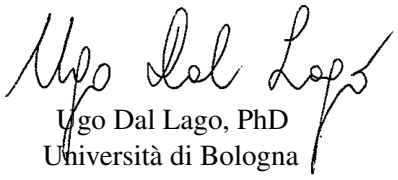
polynomial complexity classes: on the one hand results about proof circuits and parallel classes, like **NC** and **AC** are systematized, while on the other a careful analysis of operator algebras as a computational model if given, together with a proof that under some hypotheses the latter characterizes **L** and **NL**. Along the way, new computational models and complexity classes are introduced and studied, making the overall development rich and robust.

The contributions of the thesis are detailed below, together with a sketch of its content:

- After a brief Introduction, Chapter 1 presents some historical and technical preliminaries about complexity theory and linear logic. Computational models like Turing machines and Boolean circuits are introduced, together with classical encodings of circuits as proofs of multiplicative linear logic. References to related work are given and discussed.
- Chapter 2 is about Boolean circuits and the relationships between the latter and proof-nets of multiplicative linear logic. In particular, a new class of proof-nets, called *proof circuits*, is introduced and shown to be a subclass of Boolean proof-nets. The relations existing between Proof circuits, Boolean circuits and Alternating Turing Machines are then analyzed through a series of correspondence results.
- Chapter 3 shows how operator algebras and the hyperfinite factor can be turned into a computational model whose decision problem (i.e., given a string, decide if it is accepted or not) can be solved in logarithmic space. The introduced model is different than traditional computational models like automata, machines or programming languages: even the notion of data representation is quite peculiar, being based on normativity, and ultimately on interaction.
- Chapter 4 completes the picture by studying pointer machines, and by showing that a new sort of pointer machine, called a *nondeterministic pointer machine*, not only characterizes logarithmic space computation but can be simulated by operator algebras as defined in the previous chapter. This implies that operator algebras indeed characterize logarithmic space computation, both in the deterministic and in the nondeterministic cases. The first part of the chapter serves as a nice introduction to some of the many machine-based characterizations of logarithmic-space complexity from the literature.

The contributions of this thesis are interesting, and timely. Indeed, linear logic and the fine-grained view of computation it provides are ideal candidates when looking for a proof-theoretical interpretation of programs of subpolynomial complexity. The Geometry of interaction, on the other hand, not only provides a framework in which linear logic proofs can be faithfully interpreted, but is both operational in nature and deeply rooted in mathematics. The thesis is well written, and the reader is always gently introduced to each topic before facing the technical details, the latter making the thesis also enjoyable and not only valuable from a technical point of view. The comparison with related work is well done.

In conclusion, this thesis fully justifies the award of the title of “Docteur en Informatique” for Clément Aubert.



Ugo Dal Lago, PhD
Università di Bologna

UNIVERSITÉ PARIS 13 NORD

BUREAU DE LA RECHERCHE ET DES ETUDES DOCTORALES

PROCES - VERBAL DE SOUTENANCE

DOCTORAT DE L'UNIVERSITE PARIS 13
(arrêté du 5 Juillet 1984, modifié par l'arrêté du 30 Mars 1992)
Ecole Doctorale Sciences, Technologie, Santé, Galilée (ED 146)

SPECIALITE : **Informatique**

Le **Mardi 26 novembre 2013**, les membres du jury étant réunis sous la présidence de ARNAUD DURAND,

Monsieur Clément AUBERT

Né le **14 septembre 1984** à **Reims (93)**

a présenté une thèse de Doctorat intitulée :

Logique linéaire et classes de complexité sous-polynomiales

Les membres du jury soussignés, ayant délibéré, l'ont déclaré(e) digne du titre de Docteur de l'Université Paris 13 en **Informatique** et lui ont attribué la mention suivante :

□ Honorable ☒ Très honorable

ONT SIGNÉ :

Nom, Prénom, fonction ou qualité et lieu d'exercice de chaque membre du jury	Signatures
BAILLOT Patrick, Chargé de Recherche, Écoles normales supérieures (Lyon)	
DAL LAGO Ugo, Chargé de Recherche, Université de Bologne (Italie)	
DURAND Arnaud, Professeur, Université Paris 7	
FAGGIAN Claudia, Chargée de Recherche, Université Paris 7	
GUERRINI Stefano, Professeur, Université Paris 13	
MARION Jean-Yves, Professeur, L.O.R.I.A	
MELLIÈS Paul-André, Chargé de Recherche, Université Paris 7	
MOGBIL Virgile, Maître de Conférences, Université Paris 13	

[1] Par décision du Conseil Scientifique du 13 janvier 2009, approuvée par le Conseil d'Administration du 16 janvier 2009, la mention « Très honorable avec félicitations » a été supprimée pour les thèses soutenues dans le cadre de l'école doctorale Sciences, Technologie, Santé, Galilée.

UNIVERSITÉ PARIS 13 NORD

BUREAU DE LA RECHERCHE
ET DES ETUDES DOCTORALES

Villetaneuse, le 26/11/13

RAPPORT DE SOUTENANCE DE THESE

(DOCTORAT : UNIVERSITÉ)

SPECIALITE : **Informatique**

NOM et PRENOM du CANDIDAT : **Monsieur Clément AUBERT**

NOM DU PRESIDENT du JURY : ARNAUD DURAND

NOM DU RAPPORTEUR : PATRICK BAILLOT , UGO DAL LAGO

SIGNATURES des Membres du Jury :

BAILLOT Patrick, Chargé de Recherche, Écoles normales supérieures (Lyon)

DAL LAGO Ugo, Chargé de Recherche, Université de Bologne (Italie)

DURAND Arnaud, Professeur, Université Paris 7

FAGGIAN Claudia, Chargée de Recherche, Université Paris 7

GUERRINI Stefano, Professeur, Université Paris 13

MARION Jean-Yves, Professeur, L.O.R.I.A

MELLIÈS Paul-André, Chargé de Recherche, Université Paris 7

MOGBIL Virgile, Maître de Conférences, Université Paris 13

RAPPORT sur la SOUTENANCE :

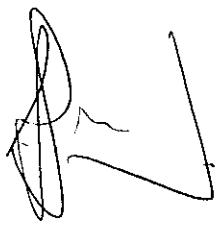
(Cette feuille constitue la première page du rapport de soutenance ; ce dernier sera libellé au verso de ce document et si nécessaire, des pages complémentaires peuvent être annexées).

RAPPORT SUR LA SOUTENANCE :

- Clément Aubert a donné une présentation très claire de son
- travail. Il a su dégager les idées clés d'un sujet qui est d'une
- difficulté technique reconnue notamment par ses interactions
- avec des mathématiques avancées comme les algèbres de von
- Neumann et la géométrie de l'interaction. Ses contributions
- touchent de nombreux domaines de l'informatique comme la
- logique, la complexité et les modèles de calcul.
- L'exposé a clairement montré sa maturité scientifique et son
- recul, son esprit de synthèse et son ouverture scientifique, autant
- de qualités déjà remarquées par les rapporteurs à la lecture de
- son manuscrit.
- Clément Aubert a répondu avec assurance et enthousiasme aux
- nombreuses questions du jury et a développé des perspectives
- de recherche convaincantes.
- Pour toutes ses raisons, le jury lui décerne le grade de docteur
- de l'Université Paris 13, spécialité informatique avec la mention
- très honorable.
-
-
-
-

-- **SIGNATURE DU PRESIDENT DU JURY**

-
-
-
-



Activités de recherche et d'enseignement

Activités de recherche

Mon activité de recherche en Informatique se conçoit sous l'angle de la logique, et se rattache à la compréhension de la complexité d'un programme informatique grâce à la correspondance de Curry-Howard. Cette correspondance établit un isomorphisme entre réécriture de preuves et exécution d'un programme exprimé en langage abstrait avec spécification. Elle permet de lier la théorie de la démonstration — l'étude des preuves et de leur dynamique — et l'étude des programmes. Par le jeu des théorèmes de correction et de complétude, on peut faire entrer dans ce panorama l'ensemble des outils mathématiques servant à interpréter les preuves. Parmi ceux-ci, on peut citer les $\mathbb{C}^*$-algèbres, les algèbres de von Neumann ou encore une construction connue sous le nom du semi-anneau d'unification. Je reviendrai sur ce dernier point et sur les avancées qu'il a permis ultérieurement.

On peut étendre ces correspondances à des considérations quantitatives et contribuer ainsi à la théorie de la complexité. Ce domaine hiérarchise la difficultés des problèmes en fonction des ressources nécessaires pour les résoudre, et propose un classement en classes de complexité, qui regroupent des problèmes asymptotiquement « aussi durs » les uns que les autres. De par le fin découpage de la logique classique et de par l'attention à la gestion des ressources qu'elle propose, la logique linéaire est toute indiquée pour rendre compte de ces questions de coûts. Elle le fait indépendamment d'une référence à un modèle de machine abstraite : il s'agit là de la complexité implicite, domaine de recherche où la logique linéaire joue un rôle majeur. Il existe déjà une importante communauté qui travaille sur les liaisons entre logique linéaire, interprétation de preuves, modèles de calcul et complexité : on peut citer en France les équipes L.C.R.*, à Paris 13, P.P.S.*, à Paris 7, Plume, à l'E.N.S. Lyon, L.D.P.* à l'I.M.L., et à l'étranger Focus*, à l'Université de Bologne, Logic and Semantics, Mathematical Foundations Computer Science, à l'Université de Bath, ou encore au *Research Institute for Mathematical Sciences*, à l'Université de Kyoto [1].

Mon parcours n'a pas été linéaire, et c'est d'abord un cursus philosophique (LOgique, Philosophie, HIstoire et Sociologie des Sciences) qui m'a formé à la rigueur de la logique mathématique. Mon travail de mémoire [8], sous la direction de Jean-Baptiste Joinet, envisage sous un angle technique un fragment de la logique qui pose de nombreuses interrogations lorsqu'il est considéré à l'aune de sa sémantique de Kripke. Ce travail reprenait et améliorait une preuve de l'élimination des coupures de ce fragment, lui conférant un pouvoir calculatoire, une dynamique. Mes connaissances mathématiques ont ensuite été consolidées en Master 2 de Mathématiques (Logique, Mathématiques et Fondements de l'Informatique) avant d'étudier les ponts entre logique et théorie de la complexité, en stage puis en thèse, tous deux menés au Laboratoire d'Informatique de Paris Nord [2] (Université de Paris 13). J'ai ensuite poursuivi et étendu mes activités de recherche en post-doctorat à l'I2M (Université d'Aix-Marseille) et au LACL (Université Paris-Est Créteil), en étant employé respectivement par le CNRS et l'INRIA.

Virgile Mogbil et Paulin Jacobé de Naurois ont su m'aiguiller vers la complexité en me proposant un sujet de stage qui nécessitait de se familiariser avec un modèle classique et néanmoins exigeant du calcul parallèle, les circuits booléens. Mon travail de mémoire [7] a consisté à comprendre, affiner et perfectionner la façon dont un fragment de la logique linéaire, présenté à l'aide d'une syntaxe parallèle connue sous le nom de réseau de preuve, pouvait calculer en parallèle. Ce travail a fait l'objet d'une publication [6] dans les actes du *workshop* international DICE [3] et a donné lieu à de nombreuses communications, en France et à l'étranger, notamment aux 16èmes rencontres du groupe de travail logique, Algèbre et Calcul du GDR-IM [4] et pendant les sessions d'audience internationale *Logic and interactions 2012* [5], organisées au Centre International de Rencontres Mathématiques.

Cette recherche constitue une partie de ma thèse [4], également menée au L.I.P.N., et se conçoit ainsi : il est possible d'associer à tout circuit booléen un réseau de preuve de type booléen, de profondeur et de taille équivalentes, qui se normalise — c'est-à-dire qui s'évalue — en un temps parallèle raisonnable. En simplifiant la cible de la traduction, j'ai abaissé la complexité de cette traduction, qui nécessitait un espace logarithmique en la taille du circuit sur une machine de Turing, et peut désormais s'effectuer avec un circuit booléen de profondeur constante. J'ai poussé plus loin cette correspondance dans ma thèse, en liant ce modèle de calcul

1. J'ai animé les séminaires des équipes dont le nom est suivi de *.
2. `http://lipn.fr/`
3. `http://dice11.loria.fr/`
4. `http://www.pps.univ-paris-diderot.fr/~saurin/GT-LAC/`
5. `http://li2012.univ-mrs.fr/`

avec les machines de Turing alternantes, qui sont un autre modèle classique du parallélisme. Il s'agit là de la première correspondance de la sorte, rendue possible par la structure contrainte que j'ai donné aux réseaux de preuves booléens.
J'ai développé les correspondances entre logique linéaire et complexité sous un autre angle durant ma thèse, largement en collaboration [6] avec Thomas Seiller (actuellement post-doctorant à l'Institut des Hautes Etudes Scientifiques, doctorant à l'Institut Mathématiques de Luminy, à Marseille, durant notre collaboration). Notre premier article [3] pose le cadre de cette étude en montrant comment une interprétation de la logique linéaire connue sous le nom de Géométrie de l'interaction capture la classe de complexité de l'espace logarithmique non-déterministe **NL**. Cette construction développe un cadre théorique assez complexe, en reposant sur l'outillage mathématique des algèbres de von Neumann. Ce cadre mathématique offre une interprétation assez spécifique des preuves représentant les entiers, puisque sont représentées certaines propriétés de l'entier plutôt que l'entier lui-même. Il est ensuite possible de faire jouer cette représentation contre des matrices représentant des programmes. Notre travail a entre autres consisté à détailler ces matrices, pour s'apercevoir que leur comportement calculatoire correspondait à un calcul de machine à pointeurs. Nous avons ensuite étendu cette correspondance au cadre déterministe [5] et l'avons consolidé par la même. Ce travail a fait l'objet de nombreuses communications, au *workshop* international *Logic and Computational Complexity* 2013 [7], et aux séminaires des équipes logique de la Programmation (I.M.L., Marseille), Focus (Université de Bologne) [8], logique, Calcul et Raisonnement (Paris 13) et au Groupe de Travail sémantique, à P.P.S. (Paris 7). Ces deux travaux sont toujours en cours d'évaluation pour les journeaux *Mathematical Structures in Computer Science* et *Information & Computation*.
La rédaction de ma thèse [4] m'a permis de reprendre ces travaux et de les présenter de façon homogène et détaillée. J'y ai adjoint une annexe présentant l'outillage mathématique employé, ce qui fût je crois apprécié par ma communauté. La rédaction m'a également poussé à étudier plus en détail la notion de pointeur et l'imposante littérature qui s'y rapporte, car le calcul que nous avons présenté plus haut se rapporte intimement aux automates bi-directionnels multi-têtes.
Mon arrivée à l'I2M, à Marseille, m'a permis de travailler sur un nouvel axe : les liens entre calculs des processus et logique linéaire, avec Emmanuel Beffara. J'ai également poursuivi, avec Marc Bagnol et Paolo Pistone, tous deux doctorants, l'étude des derniers développements de la Géométrie de l'interaction, que nous avons rapprochés de mes précédents travaux en complexité. Ces travaux, qui ont fait l'objet de deux publications [1, 2], proposent une nouvelle correspondance « quantitative » entre automates et programmation logique, *via* la théorie de la démonstration. Ils étendent, raffinent et solidifient les travaux précédemment menés.
Je me permets de souligner que tous mes co-auteurs étaient doctorants ou jeunes docteurs au moment de nos collaborations. Je me suis permis de vous joindre une lettre de recommandation de mon ancien directeur de thèse, page **??**, il me semble également que les pré-rapports et rapports de ma soutenance de thèse (p. 7-14) témoignent de mon engagement et de mes capacités de recherche.

Journeaux d'audience internationale avec comité de sélection

[3] **Clément Aubert** et Thomas Seiller. « Characterizing co-NL by a group action ». In : *Mathematical Structures in Computer Science* FirstView (déc. 2014), p. 1–33. ISSN : 1469-8072. DOI : 10.1017/S0960129514000267.

Publications d'audience internationale avec comité de sélection

[1] **Clément Aubert** et Marc Bagnol. « Unification and Logarithmic Space ». In : *Rewriting and Typed Lambda Calculi*. Sous la dir. de Gilles Dowek. T. 8650. Lecture Notes in Computer Science. Springer, 2014, p. 77–92. ISBN : 978-3-319-08917-1. DOI : 10.1007/978-3-319-08918-8_6.

[2] **Clément Aubert**, Marc Bagnol, Paolo Pistone et Thomas Seiller. « Logic Programming and Logarithmic Space ». In : *Asian Symposium on Programming Languages and Systems*. Sous la dir. de Jacques Garrigue. T. 8858. Lecture Notes in Computer Science. Springer, 2014, p. 39–57. DOI : 10.1007/978-3-319-12736-1_3.

6. Et notamment lors d'une visite de deux semaines à Marseille, rendue possible par des fonds du GDR-IM que j'avais moi-même sollicités. Cette visite m'a également permise d'animer le groupe de travail et le séminaire de l'équipe Logique de la Programmation qui m'accueillait.
7. http://www.cs.swansea.ac.uk/lcc2013/
8. Lors d'une visite de deux semaines à Ugo Dal Lago, permise par des fonds d'un PICS, « logique linéaire et applications ».

[6] **Clément Aubert**. « Sublogarithmic uniform Boolean proof nets ». In : *Developments in Implicit Computational Complexity*. Sous la dir. de Jean-Yves Marion. T. 75. Electronic Proceedings in Theoretical Computer Science. 2011, p. 15–27. DOI : 10.4204/EPTCS.75.2.

Articles soumis à des revue d'audience internationale avec comité de sélection

[5] **Clément Aubert** et Thomas Seiller. « Logarithmic Space and Permutations ». In : *Arxiv preprint* abs/1301.3189 (2013). arXiv : 1301.3189 [cs.LO]. Soumis à Information & Computation — Special Issue on Implicit Computational Complexity, 26 p.

Travaux universitaires

[4] **Clément Aubert**. « Linear Logic and Sub-polynomial Classes of Complexity ». Thèse de doct. Université Paris 13–Sorbonne Paris Cité, nov. 2013. Sous la dir. de Stefano Guerrini, Virgile Mogbil (UMR CNRS 7030 — Paris 13), 184 p.

[7] **Clément Aubert**. « Réseaux de preuves booléens sous-logarithmiques ». Mém.de mast. L.I.P.N. : L.M.F.I., Paris VII, sept. 2010. Sous la dir. de Virgile Mogbil, Paulin Jacobé de Naurois (UMR CNRS 7030 — Paris 13), 29 p.

[8] **Clément Aubert**. « L'élimination des coupures dans la Logique des Domaines Constants ». Mém.de mast. Paris 1, 2009. Sous la dir. de Jean-Baptiste Joinet (UMR CNRS 7126 — Paris 7), 29 p.

Activités d'enseignement

Durant mes trois années de doctorat (2010–2013) au L.I.P.N. (Université de Paris 13), j'ai été chargé d'une mission d'enseignement [9] à l'I.U.T. de Villetaneuse, au sein du D.U.T. Réseaux & Télécommunications [10]. Je suis actuellement (2014) vacataire à l'Université de Paris-Est Créteil, au département Informatique.
Je suis intervenu à l'I.U.T. en première année, avec une complexification croissantes des contenus, et en variant les sujets enseignés. J'ai ainsi dû rapidement me former à un langage de programmation (le `C`), me familiariser avec l'administration réseaux de systèmes *NIX et avec un langage de requêtes (`SQL`). J'avais pu auparavant perfectionner ma méthode d'enseignement en intervenant dans des modules moins exigeants pour moi en terme de contenus, mais nécessitant un grand effort pédagogique : les modules « Apprendre Autrement », au contenu mathématique, m'ont poussé à être vigilant sur l'encadrement d'élèves et à redoubler d'efforts pour présenter de façon innovante des contenus posant des difficultés aux élèves. Ces modules m'ont également permis de me familiariser avec le génie logiciel et d'enseigner aux élèves à structurer et répartir le travail entre eux.
Le contenu des enseignements en I.U.T. est fixé par le Programme Pédagogique National [11], vous pouvez vous y référer pour le détail des programmes de ces modules. Je vous renvoie au tableau page suivante pour le détail de la répartition de mon service (approximativement 190 heures) durant ces trois années. Les volumes horaires en gras y sont exprimés en heures équivalentes TP / TD.
Ce service est varié et démontre je crois ma capacité à acquérir rapidement des contenus spécialisés. J'aurai plaisir à enseigner dans ces domaines, mais également dans tous les domaines relatifs à l'informatique, avec une mention spéciale pour la programmation web, que je pratique en tant que webmestre de diverses associations. Mon expérience, ma formation ainsi que mes loisirs me poussent à croire que je serai aussi à l'aise en programmation, en administration système, en bases de données, en programmation web ou sur des pans plus théoriques.
Vous pouvez retrouver aux pages 67 à 69 trois lettres de recommandations de collègues avec qui j'ai eu la chance d'enseigner.

2014

	Initiation à l'algorithmique et à la complexité (Sem. 3) Resp. : Sergey Verlan
	Concepts fondamentaux de l'algorithmique : récursion, piles, *backtracking*, représentation de données et problèmes sur les graphes. Scéances de TP en `C`.
35h	Réparties en 11 séances de 1 heure 30 de TD et 6 séances de 3 heures de TP.

9. Sous le statut d'« allocataire-moniteur ».
10. `http://www.iutv.univ-paris13.fr/formations/dut/reseaux-et-telecommunications.html`
11. Dans sa version antérieure à la rentrée 2013 en ce qui concerne mes enseignements, disponible en ligne à `http://media.enseignementsup-recherche.gouv.fr/file/DUT_-_Programmes_pedagogiques_nationaux/40/0/PPN_Reseaux_Telecommunication_Rentree08_32400.pdf`.

2012-2013

I4 **Bases de données** (Sem. 2) Resp. : Jean-Michel Barrachina

Modèles théoriques (UML, diagrammes de classes, normalisation, algèbre relationnelle) et pratique de la conception et administration de bases de données : requêtes `SQL`, contraintes, vues, fonctions, programmation `C / SQL`.

24h Réparties en 4 séances de 3 heures de TD et 4 séances de 3 heures de TP.

❁

R3 **Administration des systèmes d'exploitation réseaux** (Sem. 2) Resp. : Laure Petrucci

Contenu sensiblement similaire à celui de 2011-2012, avec implication croissante dans la rédaction d'exercices, d'évaluations, et proposition de corrections dans le polycopié.

27h Réparties en 9 séances de 3 heures de TP-TD.

❁

R3 **Administration des systèmes d'exploitation réseaux** (Sem. 2) Resp. : Laure Petrucci

Contenu sensiblement similaire à celui de 2011-2012, avec implication croissante dans la rédaction d'exercices, d'évaluations, et proposition de corrections dans le polycopié.

27h Réparties en 9 séances de 3 heures de TP-TD.

2011-2012

I3 **Algorithmique et programmation** (Sem. 1) Resp. : Camille Coti [12]

Mise en place d'un nouveau cours d'algorithmique et de programmation en `C`. Le rythme alternait séances de TD expliquant les concepts-clés d'algorithmique en language naturel et TP. Son contenu était ambitieux : boucles, bibliothèques graphiques, pointeurs, jusqu'aux listes chaînées. Le contenu peut être consulté à `http://lipn.fr/~coti/cours/#I3`.

51h Réparties en 8 séances de 3 heures de TP et 9 séances de 3 heures de TD.

❁

R3 **Administration des systèmes d'exploitation réseaux** (Sem. 2) Resp. : Laure Petrucci [13]

Initiation à la rédaction de *scripts*, bases de l'administration système `*NIX` (gestion d'usagers, droits, système de fichier), configuration de cartes réseaux et contrôle des paquets (Wireshark). Le contenu peut être consulté à `http://lipn.fr/~petrucci/cours_R3.pdf` et `http://lipn.fr/~petrucci/tp_R3.pdf`.

27h Réparties en 9 séances de 3 heures de TP-TD.

❁

AA2 **Apprendre Autrement** (Sem. 2) Resp. : Clément Aubert

Contenu sensiblement similaire à celui développé en 2010-2011, sans CM cette fois. J'étais responsable de cette formation (concertation avec mes collègues, conception de la grille d'évaluation, référent lors du jury).

20h Réparties en 5 séances de 4 heures de TP-TD

2010-2011

AA1 **Apprendre Autrement** (Sem. 1) Resp. : Fayssal Benkhaldoun

Cours d'aide aux élèves, alors en difficulté avec les Mathématiques. L'objectif était de leur faire passer le contenu du programme différemment, avec recours à la plate-forme d'apprentissage Wims (`http://wims.unice.fr/wims/`). Nécessité d'une concertation avec les responsables des modules mathématiques, rédaction d'exercices, suivi personnalisé.

18h Réparties en 6 séances de 3 heures de TD.

❁

AA2 **Apprendre Autrement** (Sem. 2) Resp. : Fayssal Benkhaldoun

Encadrement de projets de 25 élèves répartis en groupes de 2 à 5. J'ai conçu un cours introductif sur le génie logiciel et ai guidé les élèves dans la réalisation de leur logiciel (rédaction d'un cahier des charges, répartition du travail, suivi et évaluation). Les projets étaient largement développés à domicile, les séances de TP-TD servant de marqueurs d'étapes dans leurs réalisations. Les languages choisis étaient nombreux et variés (`C`, `Java`, `PHP / mysql`, `HTML`).

23h15 Réparties en 7 séances de 3 heures de TP-TD et un CM de 1h30.

12. Dont vous pourrez retrouver une lettre de recommandation p. 69.
13. Dont vous pourrez retrouver une lettre de recommandation p. 67.

Sublogarithmic uniform Boolean proof nets

Clément Aubert*
LIPN – UMR7030, CNRS – Université Paris 13,
99 av. J.-B. Clément, 93430 Villetaneuse, France

Abstract Using a proofs-as-programs correspondence, Terui was able to compare two models of parallel computation: Boolean circuits and proof nets for multiplicative linear logic. Mogbil *et. al.* gave a logspace translation allowing us to compare their computational power as uniform complexity classes. This paper presents a novel translation in AC^0 and focuses on a simpler restricted notion of uniform Boolean proof nets. We can then encode constant-depth circuits and compare complexity classes below logspace, which were out of reach with the previous translations.

Introduction

Boolean proof nets were introduced by Terui in [8] to study the implicit complexity of *proofs nets for Multiplicative Linear Logic* [4] comparatively to Boolean circuits. Those two models of parallel computation were successfully linked using a *proofs-as-programs* framework, which matches up *cut-elimination* in proof nets with *evaluation* in circuits. Surprisingly [8] does not take into account uniformity, which guarantees that the resources needed to build a Boolean circuit is inferior to the computational power it will deliver. [7] and [6] studied the Boolean proof nets in a uniform way and introduced some non-determinism in it. As their translation from Boolean circuit families to Boolean proof net families is in logspace (L), it remained unknown if the results were still valid when applied to sublogarithmic classes of complexity, that is to say AC^0 and NC^1. By restricting the Boolean proof nets we use, this paper offers a new proof of the correspondence between circuits and proof nets and extends it to constant-depth circuits.

Boolean circuits ([9], section 1) and proof nets ([3], section 3) are canonical models of parallel computation, but the latter was mostly seen from the viewpoint of *sequential* implicit complexity. To *evaluate* a proof net is to eliminate its cuts, but to do so with suitable bounds we need to define a parallel elimination. Trying to apply the two usual rules of rewriting ($\rightarrow_m$ and $\rightarrow_a$) in parallel leads to critical pairs, so we are forced to define a new kind of cut (*tightening-cut*) and a new rewriting rule ($\rightarrow_t$). The simulation of this reduction rule by Boolean circuits needs $UstConn_2$ gates to be made with efficiency.

The proof nets we study are for Multiplicative Linear Logic with unbounded arity (**MLLu**, section 2) and because of the linearity of this logic, we are forced to keep track of the partial results generated by the evaluation that are unused in the result. Boolean proof nets (section 4) – as introduced by Terui – have an expensive way of manipulating this *garbage*. In this paper we introduce *proof circuits* (section 5) as a refinement of the Boolean proof nets that are simpler to manipulate. They are made of *pieces* which translate *gates* and compose easily, so that we reduce the size of the proof net translating the Boolean circuits. In section 6 we conclude our paper with our main result (theorem 2): there exists a constant-depth reduction from Boolean circuit families to proof circuit families. So our new framework offers a variant of the proofs for complexity results and extends them to small classes of complexity, in a uniform way.

*Work partially supported by the French project Complice (ANR-08-BLANC-0211-01).

Jean-Yves Marion (Ed.): Second Workshop on
Developments in Implicit Computational Complexity (DICE 2011)
EPTCS 75, 2012, pp. 15–27, doi:10.4204/EPTCS.75.2

1 Boolean circuits

Boolean circuits (definition 2) are of great interest in the study of complexity, for instance because of the efficiency of their parallel evaluation. One of their features is that they work only on inputs of fixed length, and that forces us to deal with *families* of Boolean circuits – and there arises the question of *uniformity* (definition 4).

Definition 1 (Boolean function)**.** A *n-ary Boolean function* f^n is a map from $\{0,1\}^n$ to $\{0,1\}$. A *Boolean function family* is a sequence $f = (f^n)_{n\in\mathbb{N}}$ and a *basis* is a set of Boolean functions and Boolean function families. We set :

$$\mathfrak{B}_0 = \{\neg, \vee^2, \wedge^2\} \text{ and } \mathfrak{B}_1 = \{\neg, (\vee^n)_{n\geqslant 2}, (\wedge^n)_{n\geqslant 2}\}$$

The Boolean function $UstConn_2$, given in input the coding of an undirected graph G of degree at most 2 and two names of gates s and t, outputs 1 iff there is a path between s and t in G.

Definition 2 (Boolean circuits)**.** Given a basis $\mathfrak{B}$, a *Boolean circuit over $\mathfrak{B}$ with n inputs C* is a directed acyclic finite and labeled graph. The nodes of fan-in 0 are called *input nodes* and are labeled with $x_1, \dots, x_n, 0, 1$. Non-input nodes are called *gates* and each one of them is labeled with a Boolean function from $\mathfrak{B}$ whose arity coincides with the fan-in of the gate. There is a unique node of fan-out 0 which is the *output gate*. We indicate with a subscript the number of inputs: a Boolean circuit C with n inputs will be named C_n.

The *depth of a Boolean circuit C_n* $d(C_n)$ is the length of the longest path between an input node and the output gate. Its *size* $|C_n|$ is its number of nodes. We will only consider Boolean circuits of size $n^{O(1)}$, that is to say polynomial in the size of their input.

C_n accepts a word $w \equiv w_1 \dots w_n \in \{0,1\}^n$ if C_n evaluates to 1 when $w_1, \dots, w_n$ are respectively assigned to $x_1, \dots, x_n$. A *family of Boolean circuits* is an infinite sequence $C = (C_n)_{n\in\mathbb{N}}$ of Boolean circuits, *C accepts a language $X \subseteq \{0,1\}^*$* iff for all $w \in X$, $C_{|w|}$ accepts w.

We now recall the definition of the *Direct Connection Language* of a family of Boolean circuits, an infinite sequence of tuples that describes it totally.

Definition 3 (Direct Connection Language [9])**.** Given $\overline{(.)}$ a suitable coding of integers and $C = (C_n)_{n\in\mathbb{N}}$ a family of Boolean circuits over a basis $\mathfrak{B}$, its *Direct Connection Language* – written $L_{DC}(C)$ – is the set of tuples $< y, \overline{g}, \overline{p}, \overline{b} >$, such that for $|y| = n$, we have: g is a gate in C_n, labeled with $b \in \mathfrak{B}$ if $p = \epsilon$, else b is its p^{th} predecessor.

Definition 4 (Uniformity [1])**.** A family C is said to be *DLOGTIME*-uniform if there exists a deterministic Turing Machine with random access to the input tape that given $L_{DC}(C)$, n and $\overline{g}$ outputs in time $O(\log(|C_n|))$ any information (position, label or predecessors) about the gate g in C_n.

Despite the fact that a *DLOGTIME* Turing Machine has more computational power than a constant-depth circuit, "*a consensus has developed among researchers in circuit complexity that this DLOGTIME uniformity is the 'right' uniformity condition*" for small complexity classes [5]. Any further reference to uniformity is to be read as *DLOGTIME* uniformity.

Definition 5 (AC^i, NC^i)**.** For all $i \in \mathbb{N}$, given $\mathfrak{B}$ a basis, a language $X \subseteq \{0,1\}^*$ belongs to the class $AC^i(\mathfrak{B})$ (resp. $NC^i(\mathfrak{B})$) if X is accepted by a uniform family of polynomial-size, $\log^i$-depth Boolean circuits over $\mathfrak{B}_1 \cup \mathfrak{B}$ (resp. $\mathfrak{B}_0 \cup \mathfrak{B}$). We set $AC^i(\emptyset) = AC^i$ and $NC^i(\emptyset) = NC^i$.

2 MLLu

Rather than using Multiplicative Linear Logic (**MLL**) – which would force us to compose binary connectives to obtains n-ary connectives – we work with **MLLu** which differs only on the arities of the connectives but better relates to circuits. We write $\overrightarrow{A}$ (resp. $\overleftarrow{A}$) for an ordered sequence of formulae $A_1, \dots, A_n$, (resp. $A_n, \dots, A_1$).

Definition 6 (Formulae of **MLLu**)**.** Given α a literal and $n \geqslant 2$, formulae of **MLLu** are:

$$A ::= \alpha \mid \alpha^\perp \mid \otimes^n(\overrightarrow{A}) \mid \parr^n(\overleftarrow{A})$$

Duality is defined with respect to De Morgan's law :

$$\begin{aligned} (A^\perp)^\perp &\equiv A \\ (\otimes^n(\overrightarrow{A}))^\perp &\equiv \parr^n(\overleftarrow{A^\perp}) \\ (\parr^n(\overleftarrow{A}))^\perp &\equiv \otimes^n(\overrightarrow{A^\perp}) \end{aligned}$$

As for the rest of this article, consider that A, B and D will refer to **MLLu** formulae. $A[B/D]$ denotes A where every occurrence of B is replaced by an occurrence of D. We write $A[D]$ if $B = \alpha$.

Definition 7 (Sequent calculus for **MLLu**)**.** A *sequent* of **MLLu** is of the form $\vdash \Gamma$, where Γ is a multiset of formulae. The *inference rules* of **MLLu** are as follow :

$$\frac{}{\vdash A, A^\perp}\, ax. \qquad \frac{\vdash \Gamma_1, A_1 \quad \dots \quad \vdash \Gamma_n, A_n}{\vdash \Gamma_1, \dots, \Gamma_n, \otimes^n(\overrightarrow{A})}\, \otimes^n$$

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\, cut \qquad \frac{\vdash \Gamma, \overleftarrow{A}}{\vdash \Gamma, \parr^n(\overleftarrow{A})}\, \parr^n$$

*Derivations of **MLLu*** are built with respect to those rules. **MLLu** has neither weakening nor contraction, but admits implicit exchange and eliminates cuts. The formulae A and $A^\perp$ in the rule *cut* are called the *cut formulae*.

3 Proof nets

Proof nets are a parallel syntax for **MLLu** that abstract away everything irrelevant and only keep the structure of the proofs. We introduce measures (definition 10) on them in order to study their structure and complexity, and a parallel elimination of their cuts (definition 11).

Definition 8 (Links)**.** We introduce in figure 1 three sorts of *links* – $\bullet$, $\otimes^n$ and $\parr^n$ – which correspond to **MLLu** rules.

Every link may have two kinds of *ports*: *principal* ones, indexed by 0 and written below, and *auxiliary* ones, indexed by $1, \dots, n$ and written above. The auxiliary ports are ordered, but as we always represent the links as in figure 1, we may safely omit the numbering. Axiom links have two principal ports, both indexed with 0, but we can always differentiate them (for instance by naming them 0_r and 0_l) if we need to.

Remark 1. There is no sort cut: a cut is represented with an edge between two principal ports.

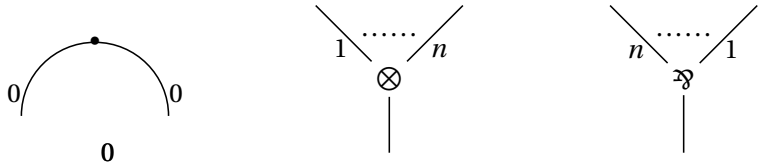


Figure 1: ax-link, $\otimes^n$-link and $\parr^n$-link

Definition 9 (Decorated derivations and proof net)**.** Given a derivation of **MLLu**, we decorate it in the following way:

- an index is associated to every formula. The formulae introduced by an axiom have the same proper index, and the formulae introduced by a logical rules ($\otimes^n$ and $\parr^n$) have a *fresh* index,
- a *description* is associated to every sequent.

The rules given in figure 2 indicate how a proof is decorated and how to build a *proof net* from a description.

The *type of a proof net* P is Γ if there exists a decorated derivation of $\vdash \Gamma \triangleright \mathscr{D}(P)$: a proof net always has several types, but up to α-equivalence (renaming of the literals) we may always assume it has a unique *principal type*. If a proof net may be typed with Γ, then for every A it may be typed with $\Gamma[A]$. By extension we will use the notion of type of an edge.

The structures obtained by following those rules respect criterion of correctness. For instance two ports of a same link may not be connected, a port may be connected only once and every auxiliary port is connected. We do not have to take into account *pseudo nets*.

Remark 2. The same proof net – as it abstracts derivations – may be induced by several descriptions. Conversely, several graphs – as representations of proof nets – may correspond to the same proof net: we get round of this difficulty by associating to every proof net a single drawing among the drawings with the minimal number of crossings between edges, for the sake of simplicity. Two graphs representing proof nets that can be obtained from the same description are taken to be equal.

Definition 10 (Size and depth of a proof net)**.** The size $|P|$ of a proof net P is the number of its links.

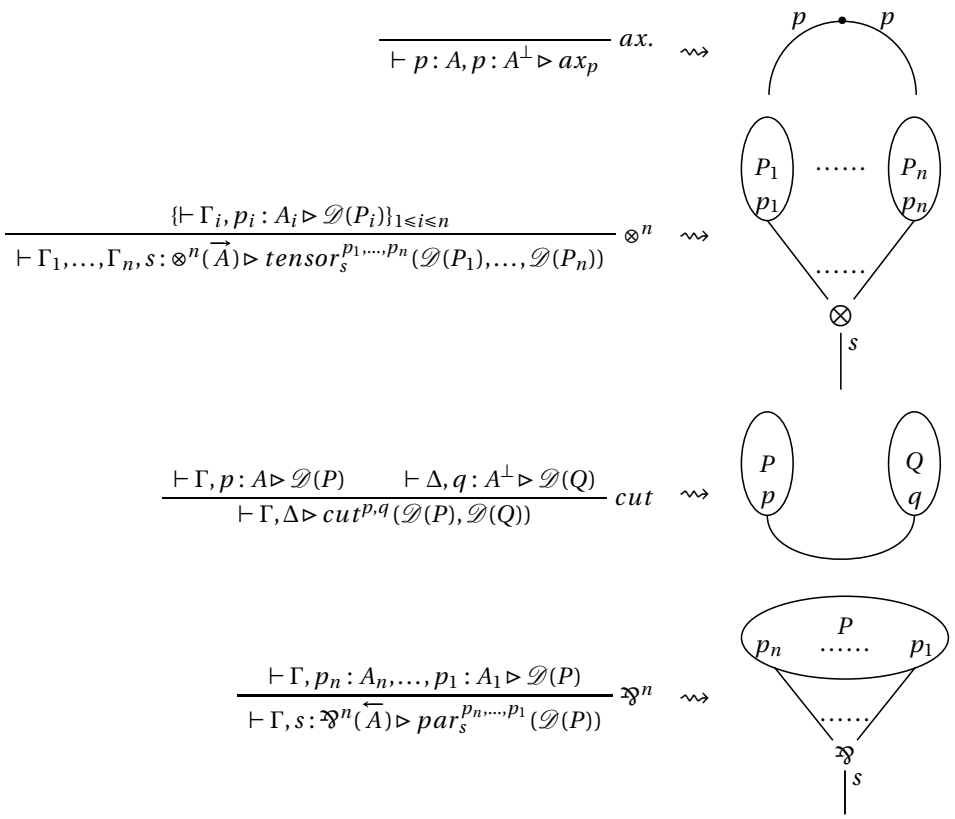
The depth of a proof net is defined with respect to its type:

- The depth of a formula is defined by recurrence:
  $d(\alpha) = d(\alpha^\perp) = 1$
  $d(\otimes^n(\overrightarrow{A})) = d(\parr^n(\overleftarrow{A})) = 1 + max(d(A_1), \dots, d(A_n))$
- The depth $d(\pi)$ of a derivation π is the maximum depth of cut formulae in it.
- The depth $d(P)$ of a proof net P is

$$min\{d(\pi) \mid \pi \text{ can be decorated as } \vdash \Gamma \triangleright \mathscr{D}(P) \text{ for some } \Gamma\}$$

  The depth $d(P)$ of a proof net depends on its type, but it is minimal when we consider the principal type of the proof net.

To make the most of the computational power of proof nets, we need to achieve a speed-up in the number of steps needed to normalize them. If we try roughly to reduce in parallel a cut between two ax-links, we are faced with a critical pair. [8] avoids this situation by using a *tightening reduction* which eliminates in one step all the cuts between axioms. We can then safely reduce all the other cuts in parallel.



Edges representing Γ or Δ are not drawn, $\mathscr{D}(P)$ is one of the description of the proof net P.

Figure 2: From decorated derivations to proof nets

Definition 11 (Cuts and parallel cut-elimination)**.** A cut is an edge between the principal ports of two links. If one of these links is an ax-link, two cases occurs:

if the other link is an ax-link, we take the maximal chain of ax-links connected by their principal ports and defines this set of cuts as a *t-cut*,

otherwise the cut is an *a-cut*.

Otherwise it is a *m-cut* and we know that for $n \geqslant 2$, one link is a $\otimes^n$-link and the other is a $\parr^n$-link.

We define on figure 3 three rewriting rules on the proof nets. For $r \in \{t, a, m\}$, if Q may be obtained from P by erasing all the r-cuts of P in parallel, we write $P \rightrightarrows_r Q$. If $P \rightrightarrows_t Q$, $P \rightrightarrows_a Q$ or $P \rightrightarrows_m Q$, we write $P \rightrightarrows Q$. To *normalize a proof net P* is to apply $\rightrightarrows$ until we reach a cut-free proof net. $\rightrightarrows^*$ is defined as the transitive reflexive closure of $\rightrightarrows$.

Theorem 1 (Parallel cut-elimination [8])**.** *Every proof net P normalizes in at most $O(d(P))$ applications of $\rightrightarrows$.*

So the time needed to evaluate a proof net is relative to its depth, and can be in the worst case linear in its size – as for the Boolean circuits.

For all $\circ \in \{(⅋^n)_{n\geqslant 2}, (\otimes^n)_{n\geqslant 2}\}$, $\circ$ may be $\bullet$ in $\rightarrow_m$.

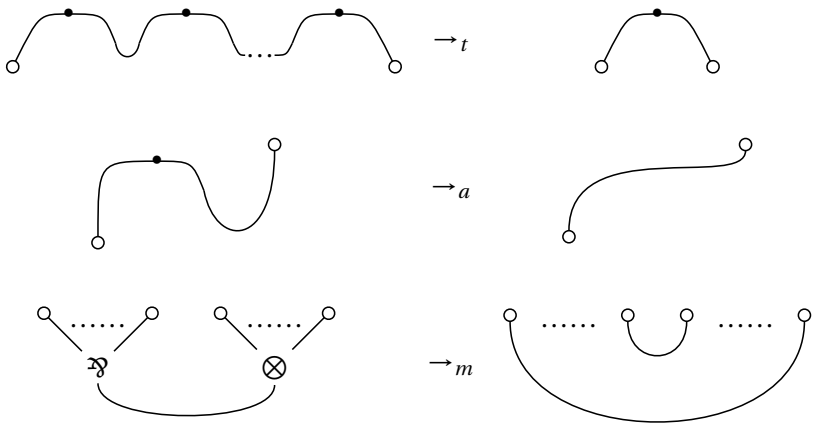


Figure 3: t-, a- and m-reductions

4 Boolean proof nets

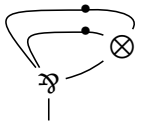
In order to compare the complexities of proof nets and of Boolean circuits, we need to define how proof nets represent Boolean values (definition 12) and Boolean functions (definition 13). To study them in a uniform framework we define their Direct Connection Language (definition 14).

Definition 12 (Boolean type, 0 and 1 [8])**.** Let b_0 and b_1 be the two proof nets of type

$$\mathbf{B} = ⅋^3(\alpha^\perp, \alpha^\perp, \alpha \otimes \alpha)$$

respectively used to represent false and true:

$$\mathscr{D}(b_0) = par_s^{q,p,r}(tensor_r^{p,q}(ax_p, ax_q)) \quad b_0 \equiv$$



$$\mathscr{D}(b_1) = par_s^{p,q,r}(tensor_r^{p,q}(ax_p, ax_q)) \quad b_1 \equiv$$

We write $\overrightarrow{b}$ for $b_{i_1}, \dots, b_{i_n}$ for $i \in \{0,1\}$.

As we can see, b_0 and b_1 differ on their planarity: descriptions and proof nets exhibit the exchanges that were kept implicit in derivations.

Definition 13 (Boolean proof nets [8])**.** A *Boolean proof net with n inputs* is a proof net $P(\overrightarrow{p})$ of type

$$\vdash p_1 : \mathbf{B}^\perp[A_1], \dots, p_n : \mathbf{B}^\perp[A_n], s : \otimes^{1+m}(\mathbf{B}[A], D_1, \dots, D_m)$$

Given $\overrightarrow{b}$ of length n, $P(\overrightarrow{b})$ is obtained by connecting with cuts p_j to b_{i_j} for all $1 \leq j \leq n$.

$P(\overrightarrow{b}) \rightrightarrows^* Q$ where Q is unique, cut-free and for some descriptions $Q_1, \dots, Q_n$ described by

$$tensor(\mathscr{D}(b_i), Q_1, \dots, Q_m) \text{ for } i \in \{0,1\}.$$

We write $P(\overrightarrow{b}) \to_{ev.} b_i$.

$P(\overrightarrow{p})$ *represents a Boolean function* f^n if for all $w \equiv i_1 \dots i_n \in \{0,1\}^n$, $P(b_{i_1}, \dots, b_{i_n}) \to_{ev.} b_{f(w)}$.

We may easily define *families of Boolean proof nets* and *language accepted by a family of Boolean proof nets*.

The tensor indexed with s in the type is the *result tensor*: it collects the result of the computation on its first auxiliary port and the *garbage* – here named $D_1, \dots, D_m$ – on its other auxiliary ports.

Definition 14 (Direct Connection Language for proof nets [7])**.** Given $P = (P_n)_{n\in\mathbb{N}}$ a family of Boolean proof nets, its *Direct Connection Language* – written $L_{DC}(P)$ – is the set of tuples $< y, \overline{g}, \overline{p}, \overline{b} >$ where for $|y| = n$: g is a link in P_n, of sort b if $p = \epsilon$ else the p^{th} premise of g is the link b.

If $< y, \overline{p}, 0, \overline{b} >$ or $< y, \overline{b}, 0, \overline{p} >$ belong to $L_{DC}(P)$, there is a cut between b and p in $C_{|y|}$.

5 Proof circuits

A proof circuit is a Boolean proof net (fact 1) made out of pieces (definition 15) which represents Boolean functions, constants or duplicates values. Garbage is manipulated in an innovative way, examples 1 should help to understand the mechanism of computation.

From now on every edge represented by  is connected on its right to an auxiliary port numbered with an integer other than 1 of the result tensor: it carries a piece of garbage.

Definition 15 (Pieces)**.** We present in the table 1 the set of *pieces* at our disposal. Entries are labeled with e, exits with s and garbage with g. Edges labeled b_k – for $k \in \{0,1\}$ – are connected to the edge labeled s of the piece b_k.

Table 1: Pieces

We set $2 \leqslant j \leqslant i$.

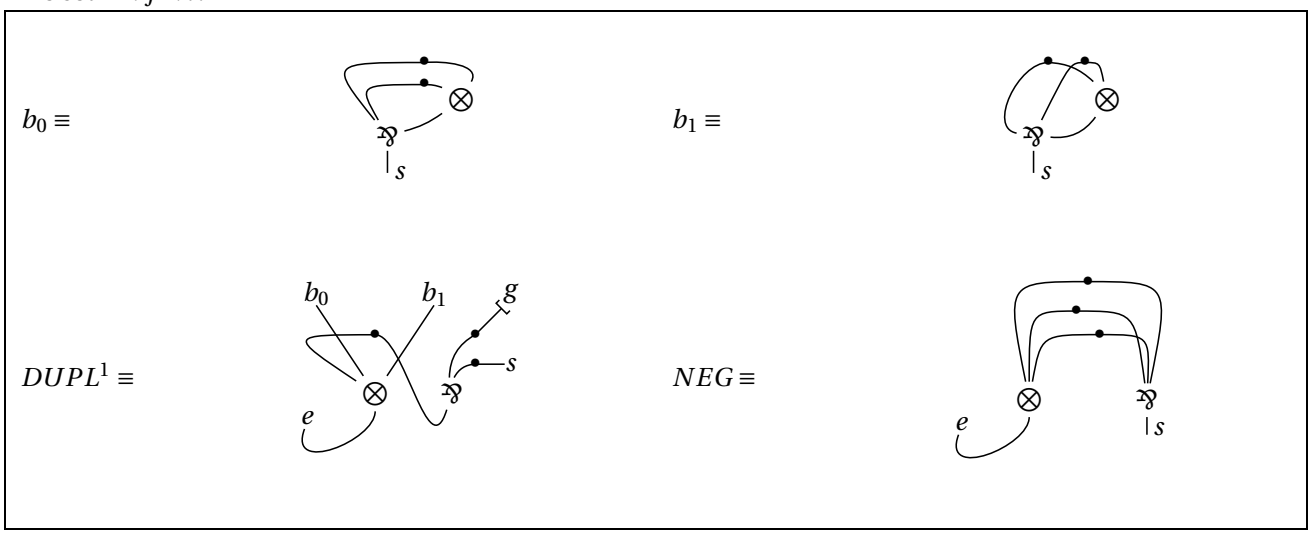
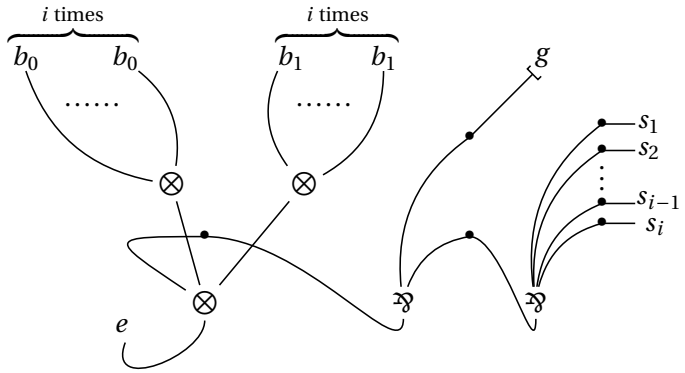


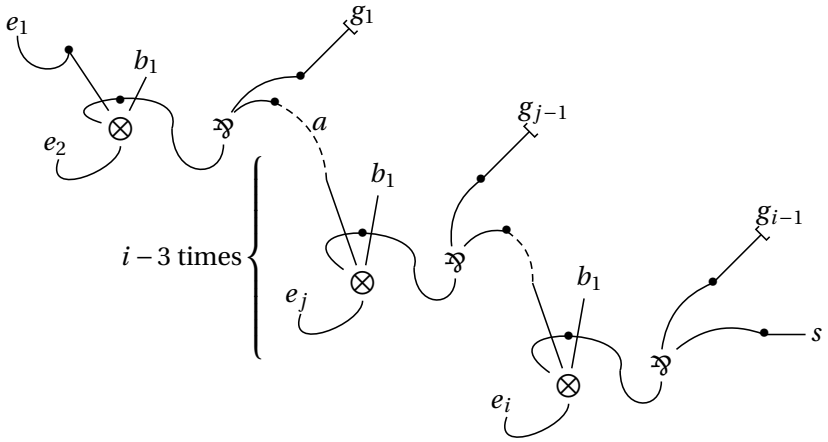
Table 1: Pieces – continued from previous page

$DUPL^i \equiv$

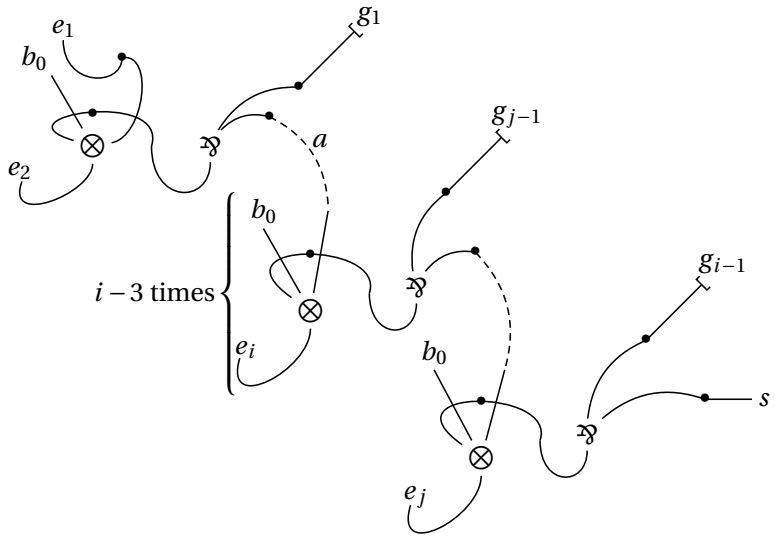


If $i = 2$, the edge a *is* the edge s.

$DISJ^i \equiv$



$CONJ^i \equiv$



A *piece* $\mathscr{P}$ *with* $i \geqslant 0$ *entries,* $j \geqslant 1$ *exits and* $k \geqslant 0$ *garbage* is one of the piece in the table 1, where i edges are labeled with $e_1, \dots, e_i$, j edges are labeled with $s_1, \dots, s_j$ and k edges go to the result tensor.

We have $\mathscr{P} \in \{b_0, b_1, NEG, \{DUPL^i\}_{i\geqslant 1}, \{DISJ^i\}_{i\geqslant 2}, \{CONJ^i\}_{i\geqslant 2}\}$.

To *compose two pieces* $\mathscr{P}_1$ and $\mathscr{P}_2$ we connect an exit of $\mathscr{P}_1$ to an entry of $\mathscr{P}_2$. It is not allowed to loop: we can not connect an entry and an exit belonging to the same piece.

An entry (resp. an exit) that is not connected to an exit (resp. an entry) of another piece is said to be *unconnected*.

Definition 16 (Proof circuits)**.** A *proof circuit* $\mathscr{C}_n(\overrightarrow{p})$ *with* n *inputs and one output* is obtained by composing pieces such that n entries and one exit are unconnected. If no garbage is created we add a $DUPL^1$-piece connected to the unconnected exit to produce some artificially. Then we add a result tensor whose first edge is connected to the exit – which is also the output of the proof circuit – and the others to the garbage. We then label every unconnected entries with $p_1, \dots, p_n$: those are the inputs of the proof circuit.

Given $\overrightarrow{b}$ of length n, $\mathscr{C}_n(\overrightarrow{b})$ is obtained by connecting with cuts p_j to b_{i_j} for all $1 \leq j \leq n$.

Fact 1. *Every proof circuit is a Boolean proof net.*

Proof. We prove this fact with a contractibility criterion [2], by induction on the height of the pieces of the proof circuit (counted as the number of pieces from the considered piece to the result tensor). As a proof circuit can always be typed with

$$\vdash \mathbf{B}^{\perp}[A_1], \dots, \mathbf{B}^{\perp}[A_n], \otimes^{1+m}(\mathbf{B}[A], D_1, \dots, D_m)$$

it is a Boolean proof net.

This fact establishes that proof circuits normalize and output a value, and that it is possible to represent Boolean functions with them.

Families of proof circuits and *acceptation of a language by a family of proof circuits* are defined as usual.

Examples 1. We present briefly two examples of computation: the normalization of a piece b_0 connected to a NEG-piece (figure 4, page 24) and how the conditional works (figure 5, page 25). Conditional is the core of the computation, we find this pattern in every piece except NEG and the constants.

Remark 3. To *compose two proof circuits* $\mathscr{C}_1$ and $\mathscr{C}_2$, we remove the result tensor of $\mathscr{C}_1$, identify the unconnected exit of $\mathscr{C}_1$ with the selected input of $\mathscr{C}_2$, and recollect all the garbage with the result tensor of $\mathscr{C}_2$. We then label the unconnected entries anew and obtain a proof circuit.

Definition 17 (PCC^i (resp. mBN^i, [7]))**.** A language $X \subseteq \{0,1\}^*$ belongs to the class PCC^i (resp. mBN^i) if X is accepted by a polynomial-size, $\log^i$-depth uniform family of proof circuits (resp. of Boolean proof nets).

If we add "$UstConn_2$*-pieces*" to the set of pieces, we may easily define $PCC^i(UstConn_2)$ and remark that for all $i \in \mathbb{N}$, $PCC^i \subseteq PCC^i(UstConn_2)$. [8] proves that there exists Boolean proof nets of constant-depth and polynomial size that represent $UstConn_2$, so we do not define $mBN^i(UstConn_2)$ because it would be easy to prove that this class is equal to mBN^i for all $i \in \mathbb{N}$.

Remark 4. By fact 1 we have trivially that for all $i \in \mathbb{N}$, $PCC^i \subseteq mBN^i$.

Lemma 1. *For all proof circuit* $\mathscr{C}_n(\overrightarrow{p})$ *and all* $\overrightarrow{b}$*, the cuts at maximum depth in* $\mathscr{C}_n(\overrightarrow{b})$ *are between the entry of a piece and a value (a constant* b_0 *or* b_1*, or an input* b_{i_j} *for some* $1 \leq j \leq n$*).*

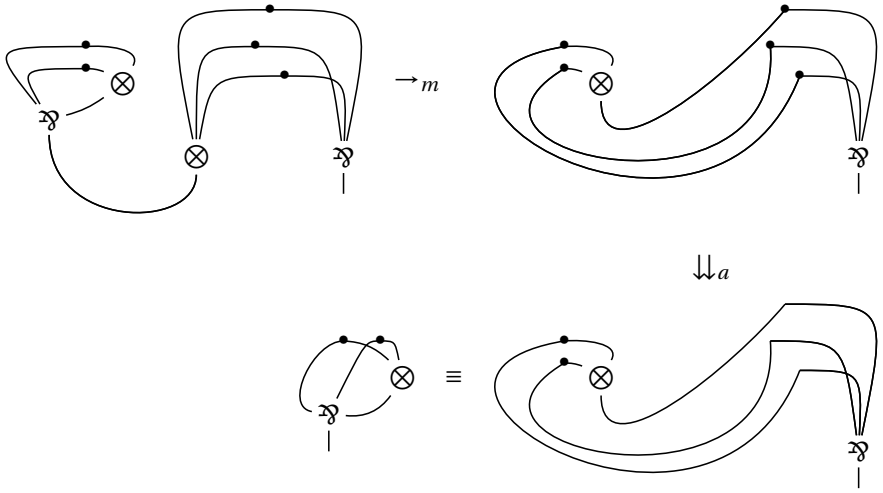


Figure 4: b_0 connected to NEG normalizes to b_1

Proof. For every piece $\mathscr{P}$ of $\mathscr{C}_n(\overrightarrow{b})$ any cut connecting an entry is always of depth superior or equal to the maximal depth of the cuts connecting the exits. The cuts of $\mathscr{P}$ that do not connect an entry or an exit of a piece are always of depth inferior or equal to cuts connecting the entries.

The depths of the cut formulae slowly increase from the exit to the entry, and as the entries that are not connected to other pieces are connected to values, this lemma is proved.

6 Results

By using our proof circuits we prove anew the inclusions between AC^i and logical classes of complexity and extend this inclusion to sublogarithmic classes of complexity.

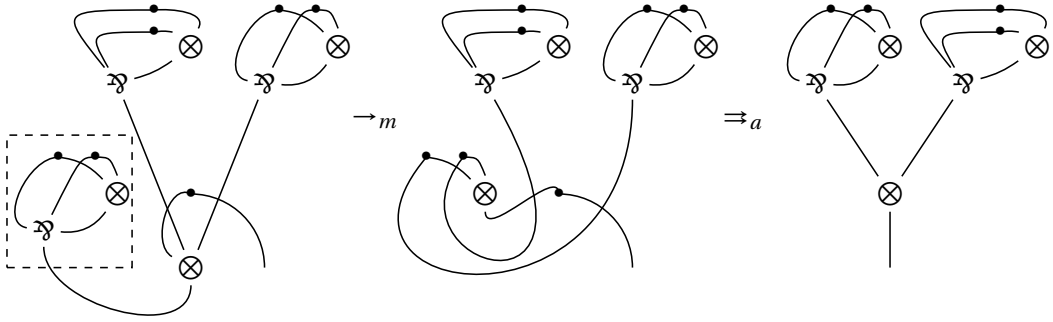
Definition 18 (Problem: Translation from AC^i to PCC^i)**.**

Input:	$L_{DC}(C)$ for C a family of Boolean circuits in AC^i.
Output:	$L_{DC}(\mathscr{C})$ for $\mathscr{C}$ a family of proof circuits in PCC^i, such that for all $n \in \mathbb{N}$, for all $\overrightarrow{b} \equiv b_{i_1}, \dots, b_{i_n}$, $\mathscr{C}_n(\overrightarrow{b}) \to_{ev.} b_j$ iff $C_n(i_1, \dots, i_n)$ evaluates to j.

Theorem 2. *For all $i \in \mathbb{N}$, translation from AC^i to PCC^i belongs to AC^0.*

Proof. The translation from C to $\mathscr{C}$ is obvious, it relies on coding: for every n, a first constant-depth circuit associate to every gate of C_n the corresponding piece simulating its Boolean function. If the fan-out of this gate is $k > 1$, a $DUPL^k$-piece is associated to the exit of the piece, and the pieces are connected as the gates. The input nodes are associated to the inputs of $\mathscr{C}_n$. A second constant-depth circuit recollects the only free exit and the garbage of the pieces and connects them to the result tensor. The composition of these two Boolean circuits produces a constant-depth Boolean circuit that builds proof circuits.

It is easy to check that $CONJ^k$, $DISJ^k$ and NEG represent $\wedge^k$, $\vee^k$ and $\neg$ respectively. $DUPL^k$ duplicates a value k times, b_0 and b_1 represent 0 and 1 by convention. The composition of these pieces does not raise any trouble: $\mathscr{C}_n$ effectively simulates C_n on every input of size n.



The input (here in a dashed rectangle) is proof net of type **B**, and it "selects" – according to its planarity or non-planarity– during the normalization which one of b_0 or b_1 is connected to the first auxiliary port of the tensor and so is considered as the result – the other being treated as garbage.

Figure 5: The conditional, the core of the computation

Concerning the bounds: the longest path between an entry or a constant and the result tensor goes through at most $2 \times d(C_n)$ pieces and we know by lemma 1 that the increase of the depth is linear in the number of pieces crossed. We conclude that $d(\mathscr{C}_n) \leqslant 2 \times 3 \times d(C_n)$ and that $\mathscr{C}_n$ normalizes in $O(d(C_n))$ parallel steps.

Concerning the size, by counting we know that a gate of fan-in n and fan-out m is simulated by a piece made of $O(m+n)$ links. As the number of edges in C_n is bounded by $|C_n|^2$, the size of $\mathscr{C}_n$ is at most $O(|C_n|^2)$.

A Boolean circuit with unbounded (resp. bounded) arity of size s is translated by a Proof circuit of size quadratic (resp. linear) in s, whereas [8] considers only unbounded Boolean circuits and translate them with Boolean proof nets of size $O(s^5)$. Our translation – thanks mostly to the easier garbage collection – needs less computational power, is more clear and besides lower the size of the Boolean proof nets obtained.

Of course, we could naturally extend this translation to a translation from $AC^i(\mathit{UstConn}_2)$ to $PCC^i(\mathit{UstConn}_2)$ and still remain in AC^0. But it is of little interest to look for a sublogarithmic translation toward a class of complexity which is probably not strictly included in L.

Fact 2. *As the reduction from C to $\mathscr{C}$ is in AC^0, we know that this reduction is correct for Boolean circuit families in AC^0 and that every $\mathscr{C}$ obtained by this translation is uniform.*

This result brings a novelty in the study of the proof nets as a class of complexity, making them able to simulate very small classes of complexity born from the Boolean circuits.

Theorem 3 (Simulation)**.** *For all $i \in \mathbb{N}$, for all Boolean proof net family $P = (P_n)_{n\in\mathbb{N}}$ in mBN^i, there exists a family of Boolean circuits $C = (C_n)_{n\in\mathbb{N}}$ in $AC^i(\mathit{UstConn}_2)$ and a constant-depth circuit in AC^0 that given $L_{DC}(P)$ outputs $L_{DC}(C)$ such that for all $\overrightarrow{b} \equiv b_{i_1} \dots b_{i_n}$, $P_n(\overrightarrow{b}) \to_{ev.} b_j$ iff $C_n(i_1, \dots, i_n)$ evaluates to j.*

Proof. We know thanks to [8] that for $r \in \{a, m, t\}$ an unbounded fan-in constant-depth circuit with $O(|P_n|^3)$ gates – with $\mathit{UstConn}_2$ gates to identify chains of axioms if $r = t$– is able to reduce all the r-cuts of P_n in parallel.

A first constant-depth circuit establishes the configuration – which describes P_n – from $L_{DC}(P)$ and constant-depth circuits update this configuration after steps of normalization. Once the configuration of the normal form of P_n is obtained, a last constant-depth circuit identifies the first proof

net connected to the result tensor and establishes if it is b_0 or b_1 – that is to say if the result of the evaluation is *false* or *true*.

As all the circuits are of constant depth, the depth of C_n is linear in $d(P_n)$. The size of C_n is $O(|P_n|^4)$: every circuit simulating a parallel reduction needs $O(|P_n|^3)$ gates and in the worst case – if $d(P_n)$ is linear in the size of the proof circuit – $O(|P_n|)$ steps are needed to normalize the proof net.

The remark 4 helps us to conclude that $PCC^i \subseteq mBN^i \subseteq AC^i(UstConn_2)$

The simulation is slightly different from the translation: the Boolean circuit does not have to identify the pieces or any mechanism of computation of P_n, but simply to apply $\rightrightarrows_t$, $\rightrightarrows_a$, $\rightrightarrows_m$ to it until it reaches a normal form and then look at the value obtained. This simulation can be applied to any Boolean proof net, but in order to have results concerning complexity we preferred to stay in this uniform framework.

Theorem 4. *For all $i \in \mathbb{N}$, $AC^i \subseteq PCC^i \subseteq AC^i(UstConn_2)$.*

Proof. By theorem 2 and theorem 3. The key point is to notice – as the reductions are in AC^0– that $AC^0 \subseteq PCC^0 \subseteq AC^0(UstConn_2)$.

Remark 5. We focused on sublogarithmic classes of complexity, but we can draw more general conclusions by re-introducing $UstConn_2$. From what precedes and from the fact that $UstConn_2$ may be represented by Boolean proof nets of constant-depth and polynomial size, it is easy to conclude that for all $i \in \mathbb{N}$, $PCC^i(UstConn_2) = AC^i(UstConn_2) = mBN^i$.

Conclusion

By restricting ourselves to the uniform complexity classes and by lightening the simulation of the Boolean functions by proof nets, we established the validity of results given by [8] and [6] when extended to constant-depth Boolean circuits. Those complexity classes are of great interest as they are below L and mostly used in reductions. This paper proves that proof nets for Multiplicative Linear Logic are a pertinent tool to study complexity classes, including very small ones, even if we do not use exponentials.

The simulation of the parallel elimination of t-cuts by Boolean circuits needs $UstConn_2$ gates. But as $UstConn_2 \in L$, there is for the time being no clue if a sublogarithmic Boolean circuit can simulate Boolean proof nets: $AC^0(UstConn_2) \subseteq AC^1 \supseteq L$.

Our future work will aim to prove that proof nets are a model of computation as relevant as Alternating Turing Machines but easier to manipulate: as we are in an implicit complexity framework, the size of our object suffices to know in which class of complexity it rests, whereas the only way of knowing where is an ATM is to run it on inputs. We already have gateways – by using correspondences with Boolean circuits – between Boolean proof nets and ATM, but our objective is to establish direct proofs.

References

[1] David A. Barrington, Neil Immerman & Howard Straubing (1990): *On uniformity within NC1*. *Journal of Computer and System Sciences* 41(3), pp. 274–306, doi:10.1109/SCT.1988.5262.

[2] Vincent Danos (1990): *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul)*. *These de doctorat, Université Paris VII*.

[3] Vincent Danos & Laurent Regnier (1989): *The structure of multiplicatives*. *Archive for Mathematical logic* 28(3), pp. 181–203, doi:10.1007/BF01622878.

[4] Jean-Yves Girard (1996): *Proof-nets: The parallel syntax for proof-theory*. *Logic and Algebra* 180, pp. 97–124.

[5] William Hesse, Eric Allender & David A. Barrington (2002): *Uniform constant-depth threshold circuits for division and iterated multiplication*. *Journal of Computer and System Sciences* 65(4), pp. 695–716, doi:10.1016/S0022-0000(02)00025-9. Available at `http://ftp.cs.rutgers.edu/pub/allender/division.pdf`.

[6] Virgile Mogbil (2009): *Non-deterministic Boolean Proof Nets*. In: *Proceedings of FOPARA'09*, *Lecture Notes in Computer Science* 6324, Springer, pp. 131–145, doi:10.1007/978-3-642-15331-0_9. Available at `http://hal.archives-ouvertes.fr/docs/00/44/39/25/PDF/nBPN_preprintLIPN09.pdf`.

[7] Virgile Mogbil & Vincent Rahli (2007): *Uniform circuits, & Boolean proof nets*. In: *Proceedings of LFCS'07*, *Lecture Notes in Computer Science* 4514, Springer, pp. 401–421, doi:10.1007/978-3-540-72734-7_28. Available at `http://hal.archives-ouvertes.fr/docs/00/14/39/28/PDF/mwBN_preprintLIPN07.pdf`.

[8] Kazushige Terui (2004): *Proof Nets and Boolean Circuits*. In: *Proceedings of LICS'04*, pp. 182–191, doi:10.1109/LICS.2004.1319612. Available at `http://www.kurims.kyoto-u.ac.jp/~terui/pn.pdf`.

[9] Heribert Vollmer (1999): *Introduction to Circuit Complexity: A Uniform Approach*. Springer Verlag.

Unification and Logarithmic Space*

Clément Aubert and Marc Bagnol

Aix-Marseille Université, CNRS, I2M, UMR 7373, 13453 Marseille, France

Abstract. We present an algebraic characterization of the complexity classes Logspace and NLogspace, using an algebra with a composition law based on unification. This new bridge between unification and complexity classes is inspired from proof theory and more specifically linear logic and Geometry of Interaction.

We show how unification can be used to build a model of computation by means of specific subalgebras associated to finite permutation groups.

We then prove that whether an observation (the algebraic counterpart of a program) accepts a word can be decided within logarithmic space. We also show that the construction can naturally encode pointer machines, an intuitive way of understanding logarithmic space computing.

Keywords: Implicit Complexity, Unification, Logarithmic Space, Proof Theory, Pointer Machines, Geometry of Interaction.

Introduction

Proof Theory and Complexity Theory. There is a longstanding tradition of relating proof theory (more specifically linear logic [1]) and implicit complexity theory that dates back to the introduction of bounded [2] and light [3] logics. Control over the modalities [4,5], type assignment [6] and stratification of exponential boxes [7], to name a few, led to a clearer understanding of the complexity bounds linear logic could entail on the cut-elimination procedure.

We propose to push further this approach by adopting a more semantical and algebraic point of view that will allow us to capture non-deterministic logarithmic space computation.

Geometry of Interaction. As the study of cut-elimination has grown as a central topic in proof theory, its mathematical modeling became of great interest. The Geometry of Interaction [8] research program led to mathematical models of cut-elimination in terms of paths in proofnets [9], token machines [10] and operator algebras [11]. Related complexity concerns have already been explored [12,13].

Recent works [13,14,15] studied the link between Geometry of Interaction and logarithmic space, relying on the theory of von Neumann algebras. Those three articles are indubitably sources of inspiration of this work, but the whole construction is made anew, in a simpler framework.

* This work was partly supported by the ANR-10-BLAN-0213 Logoi and the ANR-11-BS02-0010 Récré.

G. Dowek (ed.): RTA-TLCA 2014, LNCS 8560, pp. 77–92, 2014.

Unification. Unification is one of the key-concepts of theoretical computer science, for it is used in logic programming and is a classical subject of study for complexity theory. It was shown [16,17] that one can model cut-elimination with unification techniques.

Execution will be expressed in terms of matching in a *unification algebra*. This is a simple framework, yet expressive enough to encode the action of finite permutation groups on an unbounded tensor product, which is a crucial ingredient of our construction.

Contribution. We carry on the methodology of bridging Geometry of Interaction and complexity theory with a renewed approach. It relies on an simpler representation of execution in a unification-based algebra, proved to capture exactly logarithmic space complexity.

While the representation of inputs (words over a finite alphabet) comes from the classical Church representation, observations (the algebraic counterpart of programs) are shown to correspond very naturally to a notion of pointer machines. This correspondence allows us to prove that reversibility (of machines) is related to the algebraic notion of isometricity (of observations).

Organization of This Article. In Sect.1 we review some classical results on unification of first-order terms and use them to build the algebra that will constitute our computational setting.

We explain in Sect.2 how words and computing devices (observations) can be modeled by particular elements of this algebra. The way they interact to yield a notion of language recognized by an observation is described in Sect.3.

Finally, we show in Sect.4 that our construction captures exactly logarithmic space computation, both deterministic and non-deterministic.

1 The Unification Algebra

1.1 Unification

Unification can be generally thought of as the study of formal solving of equations between terms.

This topic was introduced by Herbrand, but became really widespread after the work of J. A. Robinson on automated theorem proving. The unification technique is also at the core of the logic programming language Prolog and type inference for functional programming languages such as CaML and Haskell.

Specifically, we will be interested in the following problem:

Given two terms, can they be "made equal" by replacing their variables?

Definition 1 ***(terms)***
We consider the following set of first-order terms

$$\mathtt{T} ::= \quad x, y, z, \ldots \mid \mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots \mid \mathtt{T} \bullet \mathtt{T}$$

where $x, y, z, \cdots \in \mathtt{V}$ *are variables,* $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots$ *are constants and* $\bullet$ *is a binary function symbol.*

For any $t \in \mathtt{T}$, we will write $\mathtt{Var}(t)$ the set of variables occurring in t. We say that a term is closed when $\mathtt{Var}(t) = \varnothing$, and denote $\mathtt{T_c}$ the set of closed terms.

Notation. *The binary function symbol $\bullet$ is not associative, but we will write it by convention as* right associating *to lighten notations: $t \bullet u \bullet v := t \bullet (u \bullet v)$*

Definition 2 ***(substitution)***
A substitution *is a map $\theta : \mathtt{V} \to \mathtt{T}$ such that the set $\mathtt{Dom}(\theta) := \{v \in \mathtt{V} | \theta(v) \neq v\}$ (the* domain *of θ) is finite. A substitution with domain $\{ x_1, \dots, x_n \}$ such that $\theta(x_1) = u_1$, …, $\theta(x_n) = u_n$ will be written as $\{ x_1 \mapsto u_1 ; \dots ; x_n \mapsto u_n \}$.*

If $t \in \mathtt{T}$ is a term we write $t.\theta$ the term t where any occurrence of any variable x has been replaced by $\theta(x)$.

If $\theta = \{ x_i \mapsto u_i \}$ and $\psi = \{ y_j \mapsto v_j \}$, their composition *is defined as*

$$\theta ; \psi := \{ x_i \mapsto u_i.\psi \} \cup \{ y_j \mapsto v_j \mid y_j \notin \mathtt{Dom}(\theta) \}$$

Remark. *The composition of substitutions is such that $t.(\theta ; \psi) = (t.\theta).\psi$ holds.*

Definition 3 ***(renamings and instances)***
A renaming *is a substitution α such that $\alpha(\mathtt{V}) \subseteq \mathtt{V}$ and that is bijective. A term t' is a* renaming *of t if $t' = t.\alpha$ for some renaming α.*

Two substitutions θ, ψ are equal up to renaming *if there is a renaming α such that $\psi = \theta ; \alpha$.*

A substitution ψ is an instance *of θ if there is a substitution σ such that $\psi = \theta ; \sigma$.*

Proposition 4
Let θ, ψ be two substitutions. If θ is an instance of ψ and ψ is an instance of θ, then they are equal up to renaming.

Definition 5 ***(unification)***
Two terms t, u are unifiable *if there is a substitution θ such that $t.\theta = u.\theta$.*

We say that θ is a most general unifier (MGU) of *t, u if any other unifier of t, u is an instance of θ.*

Remark. *It follows from Proposition 4 that any two MGU of a pair of terms are equal up to renaming.*

We will be interested mostly in the weaker variant of unification where one can first perform renamings on terms so that their variables are distinct, we introduce therefore a specific vocabulary for it.

Definition 6 ***(disjointness and matching)***
Two terms t, u are matchable *if t', u' are unifiable, where t', u' are renamings (Definition 3) of t, u such that $\mathtt{Var}(t') \cap \mathtt{Var}(u') = \varnothing$.*
If two terms are not matchable, they are said to be disjoint*.*

Example. *x and $\mathtt{f}\bullet x$ are not unifiable.*
But they are matchable, as $x.\{ x \mapsto y\,;\, y \mapsto x \} = y$ which is unifiable with $\mathtt{f}\bullet x$.
More generally, disjointness is stronger than non-unifiability.

The crucial feature of first-order unification is the (decidable) existence of most general unifiers for unification problems that have a solution.

Proposition 7 ***(MGU)***
If two terms are unifiable, then they have a MGU.
Whether two terms are unifiable and, in case they are, finding a MGU is decidable.

As unification grew in importance, the study of its complexity gained in attention. A complete survey [18] tells the story of the bounds getting sharpened: general first-order unification was finally proved [19] to be a Ptime-complete problem.

In this article, we are concerned with a very much simpler case of the problem: the matching (Definition 6) of linear terms (*ie.* where variables occur at most once). This case can be solved in a space-efficient way.

Proposition 8 ***(matching in logarithmic space [20, Lemma 20])***
Whether two linear terms t, u with disjoint sets of variables are unifiable, and if so finding a MGU, can be computed in logarithmic space in the size[1] *of t, u on a deterministic Turing machine*

The lemma in [20] actually states that the problem is in NC^1, a complexity class of parallel computation known to be included in Logspace.
We will use only a special case of the result, matching a linear term against a closed term.

1.2 Flows and Wirings

We now use the notions we just saw to build an algebra with a product based on unification. Let us start with a monoid with a partially defined product, which will be the basis of the construction.

Definition 9 ***(flows)***
A flow *is an oriented pair written $t \leftharpoondown u$ with $t, u \in \mathtt{T}$ such that $\mathtt{Var}(t) = \mathtt{Var}(u)$.*
Flows are considered up to renaming: for any renaming α, $t \leftharpoondown u = t.\alpha \leftharpoondown u.\alpha$.
We will write $\mathcal{F}$ the set of (equivalence classes of) flows.
We set $I := x \leftharpoondown x$ and $(t \leftharpoondown u)^\dagger := u \leftharpoondown t$ so that $(.)^\dagger$ is an involution of $\mathcal{F}$.

A flow $t \leftharpoondown u$ can be thought of as a `'match ... with u -> t'` in a ML-style language. The composition of flows follows this intuition.

[1] The *size* of a term is the total number of occurrences of symbols in it.

Definition 10 ***(product of flows)***
Let $u \leftharpoonup v \in \mathcal{F}$ *and* $t \leftharpoonup w \in \mathcal{F}$*. Suppose we have chosen two representatives of the renaming classes such that their sets of variables are disjoint.*
The product *of* $u \leftharpoonup v$ *and* $t \leftharpoonup w$ *is defined if* v, t *are unifiable with MGU* θ *(the choice of a MGU does not matter because of the remark following Definition 5) and in that case:* $(u \leftharpoonup v)(t \leftharpoonup w) := u.\theta \leftharpoonup w.\theta$*.*

Definition 11 ***(action on closed terms)***
If $t \in \mathtt{T_c}$ *is a closed term,* $(u \leftharpoonup v)(t)$ *is defined whenever* t *and* v *are unifiable, with MGU* θ*, in that case* $(u \leftharpoonup v)(t) := u.\theta$

Examples. *Composition of flows:* $(x\bullet\mathtt{c} \leftharpoonup (\mathtt{c}\bullet\mathtt{c})\bullet x)(y\bullet z \leftharpoonup z\bullet y) = x\bullet\mathtt{c} \leftharpoonup x\bullet\mathtt{c}\bullet\mathtt{c}$*.*
Action on a closed term: $(x\bullet\mathtt{c} \leftharpoonup x\bullet\mathtt{c}\bullet\mathtt{c})(\mathtt{d}\bullet\mathtt{c}\bullet\mathtt{c}) = \mathtt{d}\bullet\mathtt{c}$*.*

Remark. *The condition on variables ensures that the result is a closed term (because* $\mathtt{Var}(u) \subseteq \mathtt{Var}(v)$*) and that the action is injective on its domain of definition (because* $\mathtt{Var}(v) \subseteq \mathtt{Var}(u)$*). Moreover, the action is compatible with the product of flows:* $l(k(t)) = (l\,k)(t)$ *and both are defined at the same time.*

By adding a formal element $\perp$ (representing the failure of unification) to the set of flows, one could turn the product into a completely defined operation, making $\mathcal{F}$ an *inverse monoid*. However, we will need to consider the wider algebra of *sums* of flows that is easily defined directly from the partially defined product.

Definition 12 ***(wirings)***
Wirings *are* $\mathbb{C}$*-linear combinations of flows (formally: almost-everywhere null functions from the set of flows* $\mathcal{F}$ *to* $\mathbb{C}$*), endowed with the following operations:*

$$\left(\sum_i \lambda_i\, l_i\right)\left(\sum_j \mu_j\, k_j\right) := \sum_{\substack{i,j \text{ such that} \\ (l_i k_j) \text{ is defined}}} \lambda_i\mu_j (l_i\, k_j) \qquad \textit{(with } \lambda_i,\mu_j \in \mathbb{C} \textit{ and } l_i,k_j \in \mathcal{F}\textit{)}$$

$$\textit{and} \qquad \left(\sum_i \lambda_i\, l_i\right)^\dagger := \sum_i \overline{\lambda}_i\, l_i^\dagger \qquad \textit{(where } \overline{\lambda} \textit{ is the complex conjugate of } \lambda\textit{)}$$

We write $\mathcal{U}$ *the set of wirings and refer to it as the* unification algebra*.*

Remark. *Indeed,* $\mathcal{U}$ *is a unital* $*$*-algebra: it is a* $\mathbb{C}$*-algebra (considering the product defined above) with an involution* $(.)^\dagger$ *and a unit* I*.*

Definition 13 ***(partial isometries)***
A partial isometry *is a wiring* $U \in \mathcal{U}$ *satisfying* $UU^\dagger U = U$*.*

Example. $(\mathtt{c}\bullet x \leftharpoonup x\bullet\mathtt{d}) + (\mathtt{d}\bullet\mathtt{c} \leftharpoonup \mathtt{c}\bullet\mathtt{c})$ *is a partial isometry.*

While $\mathcal{U}$ offers the general algebraic background to work in, we will need to consider a particular kind of wirings to study computation.

Definition 14 ***(concrete and isometric wirings)***
A wiring is concrete *whenever it is a sum of flows with all coefficients equal to* 1*.*

An isometric wiring *is a concrete wiring that is also a partial isometry.*
Given a set of wirings E *we write* E^+ *for the set of all concrete wirings of* E *.*

Isometric wirings enjoy a direct characterization.

Proposition 15 ***(isometric wirings)***
The isometric wirings are exactly the wirings of the form $\sum_i u_i \leftharpoondown t_i$ *with the* u_i *pairwise disjoint (Definition 6) and* t_i *pairwise disjoint.*

It will be useful to consider the action of wirings on closed terms. For this purpose we extend Definition 11 to wirings.

Definition 16 ***(action on closed terms)***
Let $\mathbb{V}_{\mathtt{c}}$ *be the free* $\mathbb{C}$*-vector space over* $\mathtt{T_c}$ *.*
Wirings act on base vectors of $\mathbb{V}_{\mathtt{c}}$ *in the following way*

$$\left(\sum_i \lambda_i\, l_i\right)(t) := \sum_{\substack{i \text{ such that} \\ l_i(t) \text{ is defined}}} \lambda_i \big(l_i(t)\big) \quad \in \mathbb{V}_{\mathtt{c}}$$

which extends by linearity into an action on the whole $\mathbb{V}_{\mathtt{c}}$ *.*

Isometric wirings have a particular behavior in terms of this action.

Lemma 17 ***(isometric action)***
Let F *be an isometric wiring and* t *a closed term. We have that* $F(t)$ *and* $F^\dagger(t)$ *are either* 0 *or another closed term* t' *(seen as an element of* $\mathbb{V}_{\mathtt{c}}$*).*

1.3 Tensor Product and Permutations

We define now the representation in $\mathcal{U}$ of structures that provide enough expressivity to model computation.
Unbounded tensor products will allow to represent data of arbitrary size, and finite permutations will be used to manipulate these data.

Notations. *Given any set of wirings or closed terms* E *, we write* $\mathtt{Vect}(E)$ *the vector space generated by* E *,* ie. *the set of* finite *linear combinations of elements of* E *(for instance* $\mathtt{Vect}(\mathtt{T_c}) = \mathbb{V}_{\mathtt{c}}$*).*
Moreover, we set $\mathcal{I} := \{\, \lambda I \mid \lambda \in \mathbb{C} \,\}$ *(with* $I = x \leftharpoondown x$ *as in Definition 9) which is the* $*$*-algebra of multiples of the identity, and* $u \leftrightharpoons v := u \leftharpoondown v + v \leftharpoondown u$ *.*
For brevity we write "$*$*-algebra" instead of the more correct "*$*$*-subalgebra of* $\mathcal{U}$*"* (ie. *a subset of* $\mathcal{U}$ *that is stable by linear combinations, product and* $(.)^\dagger$*).*

Definition 18 ***(tensor product)***
Let $u \leftharpoondown v$ *and* $t \leftharpoondown w$ *be two flows. Suppose we have chosen representatives of these renaming classes that have their sets of variables disjoint. We define their* tensor product *as* $(u \leftharpoondown v) \dot{\otimes} (t \leftharpoondown w) := u_\bullet t \leftharpoondown v_\bullet w$ *. The operation is extended to wirings by bilinearity.*
Given two $*$*-algebras* $\mathcal{A}, \mathcal{B}$ *, we define their tensor product as the* $*$*-algebra*

$$\mathcal{A} \dot{\otimes} \mathcal{B} := \mathtt{Vect}\left\{\, F \dot{\otimes} G \;\middle|\; F \in \mathcal{A},\, G \in \mathcal{B} \,\right\}$$

This actually defines an embedding of the algebraic tensor product of $*$-algebras into $\mathcal{U}$, which means in particular that $(F \dot{\otimes} G)(P \dot{\otimes} Q) = (FP) \dot{\otimes} (GQ)$. It ensures also that the $\dot{\otimes}$ operation indeed yields $*$-algebras.

Notation. *As* $\bullet$*, the* $\dot{\otimes}$ *operation is not associative. We carry on our convention and write it as* right associating*:* $\mathcal{A} \dot{\otimes} \mathcal{B} \dot{\otimes} \mathcal{C} := \mathcal{A} \dot{\otimes} (\mathcal{B} \dot{\otimes} \mathcal{C})$.

Definition 19 ***(unbounded tensor)***
Let $\mathcal{A}$ *be a* $*$*-algebra. We define the* $*$*-algebras* $\mathcal{A}^{\otimes n}$ *for all* $n \in \mathbb{N}$ *as*

$$\mathcal{A}^{\otimes 0} := \mathcal{I} \quad \textit{and} \quad \mathcal{A}^{\otimes n+1} := \mathcal{A} \dot{\otimes} \mathcal{A}^{\otimes n}$$

and the $*$*-algebra* $\mathcal{A}^{\otimes \infty} := \mathtt{Vect}\left(\bigcup_{n \in \mathbb{N}} \mathcal{A}^{\otimes n} \right)$.

We will consider finite permutations, but allow them to be composed even when their domain of definition do not match.

Notations. *Let* $\mathfrak{S}_n$ *be the set of finite permutations over* $\{1, \dots, n\}$*, if* $\sigma \in \mathfrak{S}_n$*, we define* $\sigma_{+k} \in \mathfrak{S}_{n+k}$ *as the permutation* σ *extended to* $\{1, \dots, n, \dots, n+k\}$ *letting* $\sigma_{+k}(n+i) := n+i$ *for* $i \in \{1, \dots, k\}$.
We also write $I_k := Id_{\{1,\dots,k\}} \in \mathfrak{S}_k$.

Definition 20 ***(representation)***
To a permutation $\sigma \in \mathfrak{S}_n$ *we associate the flow*

$$[\sigma] := x_1 \bullet x_2 \bullet \cdots \bullet x_n \bullet y \leftharpoondown x_{\sigma(1)} \bullet x_{\sigma(2)} \bullet \cdots \bullet x_{\sigma(n)} \bullet y$$

A permutation $\sigma \in \mathfrak{S}_n$ will act on the first n components of the unbounded tensor product (Definition 19) by swapping them and leaving the rest unchanged. The wirings $[\sigma]$ internalize this action: in the above definition, the variable y at the end stands for the components that are not affected.

Example. *Let* $\tau \in \mathfrak{S}_2$ *be the permutation swapping the two elements of* $\{1, 2\}$ *and* $U_1 \dot{\otimes} U_2 \dot{\otimes} U_3 \dot{\otimes} I \in \mathcal{U}^{\otimes 3} \subseteq \mathcal{U}^{\otimes \infty}$. *We have* $[\tau] = x_1 \bullet x_2 \bullet y \leftharpoondown x_2 \bullet x_1 \bullet y$ *and* $[\tau](U_1 \dot{\otimes} U_2 \dot{\otimes} U_3 \dot{\otimes} I)[\tau]^\dagger = U_2 \dot{\otimes} U_1 \dot{\otimes} U_3 \dot{\otimes} I$.

Proposition 21 ***(representation)***
For $\sigma \in \mathfrak{S}_n$ *and* $\tau \in \mathfrak{S}_{n+k}$ *we have*

$$[\sigma_{+k}] = [\sigma][I_{n+k}] = [I_{n+k}][\sigma] \qquad [\sigma_{+k} \circ \tau] = [\sigma][\tau] \qquad \textit{and} \qquad [\sigma^{-1}] = [\sigma]^\dagger$$

Definition 22 ***(permutation algebra)***
For $n \in \mathbb{N}$ *we set* $[\mathfrak{S}_n] := \{ [\sigma] \mid \sigma \in \mathfrak{S}_n \}$ *and* $\mathcal{S}_n := \mathtt{Vect}[\mathfrak{S}_n]$.
We define then $\mathcal{S} := \mathtt{Vect}\left(\bigcup_{n \in \mathbb{N}} \mathcal{S}_n \right)$*, which we call the* permutation algebra.

Proposition 21 ensures that the $\mathcal{S}_n$ and $\mathcal{S}$ are $*$-algebras.

2 Words and Observations

The representation of words over an alphabet in the unification algebra directly comes from the translation of Church lists in linear logic and their interpretation in Geometry of Interaction models [11,16].

This proof-theoretic origin is an useful guide for intuition, even if we give here a more straightforward definition of the notion.

From now on, we fix a set of two distinguished constant symbols $\mathtt{LR} := \{\, \mathtt{L}, \mathtt{R} \,\}$.

Definition 23 ***(word algebra)***
To a set S of closed terms, we associate the $*$-algebra

$$S^* := \mathtt{Vect}\,\{\, t \leftharpoondown u \mid t, u \in S \,\}$$

(which is indeed an algebra because unification of closed terms is simply equality)

The word algebra *associated to a finite set of constant symbols Σ is the $*$-algebra defined as*

$$\mathcal{W}_\Sigma := (\mathcal{I} \mathbin{\dot\otimes} \Sigma^* \mathbin{\dot\otimes} \mathtt{LR}^*) \mathbin{\dot\otimes} (\mathtt{T}_\mathtt{c}^*)^{\otimes 1}$$

($\mathtt{T}_\mathtt{c}$ is the set of all closed terms, $\mathcal{I}$ is defined at the beginning of Sect.1.3 $\dot\otimes$ is as in Definition 18 and $(.)^{\otimes 1}$ is the case $n = 1$ of Definition 19)

The words we consider are cyclic, with a begin/end marker $\star$, a reserved constant symbol. For example the word $\mathtt{0010}$ is to be thought of as $\star\mathtt{0010} = \mathtt{10}\star\mathtt{00} = \mathtt{0}\star\mathtt{001} = \cdots$.
We consider therefore that the alphabet Σ always contains the symbol $\star$.

Definition 24 ***(word representation)***
Let $W = \star\mathtt{c}_1 \dots \mathtt{c}_n$ be a word over Σ and $t_0, t_1, \dots, t_n$ be distinct closed terms. The representation *$W(t_0, t_1, \dots, t_n) \in \mathcal{W}_\Sigma^+$ with respect to $t_0, t_1, \dots, t_n$ of W is an isometric wiring (Definition 14), defined as*

$$\begin{aligned} W(t_0, t_1, \dots, t_n) := \quad & x\bullet\star\bullet\mathtt{R}\bullet(t_0\bullet y) \leftrightharpoons x\bullet\mathtt{c}_1\bullet\mathtt{L}\bullet(t_1\bullet y) \\ + \; & x\bullet\mathtt{c}_1\bullet\mathtt{R}\bullet(t_1\bullet y) \leftrightharpoons x\bullet\mathtt{c}_2\bullet\mathtt{L}\bullet(t_2\bullet y) \\ + \; & \cdots \\ + \; & x\bullet\mathtt{c}_n\bullet\mathtt{R}\bullet(t_n\bullet y) \leftrightharpoons x\bullet\star\bullet\mathtt{L}\bullet(t_0\bullet y) \end{aligned}$$

We now define *observations*, programs computing on representations of words. They lie in a particular $*$-algebra based on the representation of permutations presented in Sect.1.3.

Definition 25 ***(observation algebra)***
An observation *over a finite set of symbols Σ is any element of $\mathcal{O}_\Sigma^+$ where $\mathcal{O}_\Sigma := (\mathtt{T}_\mathtt{c}^* \mathbin{\dot\otimes} \Sigma^* \mathbin{\dot\otimes} \mathtt{LR}^*) \mathbin{\dot\otimes} \mathcal{S}$, i.e. a* finite *sum of flows of the form*

$$(s'\bullet\mathtt{c}'\bullet\mathtt{d}' \leftharpoondown s\bullet\mathtt{c}\bullet\mathtt{d}) \mathbin{\dot\otimes} [\sigma]$$

with s, s' closed terms, $\mathtt{c}, \mathtt{c}' \in \Sigma$, $\mathtt{d}, \mathtt{d}' \in \mathtt{LR}$ and σ is a permutation.
Moreover when an observation happens to be an isometric wiring, we will call it an isometric observation.

3 Normativity: Independence from Representations

We are going to define how observations accept and reject words. This needs to be discussed, because there is a potential issue with word representations: an observation is an element of $\mathcal{U}$ and can therefore only interact with *representations* of a word, and there are many possible representation of the same word (in Definition 24, different choices of closed terms lead to different representations). Therefore one has to ensure that acceptance or rejection is independent of the representation, so that the notion makes the intended sense. The termination of computations will correspond to the algebraic notion of *nilpotency*, which we recall here.

Definition 26 ***(nilpotency)***
A wiring F *is* nilpotent *if* $F^n = 0$ *for some* n*.*

Definition 27 ***(automorphism)***
An automorphism *of a* $*$*-algebra* $\mathcal{A}$ *is a linear application* $\varphi : \mathcal{A} \to \mathcal{A}$ *such that for all* $F, G \in \mathcal{A}$*:* $\varphi(FG) = \varphi(F)\varphi(G)$*,* $\varphi(F^\dagger) = \varphi(F)^\dagger$ *and* φ *is injective.*

Example. $\varphi(U_1 \dot{\otimes} U_2) := U_2 \dot{\otimes} U_1$ *induces an automorphism of* $\mathcal{U} \dot{\otimes} \mathcal{U}$*.*

Notation. *If* φ *is an automorphism of* $\mathcal{A}$ *and* ψ *is an automorphism of* $\mathcal{B}$*, we write* $\varphi \dot{\otimes} \psi$ *the automorphism of* $\mathcal{A} \dot{\otimes} \mathcal{B}$ *defined for all* $A \in \mathcal{A}, B \in \mathcal{B}$ *as* $(\varphi \dot{\otimes} \psi)(A \dot{\otimes} B) := \varphi(A) \dot{\otimes} \psi(B)$ *and extended to all* $\mathcal{A} \dot{\otimes} \mathcal{B}$ *by linearity.*

Definition 28 ***(normative pair)***
A pair $(\mathcal{A}, \mathcal{B})$ *of* $*$*-algebras is a* normative pair *whenever any automorphism* φ *of* $\mathcal{A}$ *can be extended into an automorphism* $\overline{\varphi}$ *of the* $*$*-algebra* $\mathcal{E}$ *generated by* $\mathcal{A} \cup \mathcal{B}$*, such that* $\overline{\varphi}(F) = F$ *for any* $F \in \mathcal{B} \subseteq \mathcal{E}$*.*

The two following propositions set the basis for a notion of acceptance/rejection independent of the representation of a word.

Proposition 29 ***(automorphic representations)***
Any two representations $W(t_0, \dots, t_n), W(u_0, \dots, u_n)$ *of a word* W *over* Σ *are automorphic: there exists an automorphism* φ *of* $(\mathtt{T}_\mathtt{c}^*)^{\otimes 1}$ *such that*

$$(Id_{\mathcal{U}} \dot{\otimes} \varphi)\big(W(t_0, \dots, t_n)\big) = W(u_0, \dots, u_n)$$

Proof. Consider a bijection $f : \mathtt{T}_\mathtt{c} \to \mathtt{T}_\mathtt{c}$ such that $f(t_i) = u_i$ for all i. Then set $\varphi(v \bullet x \leftharpoonup w \bullet x) := f(v) \bullet x \leftharpoonup f(w) \bullet x$, extended by linearity. □

Proposition 30 ***(nilpotency and normative pairs)***
Let $(\mathcal{A}, \mathcal{B})$ *be a normative pair and* φ *an automorphism of* $\mathcal{A}$*. Let* $F \in \mathcal{U} \dot{\otimes} \mathcal{A}$*,* $G \in \mathcal{U} \dot{\otimes} \mathcal{B}$ *and let* $\psi := Id_{\mathcal{U}} \dot{\otimes} \varphi$*. Then* GF *is nilpotent if and only if* $G\,\psi(F)$ *is nilpotent.*

Proof. Let $\overline{\varphi}$ be the extension of φ as in Definition 28 and $\overline{\psi} := \mathrm{Id}_{\mathcal{U}} \dot{\otimes} \overline{\varphi}$.
We have for all $n \neq 0$ that $(G\psi(F))^n = (\overline{\psi}(G)\overline{\psi}(F))^n = (\overline{\psi}(GF))^n = \overline{\psi}((GF)^n)$.
By injectivity of $\overline{\psi}$, $(G\psi(F))^n = 0$ if and only if $(GF)^n = 0$. □

Corollary 31 ***(independence)***
If $\left((\mathtt{T}_{\mathtt{c}}^{*})^{\otimes 1}, \mathcal{B}\right)$ *is a normative pair,* W *a word over* Σ *and* $F \in \mathcal{U} \dot{\otimes} \mathcal{B}$*. The product of* F *with the representation of the word,* $FW(t_0, \dots, t_n)$*, is nilpotent for one choice of* $(t_0, \dots, t_n)$ *if and only if it is nilpotent for all choices of* $(t_0, \dots, t_n)$*.*

The basic components of the word and observation algebras we introduced earlier can be shown to form a normative pair.

Theorem 32
The pair $\left((\mathtt{T}_{\mathtt{c}}^{*})^{\otimes 1}, \mathcal{S}\right)$ *is normative.*

Proof (sketch). By simple computations, the set

$$\mathcal{A} := \mathtt{Vect}\left\{ \sigma F \;\middle|\; \sigma \in \mathcal{S} \text{ and } F \in (\mathtt{T}_{\mathtt{c}}^{*})^{\otimes \infty} \right\}$$

can be shown to be a $*$-algebra $\mathcal{E}$, the $*$-algebra generated by $\mathcal{S} \cup (\mathtt{T}_{\mathtt{c}}^{*})^{\otimes 1}$.
As φ is an automorphism of $(\mathtt{T}_{\mathtt{c}}^{*})^{\otimes 1}$, it can be written as $\varphi(G \dot{\otimes} I) = \psi(G) \dot{\otimes} I$ for all G, with ψ an automorphism of $\mathtt{T}_{\mathtt{c}}^{*}$.

We set for $F = F_1 \dot{\otimes} \cdots \dot{\otimes} F_n \dot{\otimes} I \in (\mathtt{T}_{\mathtt{c}}^{*})^{\otimes n}$, $\tilde{\varphi}(F) := \psi(F_1) \dot{\otimes} \cdots \dot{\otimes} \psi(F_n) \dot{\otimes} I$ which extends into an automorphism of $(\mathtt{T}_{\mathtt{c}}^{*})^{\otimes \infty}$ by linearity. Finally, we extend $\tilde{\varphi}$ to $\mathcal{A}$ by $\overline{\varphi}(\sigma F) := \sigma\, \tilde{\varphi}(F)$. It is then easy to check that $\overline{\varphi}$ has the required properties. □

Remark. *Here we sketched a direct proof for brevity, but this can also be shown by involving a little more mathematical structure (actions of permutations on the unbounded tensor and crossed products) which would give a more synthetic proof.*

We can then define the notion of the language recognized by an observation, *via* Corollary 31.

Definition 33 ***(language of an observation)***
Let $\phi \in \mathcal{O}_{\Sigma}^{+}$ *be an observation over* Σ*. The* language recognized by ϕ *is the following set of words over* Σ*:*

$$\mathcal{L}(\phi) := \{\, W \text{ word over } \Sigma \mid \phi\, W(t_0, \dots, t_n) \text{ nilpotent for any } (t_0, \dots, t_n) \,\}$$

4 Wirings and Logarithmic Space

Now that we have defined our framework and showed how observations can compute, we study the complexity of deciding whenever an observation accepts a word (4.1), and how wirings can recognize any language in (N)Logspace (4.2).

4.1 Soundness of Observations

The aim of this subsection is to prove the following theorem:

Theorem 34 ***(space soundness)***
Let $\phi \in \mathcal{O}^+_\Sigma$ be an observation over Σ.

- *$\mathcal{L}(\phi)$ is decidable in non-deterministic logarithmic space.*
- *If ϕ is isometric, then $\mathcal{L}(\phi)$ is decidable in deterministic logarithmic space.*

Actually, the result stands for the complements of these languages, but as coNLogspace = NLogspace by the Immerman-Szelepcsényi theorem, this makes no difference.

The main tool for this purpose is the notion of *computation space*: finite dimensional subspaces of $\mathbb{V}_{\mathtt{c}}$ (Definition 16) on which we will be able to observe the behavior of certain wirings. It can be understood as the place where all the relevant interactions between an observation and a representation of a word take place.

Definition 35 ***(separating space)***
A subspace E of $\mathbb{V}_{\mathtt{c}}$ is separating *for a wiring F whenever $F(E) \subseteq E$ and $F^n(E) = 0$ implies $F^n = 0$.*

Observations are *finite* sums of wirings. We can naturally associate a finite-dimensional vector space to an observation and a finite set of closed terms.

Definition 36 ***(computation space)***
Let $\{ t_0, \dots, t_n \}$ be a set of distinct closed terms and $\phi \in \mathcal{O}^+_\Sigma$ an observation. Let $N(\phi)$ be the smallest integer and $\mathtt{S}(\phi)$ the smallest (finite) set of closed terms such that $\phi \in (\mathtt{S}(\phi)^* \dot{\otimes} \Sigma^* \dot{\otimes} \mathtt{LR}^*) \dot{\otimes} \mathcal{S}_{N(\phi)}$.
The computation space $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ *is the subspace of $\mathbb{V}_{\mathtt{c}}$ generated by the terms*

$$s \bullet \mathtt{c} \bullet \mathtt{d} \bullet (a_1 \bullet \cdots \bullet a_{N(\phi)} \bullet \star)$$

where $s \in \mathtt{S}(\phi)$, $\mathtt{c} \in \Sigma$, $\mathtt{d} \in \mathtt{LR}$ and the $a_i \in \{ t_0, \dots, t_n \}$.
The dimension of $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ is $|\mathtt{S}(\phi)|\,|\Sigma| 2(n+1)^{N(\phi)}$ (where $|A|$ is the cardinal of A), which is polynomial in n.

Lemma 37 ***(separation)***
For any observation ϕ and any word W, the space $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ is separating for the wiring $\phi W(t_0, \dots, t_n)$.

Proof (of Theorem 34). With these lemmas at hand, we can define the non-deterministic algorithm below. It takes as an input the representation $W(t_0, \dots, t_n)$ of a word W of length n.

ϕ being a constant, one can compute once and for all $N(\phi)$ and $\mathtt{S}(\phi)$.

1: $D \leftarrow |\mathtt{S}(\phi)|\,|\Sigma|2(n+1)^{N(\phi)}$
2: $C \leftarrow 0$
3: pick a term $v \in \mathtt{Comp}_\phi(t_0, \dots, t_n)$
4: **while** $C \leq D$ **do**
5: **if** $(\phi W(t_0, \dots, t_n))(v) = 0$ **then**
6: **return** ACCEPT
7: **end if**
8: pick a term v' in $(\phi W(t_0, \dots, t_n))(v)$
9: $v \leftarrow v'$
10: $C \leftarrow C + 1$
11: **end while**
12: **return** REJECT

All computation paths (the "pick" at lines 3 and 8 being non-deterministic choices) accept if and only if $(\phi W(t_0, \dots, t_n))^n(\mathtt{Comp}_\phi(t_0, \dots, t_n)) = 0$ for some n lesser or equal to the dimension D of the computation space $\mathtt{Comp}_\phi(t_0, \dots, t_n)$. By Lemma 37, this is equivalent to $\phi W(t_0, \dots, t_n)$ being nilpotent.

The term chosen at lines 3 is representable by an integer of size at most D and is erased by the one chosen at line 8 every time we go through the **while**-loop. C and D are integers proportional to the dimension of the computation space, which is polynomial in n (Definition 36), thus representable in logarithmic space in the size of the input.

The computation of $(\phi W(t_0, \dots, t_n))(v)$ at line 5 and 8 and can be performed in logarithmic space by Proposition 8, as we are unifying closed terms with linear terms.

Moreover, if ϕ is an isometric wiring, $(\phi W(t_0, \dots, t_n))(v)$ consists of a single term instead of a sum by Lemma 17, and there is therefore no non-deterministic choice to be made at line 8. It is then enough to run the algorithm enumerating all possible terms of $\mathtt{Comp}_\phi(t_0, \dots, t_n)$ at line 3 to determine the nilpotency of $\phi W(t_0, \dots, t_n)$. □

4.2 Completeness: Representing Pointer Machines as Wirings

To prove the converse of Theorem 34, we prove that wirings can encode a special kind of read-only multi-head Turing Machine: pointers machines. The definition of this model will be guided by our understanding of the computation of wirings: they won't have the ability to write and acceptance will be defined as termination of all paths of computation. For a survey of this topic, one may consult the first author's thesis [21, Chap.4], the main novelty of this part of our work is to notice that reversible computation is represented by isometric operators.

Definition 38 ***(pointer machine)***
A pointer machine *over an alphabet Σ is a tuple $(N, \mathtt{S}, \Delta)$ where*

- *$N \neq 0$ is an integer, the* number of pointers,
- *$\mathtt{S}$ is a finite set, the* states *of the machine,*
- *$\Delta \subseteq (\mathtt{S} \times \Sigma \times \mathtt{LR}) \times (\mathtt{S} \times \Sigma \times \mathtt{LR}) \times \mathfrak{S}_N$, the* transitions *of the machine (we will write $(s, \mathtt{c}, \mathtt{d}) \to (s', \mathtt{c}', \mathtt{d}') \times \sigma$ the transitions, for readability).*

A pointer machine will be called deterministic *if for any $A \in \mathtt{S} \times \Sigma \times \mathtt{LR}$, there is at most one $B \in \mathtt{S} \times \Sigma \times \mathtt{LR}$ and one $\sigma \in \mathfrak{S}_N$ such that $A \to B \times \sigma \in \Delta$. In that case we can see Δ as a partial function, and we say that M is* reversible *if Δ is a partial injection.*

We call the first of the N pointers the *main* pointer, it is the only one that can move. The other pointers are referred to as the *auxiliary* pointers. An auxiliary pointer will be able to become the main pointer during the computation thanks to permutations.

Definition 39 ***(configuration)***
Given the length n of a word $W = \star \mathtt{c}_1 \dots \mathtt{c}_n$ over Σ and a pointer machine $M = (N, \mathtt{S}, \Delta)$, a configuration *of (M, n) is an element of*

$$\mathtt{S} \times \Sigma \times \mathtt{LR} \times \{0, 1, \dots, n\}^N$$

The element of $\mathtt{S}$ is the state of the machine and the element of Σ is the letter the main pointer points at. The element of $\mathtt{LR}$ is the direction of the next move of the main pointer, and the elements of $\{0, 1, \dots, n\}^N$ correspond to the positions of the (main and auxiliary) pointers on the input.

As the input tape is considered cyclic with a special symbol marking the beginning of the word (recall Definition 24), the pointer positions are integers *modulo* $n + 1$ for an input word of length n.

Definition 40 ***(transition)***
Let W be a word and $M = (N, \mathtt{S}, \Delta)$ be a pointer machine. A transition *of M on input W is a triple of configurations*

$$s, \mathtt{c}, \mathtt{d}, (p_1, \dots, p_N) \xrightarrow{\texttt{MOVE}} s, \mathtt{c}', \overline{\mathtt{d}}, (p'_1, \dots, p'_N) \xrightarrow{\texttt{SWAP}} s', \mathtt{c}'', \mathtt{d}', (p'_{\sigma(1)}, \dots, p'_{\sigma(N)})$$

such that

1. *if $\mathtt{d} \in \mathtt{LR}$, $\overline{\mathtt{d}}$ is the other element of $\mathtt{LR}$,*
2. *$p'_1 = p_1 + 1$ if $\mathtt{d} = \mathtt{R}$ and $p'_1 = p_1 - 1$ if $\mathtt{d} = \mathtt{L}$,*
3. *$p'_i = p_i$ for $i \neq 1$,*
4. *$\mathtt{c}$ is the letter at position p_1 and $\mathtt{c}'$ is the letter at position p'_1,*
5. *and $(s, \mathtt{c}', \overline{\mathtt{d}}) \to (s', \mathtt{c}'', \mathtt{d}') \times \sigma$ belongs to Δ.*

There is no constraint on c'', but every time this value differs from the letter pointed by $p'_{\sigma(1)}$, the computation will halt on the next `MOVE` phase, because there is a mismatch between the value that is supposed to have been read and the actual bit of W stored at this position, and that would contradict the first part of item 4. In terms of wirings, the `MOVE` phase corresponds to the application of the representation of the word, whereas the `SWAP` phase corresponds to the application of the observation.

Definition 41 ***(acceptance)***
We say that M accepts W if any sequence of transitions $\left(C_i \xrightarrow{\texttt{MOVE}} C'_i \xrightarrow{\texttt{SWAP}} C''_i\right)$ such that $C''_i = C_{i+1}$ for all i is necessarily finite.
We write $\mathcal{L}(M)$ the set of words accepted by M.

This means informally: we consider that a pointer machine accepts a word if it cannot ever loop, from whatever configuration it starts from. That a lot of paths of computation accepts "wrongly" is no worry, since only rejection is meaningful: our pointer machines compute in a "universally non-deterministic" way, to stick to the acceptance condition of wirings, nilpotency.

Proposition 42 ***(space and pointer machines)***
If $L \in$ NLogspace*, then there exist a pointer machine* M *such that* $\mathcal{L}(M) = L$*. Moreover, if* $L \in$ Logspace *then* M *can be chosen to be reversible.*

Proof (sketch). It is well-known [22] that read-only Turing Machines – or equivalently (non-)Deterministic Multi-Head Finite Automata – characterize (N)Logspace. It takes little effort to see that our pointer machines are just a reasonable rearrangement of this model, since it is always possible to encode the missing information in the states of the machine.
That acceptance and rejection are "reversed" is harmless in the deterministic (or equivalently reversible [23]) case, and uses that co-NLogspace = NLogspace to get the expected result in the non-deterministic case. □

As we said, our pointer machines are designed to be easily simulated by wirings, so that we get the expected result almost for free.

Theorem 43 ***(space completeness)***
If $L \in$ NLogspace*, then there exist an observation* $\phi \in \mathcal{O}_\Sigma^+$ *such that* $\mathcal{L}(\phi) = L$*. Moreover, if* $L \in$ Logspace *then* ϕ *is an isometric wiring.*

Proof. By Proposition 42, there exists a pointer machine $M = (N, \mathtt{S}, \Delta)$ such that $\mathcal{L}(M) = L$. We associate to the set $\mathtt{S}$ a set of distinct closed terms $[\mathtt{S}]$ and write $[s]$ the term associated to s. To any element $D = (s, \mathtt{c}, \mathtt{d}) \to (s', \mathtt{c}', \mathtt{d}') \times \sigma$ of Δ we associate the flow

$$[D] := ([s']_{\bullet\mathtt{c}'\bullet\mathtt{d}'} \leftharpoondown [s]_{\bullet\mathtt{c}\bullet\mathtt{d}}) \mathbin{\dot\otimes} [\sigma] \ \in ([\mathtt{S}]^* \mathbin{\dot\otimes} \Sigma^* \mathbin{\dot\otimes} \mathtt{LR}^*) \mathbin{\dot\otimes} \mathcal{S}_n \ \subseteq \ \mathcal{O}_\Sigma^+$$

and we define the observation $[M] \in \mathcal{O}_\Sigma^+$ as $\sum_{D \in \Delta} [D]$.

One can easily check that this translation preserves the language recognized (there is even a step by step simulation of the computation on the word W by the wiring $[M]W(t_0, \dots, t_n)$) and relates reversibility with isometricity: in fact, M is reversible if and only if $[M]$ is an isometric wiring. Then, if $L \in$ Logspace, M is deterministic and can always be chosen to be reversible [23]. □

Discussion

The language of the unification algebra gives us a twofold point of view on computation, either through algebraic structures (that are described finitely by wirings) or pointer machines. We may therefore start exploring possible variations of the construction, combining intuitions from both worlds.

For instance, the choice of a normative pair can affect the expressivity of the construction: the more restrictive the notion of representation of a word is, the more liberal that of an observation can become, as suggested by T. Seiller. Whether and how this can affect the corresponding complexity class is definitely a direction for future work.

Another pending question about this approach to complexity classes is to delimit the minimal prerequisites of the construction, its core.
Earlier works [13,14,15] made use of von Neumann algebras to get a setting that is expressive enough, we lighten the construction by using simpler objects. Yet, the possibility of representing the action of permutations on a unbounded tensor product is a common denominator that seems deeply related to logarithmic space and pointer machines.

The logical counterpart of this work also needs clarifying. Indeed, the idea of representation of words comes directly from proof-theory, while the notion of observation does not seem to correspond to any known logical construction.

Finally, execution in our setting being based on iteration of matching, which is computable efficiently by a parallel machine, it seems possible to relate our modelisation with parallel computation.

References

1. Girard, J.Y.: Linear logic. Theoret. Comput. Sci. 50(1), 1–101 (1987)
2. Girard, J.Y., Scedrov, A., Scott, P.J.: Bounded linear logic: a modular approach to polynomial-time computability. Theoret. Comput. Sci. 97(1), 1–66 (1992)
3. Girard, J.Y.: Light linear logic. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 145–176. Springer, Heidelberg (1995)
4. Schöpp, U.: Stratified bounded affine logic for logarithmic space. In: LICS, pp. 411–420. IEEE Computer Society (2007)
5. Dal Lago, U., Hofmann, M.: Bounded linear logic, revisited. LMCS 6(4) (2010)
6. Gaboardi, M., Marion, J.Y., Ronchi Della Rocca, S.: An implicit characterization of PSPACE. ACM Trans. Comput. 13(2), 18:1–18:36 (2012)
7. Baillot, P., Mazza, D.: Linear logic by levels and bounded time complexity. Theoret. Comput. Sci. 411(2), 470–503 (2010)
8. Girard, J.Y.: Towards a geometry of interaction. In: Gray, J.W., Ščedrov, A. (eds.) Proceedings of the AMS Conference on Categories, Logic and Computer Science. Categories in Computer Science and Logic, vol. 92, pp. 69–108. AMS (1989)
9. Asperti, A., Danos, V., Laneve, C., Regnier, L.: Paths in the lambda-calculus. In: LICS, pp. 426–436. IEEE Computer Society (1994)
10. Laurent, O.: A token machine for full geometry of interaction (extended abstract). In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 283–297. Springer, Heidelberg (2001)
11. Girard, J.Y.: Geometry of interaction 1: Interpretation of system F. Studies in Logic and the Foundations of Mathematics 127, 221–260 (1989)
12. Baillot, P., Pedicini, M.: Elementary complexity and geometry of interaction. Fund. Inform. 45(1-2), 1–31 (2001)
13. Girard, J.Y.: Normativity in logic. In: Dybjer, P., Lindström, S., Palmgren, E., Sundholm, G. (eds.) Epistemology Versus Ontology. Logic, Epistemology, and the Unity of Science, vol. 27, pp. 243–263. Springer (2012)

14. Aubert, C., Seiller, T.: Characterizing co-NL by a group action. Arxiv preprint abs/1209.3422 (2012)
15. Aubert, C., Seiller, T.: Logarithmic space and permutations. Arxiv preprint abs/1301.3189 (2013)
16. Girard, J.Y.: Geometry of interaction III: accommodating the additives. In: Girard, J.Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic. London Mathematical Society Lecture Note Series, vol. 222, pp. 329–389. CUP (1995)
17. Girard, J.Y.: Three lightings of logic. In: Ronchi Della Rocca, S. (ed.) CSL. LIPIcs, vol. 23, pp. 11–23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
18. Knight, K.: Unification: A multidisciplinary survey. ACM Comput. Surv. 21(1), 93–124 (1989)
19. Dwork, C., Kanellakis, P.C., Mitchell, J.C.: On the sequential nature of unification. J. Log. Program. 1(1), 35–50 (1984)
20. Dwork, C., Kanellakis, P.C., Stockmeyer, L.J.: Parallel algorithms for term matching. SIAM J. Comput. 17(4), 711–731 (1988)
21. Aubert, C.: Linear Logic and Sub-polynomial Classes of Complexity. PhD thesis, Université Paris 13–Sorbonne Paris Cité (2013)
22. Hartmanis, J.: On non-determinancy in simple computing devices. Acta Inform. 1(4), 336–344 (1972)
23. Lange, K.J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. J. Comput. System Sci. 60(2), 354–367 (2000)

Logic Programming and Logarithmic Space

Clément Aubert[1], Marc Bagnol[1], Paolo Pistone[1], and Thomas Seiller[2,*]

[1] Aix Marseille Université, CNRS, Centrale Marseille, I2M UMR 7373
13453 Marseille, France
[2] I.H.É.S., Le Bois-Marie, 35, Route de Chartres, 91440 Bures-sur-Yvette, France

Abstract. We present an algebraic view on logic programming, related to proof theory and more specifically linear logic and geometry of interaction. Within this construction, a characterization of logspace (deterministic and non-deterministic) computation is given *via* a syntactic restriction, using an encoding of words that derives from proof theory.

We show that the acceptance of a word by an observation (the counterpart of a program in the encoding) can be decided within logarithmic space, by reducing this problem to the acyclicity of a graph. We show moreover that observations are as expressive as two-ways multi-head finite automata, a kind of pointer machine that is a standard model of logarithmic space computation.

Keywords: Implicit Complexity, Unification, Logic Programming, Logarithmic Space, Proof Theory, Pointer Machines, Geometry of Interaction, Automata.

1 Introduction

Proof Theory and Implicit Computational Complexity. Very generally, the aim of implicit computational complexity (ICC) is to describe complexity classes with no explicit reference to cost bounds: through a type system or a weakened recursion scheme for instance. The last two decades have seen numerous works relating proof theory (more specifically linear logic [15]) and ICC, the basic idea being to look for restricted substructural logics [19] with an expressiveness that corresponds exactly to some complexity class.

This has been achieved by various syntactic restrictions, which entail a less complex[1] cut-elimination procedure: control over the modalities [31,10], type assignments [14] or stratification properties [5], to name a few.

Geometry of Interaction. In recent years, the cut-elimination procedure and its mathematical modeling has become a central topic in proof theory. The aim of the geometry of interaction research program [16] is to provide the tools for such a modeling [1,25,32].

* This work was partly supported by the ANR-10-BLAN-0213 Logoi and the ANR-11-BS02-0010 Récré.

[1] Any function provably total in second-order Peano Arithmetic [15] can be encoded in second-order linear logic.

J. Garrigue (Ed.): APLAS 2014, LNCS 8858, pp. 39–57, 2014.

As for complexity theory, these models allow for a more synthetic and abstract study of the resources needed to compute the normal form of a program, leading to some complexity characterization results [6,20,2].

Unification. Unification is one of the key-concepts of theoretical computer science: it is a classical subject of study for complexity theory and a tool with a wide range of applications, including logic programming and type inference algorithms.

Unification has also been used to build syntactic models of geometry of interaction [18,6,21] where first-order terms with variables allow for a manipulation of infinite sets through a finite language.

Logic Programming. After the work of Robinson [29] on the resolution procedure, logic programming has emerged as a new computation paradigm with concrete realizations such as the languages Prolog and Datalog.

On the theoretical side, constant efforts have been provided to clarify expressiveness and complexity issues [11]: most problems arising from logic programming are undecidable in their most general form and some restrictions must be introduced in order to make them tractable. For instance, the notion of *finitely ground program* [9] is related to our approach.

Pointer Machines. Multi-head finite automata provide an elegant characterization of logarithmic space computation, in terms of the (qualitative) type of memory used rather than the (quantitative) amount of tape consumed. Since they can scan but not modify the input, they are usually called "pointer machines", even if this nomenclature can be misleading [8].

This model was already at the heart of previous works relating geometry of interaction and complexity theory [20,3,2].

Contribution and Outline. We begin by exposing the idea of relating geometry of interaction and logic programming, already evoked [18] but never really developed, and by recalling the basic notions on unification theory needed for this article and some related complexity results.

We present in Sect. 2 the algebraic tools used later on to define the encoding of words and pointer machines. Section 2.2 and Sect. 2.3 introduce the syntactical restriction and associated tools that allow us to characterize logarithmic space computation. Note that, compared to earlier work [2], we consider a much wider class of programs while preserving bounded space evaluation: we switch from representation of permutations to a class defined by a syntactical restriction on height of variables, which contains permutations as a strict subset.

The encoding of words enabling our results, which comes from the classical (Church) encoding of lists in proof theory, is given in Sect. 3. It allows to define the counterpart of programs, and a notion of acceptance of a word by a program.

Finally, Sect. 4 makes use of the tools introduced earlier to state and prove our complexity results. While the expressiveness part is quite similar to earlier presentations [3,2], the proof that acceptance can be decided within logarithmic

space has been made more modular by reducing this problem to the standard problem of cycle search in a graph.

1.1 Geometry of Interaction and Logic Programming

The geometry of interaction program (GoI), started in 1989 [17], aims at describing the dynamics of computation by developing a fully mathematical model of cut-elimination. The original motivations of GoI must be traced back, firstly, to the *Curry-Howard correspondence* between sequent calculus derivations and typed functional programs: it is on the basis of this correspondence that cut-elimination had been proposed by proof-theorists as a paradigm of computation; secondly, to the finer analysis of cut-elimination coming from linear logic [15] and the replacement of sequent calculus derivations with simpler geometrical structures (proof-nets), more akin to a purely mathematical description.

In the first formulation of GoI [16], derivations in second order intuitionistic logic LJ[2] (which can be considered, by *Curry-Howard*, as programs in System F) are interpreted as pairs (U, σ) of elements (called *wirings*) of a $\mathbb{C}^*$-algebra, U corresponding to the axioms of the derivation and σ to the cuts.

The main property of this interpretation is *nilpotency*, *i.e.* if there exists an integer n such that $(\sigma U)^n = 0$. The cut-elimination (equivalently, the normalization) procedure is then interpreted by the application of an *execution operator*

$$EX(U, \sigma) = \sum_k (\sigma U)^k$$

From the viewpoint of proof theory and computation, nilpotency corresponds to the *strong normalization property*: the termination of the normalization procedure with any strategy.

Several alternative formulations of geometry of interaction have been proposed since 1989 (see for instance [1,25,32]); in particular, wirings can be described as logic programs [18,6,21] made of particular clauses called *flows*, which will be defined in Sect. 2.1.

In this setting the resolution rule induces a notion of product of wirings (Theorem 8) and in turn a structure of semiring: the *unification semiring* $\mathcal{U}$, which can replace the $\mathbb{C}^*$-algebras of the first formulations of GoI[2].

The $EX(.)$ operator of wirings can be understood as a way to compute the fixed point semantics of logic programs. The nilpotency property of wirings means then that the fixed point given by $EX(.)$ is finite, which is close to the notion of *boundedness*[3] [11] of logic programs.

[2] By adding complex scalar coefficients, one can actually extend $\mathcal{U}$ into a $\mathbb{C}^*$-algebra [18].

[3] A program is *bounded* if there is an integer k such that the fixed point computation of the program is stable after k iterations, independently of the facts input.

In definitive, from the strong normalization property for intuitionistic second order logic (or any other system which enjoys a GoI interpretation), one obtains through the GoI interpretation a family of bounded (nilpotent) logic programs computing the recursive functions typable in System F.

This is quite striking in view of the fact that to decide whenever a program is *bounded* is – even with drastic constraints – an undecidable problem [22], and that in general boundedness is a property that is difficult to ensure.

1.2 Unification and Complexity

We recall in the following some notations and some of the numerous links between complexity and unification, and by extension logic programming.

Notations. We consider a set of first-order terms $\mathtt{T}$, assuming an infinite number of variables $x, y, z, \ldots \in \mathtt{V}$, a binary function symbol $\bullet$ (written in *infix notation*), infinitely many constant symbols $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots$ including the (multipurpose) dummy symbol $\star$ and, for any $n \in \mathbb{N}^*$, at least one n-ary function symbol $\mathtt{A}_n$.

Note that the binary function symbol $\bullet$ is not associative. However, we will write it by convention as *right associating* to lighten notations: $t \bullet u \bullet v := t \bullet (u \bullet v)$.

For any $t \in \mathtt{T}$, we write $\mathtt{Var}(t)$ the set of variables occurring in t (a term is *closed* when $\mathtt{Var}(t) = \varnothing$) and $\mathtt{h}(t)$ the *height* of t: the maximal distance from the root to any leaf in the tree structure of t.

The *height of a variable occurrence* in a term t is its distance from the root in the tree structure of the term. A *substitution* θ is a mapping from variables to terms such that $x\theta = x$ for all but finitely many $x \in \mathtt{V}$. A *renaming* is a substitution α mapping variables to variables and that is bijective. A term t' is a *renaming* of t if $t' = t\alpha$ for some renaming α.

Definition 1 (unification, matching and disjointness). *Two terms t, u are*

- unifiable *if there exists a substitution θ, called a* unifier *of t and u, such that $t\theta = u\theta$. A unifier θ such that any other unifier of t and u is an instance of θ is called a* most general unifier *(MGU) of t and u,*
- matchable *if t', u' are unifiable, where t', u' are renamings of t, u such that $\mathtt{Var}(t') \cap \mathtt{Var}(u') = \varnothing$,*
- disjoint *if they are not matchable.*

A fundamental result [29] of the theory of unification is that two unifiable terms indeed have a MGU and that it can be computed.

More specifically, the problem of deciding whether two terms are unifiable is Ptime-complete [12, Theorem 1], which implies that parallel algorithms for this problem do not improve much on serial ones. Finding classes of terms where the MGU research can be efficiently parallelized is a real challenge.

It has been proven that this problem remains Ptime-complete even if the arity of the function symbols or the height of the terms is bounded [27, Theorems 4.2.1 and 4.3.1], if both terms are linear or if they do not share variables [12,13]. More recently [7], an innovative constraint on variables helped to discover an upper bound of the unification classes that are proven to be in NC.

Regarding space complexity, the result stating that the *matching problem* is in DLogspace [12] (recalled as Theorem 36) will be used in Sect. 4.2.

2 The Unification Semiring

This section presents the technical setting of this work, the *unification semiring*: an algebraic structure with a composition law based on unification, that can be seen as an algebraic presentation of a fragment of logic programming.

2.1 Flows and Wirings

Flows can be thought of as very specific Horn clauses: safe (the variables of the head must occur in the body) clauses with exactly one atom in the body.

As it is not relevant to this work, we make no technical difference between predicate symbols and function symbols, for it makes the presentation easier. Anyway, to retrieve the connection with logic programming, simply assume a class of function symbols called "predicate symbols" (written in boldface) that can only occur at the root of a term.

Definition 2 (flows). *A* flow *is a pair of terms $t \leftharpoonup u$ with $\mathtt{Var}(t) \subseteq \mathtt{Var}(u)$. Flows are considered up to renaming: for any renaming α, $t \leftharpoonup u = t\alpha \leftharpoonup u\alpha$.*

An example of flow that indeed is a clause of logic programming would be for instance $\mathbf{colored}(x) \leftharpoonup \mathbf{blue}(x)$ which states that if x is blue, then it is colored.

Facts, which are usually defined as ground (using only closed terms) clauses with an empty body, can still be represented as a special kind of flows.

Definition 3 (facts). *A* fact *is a flow of the form $t \leftharpoonup \star$.*

Remark 4. Note that this implies that t is closed.

Following on the example above, $\mathbf{blue}(\mathtt{c}) \leftharpoonup \star$ would be the fact stating that the object $\mathtt{c}$ is blue.

The main interest of the restriction to flows is that it yields an algebraic structure: a semigroup with a partially defined product.

Definition 5 (product of flows). *Let $u \leftharpoonup v$ and $t \leftharpoonup w$ be two flows. Suppose we have representatives of the renaming classes such that $\mathtt{Var}(v) \cap \mathtt{Var}(w) = \varnothing$. The* product *of $u \leftharpoonup v$ and $t \leftharpoonup w$ is defined if v, t are unifiable with MGU θ as $(u \leftharpoonup v)(t \leftharpoonup w) := u\theta \leftharpoonup w\theta$.*

Remark 6. The condition on variables ensures that facts form a "left ideal" of the set of flows: if $\mathbf{u}$ is a fact and f a flow, then $f\mathbf{u}$ is a fact when it is defined.

Example 7. $(\mathtt{f}(x) \leftharpoonup x)(\mathtt{f}(x) \leftharpoonup \mathtt{g}(x)) = \mathtt{f}(\mathtt{f}(x)) \leftharpoonup \mathtt{g}(x)$
$(x{\bullet}\mathtt{d} \leftharpoonup (y{\bullet}y){\bullet}x)((\mathtt{c}{\bullet}\mathtt{c}){\bullet}x \leftharpoonup y{\bullet}x) = x{\bullet}\mathtt{d} \leftharpoonup y{\bullet}x$
$(\mathtt{f}(x{\bullet}\mathtt{c}) \leftharpoonup x{\bullet}\mathtt{d})(\mathtt{d}{\bullet}\mathtt{d} \leftharpoonup \star) = \mathtt{f}(\mathtt{d}{\bullet}\mathtt{c}) \leftharpoonup \star$
$(x \leftharpoonup \mathtt{g}(\mathtt{h}(x)))(\mathtt{g}(y) \leftharpoonup y) = x \leftharpoonup \mathtt{h}(x)$

The product of flows corresponds to the resolution rule in the following sense: given two flows $f = u \leftharpoonup v$ and $g = t \leftharpoonup w$ and a MGU θ of v and t, then the resolution rule applied to f and g would yield fg.

To finish with our logic programming example, the product of the flows $\mathbf{colored}(x) \leftharpoonup \mathbf{blue}(x)$ and $\mathbf{blue}(\mathtt{c}) \leftharpoonup \star$ would yield $\mathbf{colored}(\mathtt{c}) \leftharpoonup \star$.

Wirings then correspond to logic programs (sets of clauses) and the nilpotency condition can be seen as an algebraic variant of the notion of boundedness of these programs.

Definition 8 (wirings). Wirings *are finite sets of flows. The product of wirings is defined as* $FG := \{\, fg \mid f \in F,\, g \in G,\, fg \text{ defined} \,\}$.

We write $\mathcal{U}$ *for the set of wirings and refer to it as the* unification semiring.

The set of wirings $\mathcal{U}$ has the structure of a semiring. We use an *additive notation* for sets of flows to stress this point:

- The symbol + will be used in place of $\cup$.
- We write sets as the sum of their elements: $\{\, f_1, \dots, f_n \,\} := f_1 + \cdots + f_n$.
- We write 0 for the empty set.
- The unit is $I := x \leftharpoonup x$.

We will call *semiring* any subset $\mathcal{A}$ of $\mathcal{U}$ such that

- $0 \in \mathcal{A}$,
- if $F \in \mathcal{A}$ and $G \in \mathcal{A}$ then $FG \in \mathcal{A}$.
- if $F, G \in \mathcal{A}$, then $F + G \in \mathcal{A}$,

A subset satisfying only the first two conditions will be called a *semigroup*.

Definition 9 (nilpotency). *A wiring* F *is* nilpotent *if* $F^n = 0$ *for some* $n \in \mathbb{N}$. *We may use the notation* $\mathbf{Nil}(F)$ *to express the fact that* F *is nilpotent.*

As mentioned in Sect. 1.1, nilpotency is related with the notion of *boundedness* [11] of a logic program. Indeed, if we have a wiring F and a finite set of facts $\mathbf{U}$, let us consider the set of facts that can be obtained through F, $\{\, \mathbf{u} \mid \mathbf{u} \in F^n\mathbf{U} \text{ for some } n \,\}$ which can also be written as $(I + F + F^2 + \cdots)\mathbf{U}$ or $EX(F)\mathbf{U}$ (where $EX(.)$ is the execution operator of Sect. 1.1). If F is nilpotent, one needs to compute the sum only up to a finite rank that does not depend on $\mathbf{U}$, which implies the boundedness property.

Among wirings, those that can produce at most one fact from any fact will be of interest when considering deterministic *vs.* non-deterministic computation.

Definition 10 (deterministic wirings). *A wiring* F *is* deterministic *if given any fact* $\mathbf{u}$, $\mathrm{card}(F\mathbf{u}) \leq 1$. *We will write* $\mathcal{U}_d$ *the set of deterministic wirings.*

It is clear from the definition that $\mathcal{U}_d$ forms a semigroup. The lemma below gives us a class of wirings that are deterministic and easy to recognize, due to its more syntactic definition.

Lemma 11. *Let* $F = \sum_i u_i \leftharpoonup t_i$. *If the* t_i *are pairwise disjoint (Theorem 1), then* F *is deterministic.*

Proof. Given a closed term t there is at most one of the t_i that matches t, therefore $F(t \leftharpoonup \star)$ is either a single fact or 0. □

2.2 The Balanced Semiring

In this section, we study a constraint on variable height of flows which we call *balance*. This syntactic constraint can be compared with similar ones proposed in order to get logic programs that are *finitely ground* [9]: balanced wirings are a special case of *argument-restricted* programs in the sense of [26].

We will be able to decide the nilpotency of balanced wirings in a space-efficient way, thanks to the results of Sect. 2.3.

Definition 12 (balance). *A flow $f = t \leftharpoonup u$ is* balanced *if for any variable $x \in \mathtt{Var}(t) \cup \mathtt{Var}(u)$, all occurrences of x in either t or u have the same height (recall notations p. 42) which we write $\mathtt{h}_f(x)$, the* height of x in f*. A wiring F is* balanced *if it is a sum of balanced flows.*

We write $\mathcal{U}_b$ for the set of balanced wirings and refer to it as the balanced semiring*.*

In Theorem 7, only the second line shows the product of balanced flows.

The basic idea behind the notion of balance is that it forbids variations of height which may be used to store information "above" a variable. Typically, the flow $\mathtt{f}(x) \leftharpoonup x$ is not balanced.

Definition 13 (height). *The* height $\mathtt{h}(f)$ *of a flow $f = t \leftharpoonup u$ is* $\max\{\mathtt{h}(t), \mathtt{h}(u)\}$*. The* height $\mathtt{h}(F)$ *of a wiring F is the maximal height of flows in it.*

The following lemma summarizes the properties that are preserved by the product of balanced flows. It implies in particular that $\mathcal{U}_b$ is indeed a semiring.

Lemma 14. *When it is defined, the product fg of two balanced flows f and g is still balanced and its height is at most* $\max\{\mathtt{h}(f), \mathtt{h}(g)\}$*.*

Proof (sketch). By showing that the variable height condition and the global height are both preserved by the basic steps of the unification procedure. □

2.3 The Computation Graph

The main tool for a space-efficient treatment of balanced wirings is an associated notion of graph. This section focuses on the algebraic aspects of this notion, proving various technical lemmas, and leaves the complexity issues to Sect. 4.2.

A separating space can be thought of as a finite subset of the Herbrand universe associated with a logic program, containing enough information to decide the problem at hand.

Definition 15 (separating space). *A* separating space *for a wiring F is a set of facts* **S** *such that*

- *For all* $\mathbf{u} \in \mathbf{S}$, $F\mathbf{u} \subseteq \mathbf{S}$.
- $F^n\mathbf{u} = 0$ *for all* $\mathbf{u} \in \mathbf{S}$ *implies* $F^n = 0$.

We can define such a space for balanced wirings with Theorem 14 in mind: balanced wirings behave well with respect to height of terms.

Definition 16 (computation space). *Given a balanced wiring* F*, we define its* computation space $\mathbf{Comp}(F)$ *as the set of facts of height at most* $\mathtt{h}(F)$*, built using only the symbols appearing in* F *and the constant symbol* $\star$*.*

Lemma 17 (separation). *If* F *is balanced, then* $\mathbf{Comp}(F)$ *is separating for* F*.*

Proof. By Theorem 14, $F(u \leftharpoonup \star)$ is of height at most $\max\{\mathtt{h}(F), \mathtt{h}(u)\} \leq \mathtt{h}(F)$ and it contains only symbols occurring in F and u, therefore if $\mathbf{u} \in \mathbf{Comp}(F)$ we have $F\mathbf{u} \subseteq \mathbf{Comp}(F)$.

By Theorem 14 again, F^n is still of height at most $\mathtt{h}(F)$. If $(F^n)\mathbf{u} = 0$ for all $\mathbf{u} \in \mathbf{Comp}(F)$, it means the flows of F^n do not match any closed term of height at most $\mathtt{h}(F)$ built with the symbols occurring in F (and eventually $\star$). This is only possible if F^n contains no flow, *ie.* $F^n = 0$. □

As F is a finite set, thus built with finitely many symbols, $\mathbf{Comp}(F)$ is also a finite set. We can be a little more precise and give a bound to its cardinality.

Proposition 18 (cardinality). *Let* F *be a balanced wiring,* A *the maximal arity of function symbols occurring in* F *and* S *the set of symbols occurring in* F*, then* $\mathrm{card}(\mathbf{Comp}(F)) \leq (\mathrm{card}(S)+1)^{P_{\mathtt{h}(F)}(A)}$*, where* $P_k(X) = 1 + X + \cdots + X^k$*.*

Proof. The number of terms of height $\mathtt{h}(F)$ built over the set of symbols $S \cup \{\star\}$ of arity bounded by A is at most as large as the number of complete trees of degree A and height $\mathtt{h}(F)$ (that is, trees where nodes of height less than $\mathtt{h}(F)$ have exactly A childs), with nodes labeled by elements of $S \cup \{\star\}$. □

Then, we can encode in a directed graph[4] the action of the wiring on its computation space.

Definition 19 (computation graph). *If* F *is a balanced wiring, we define its* computation graph $\mathbf{G}(F)$ *as the directed graph:*

- *The vertices of* $\mathbf{G}(F)$ *are the elements of* $\mathbf{Comp}(F)$*.*
- *There is an edge from* $\mathbf{u}$ *to* $\mathbf{v}$ *in* $\mathbf{G}(F)$ *if* $\mathbf{v} \in F\mathbf{u}$*.*

We state finally that the computation graph of a wiring contains enough information on the latter to determine its nilpotency. This is a key ingredient in the proof of Theorem 35, as the research of paths and cycles in graphs are problems that are well-known [24] to be solvable within logarithmic space.

Lemma 20. *A balanced wiring* F *is nilpotent (Theorem 9) iff* $\mathbf{G}(F)$ *is acyclic.*

[4] Here by directed graph we mean a set of *vertices* V together with a set of *edges* $E \subseteq V \times V$. We say that there is an edge *from* $e \in V$ *to* $f \in V$ when $(e, f) \in E$.

Proof. Suppose there is a cycle of length n in $\mathbf{G}(F)$, and let $\mathbf{u}$ be the label of a vertex which is part of this cycle. By definition of $\mathbf{G}(F)$, $\mathbf{u} \in (F^n)^k\mathbf{u}$ for all k, which means that $(F^n)^k \neq 0$ for all k and therefore F cannot be nilpotent.

Conversely, suppose there is no cycle in $\mathbf{G}(F)$. As it is a finite graph, this entails a maximal length N of paths in $\mathbf{G}(F)$. By definition of $\mathbf{G}(F)$, this means that $F^{N+1}\mathbf{u} = 0$ for all $\mathbf{u} \in \mathbf{Comp}(F)$ and with Theorem 17 we get $F^{N+1} = 0$. □

Moreover, the computation graph of a deterministic (Theorem 10) wiring has a specific shape, which in turn induces a deterministic procedure in this case.

Lemma 21. *If F is a balanced and deterministic wiring, $\mathbf{G}(F)$ has an out-degree (the maximal number of edges a vertex can be the source of) bounded by* 1*.*

Proof. It is a direct consequence of the definitions of $\mathbf{G}(F)$ and determinism. □

2.4 Tensor Product and Other Semirings

Finally, we list a few other semirings that will be used in the next section, where we define the notions of representation of a word and observation.

The binary function symbol $\bullet$ can be used to define an operation that is similar to the algebraic notion of tensor product.

Definition 22 (tensor product). *Let $u \leftharpoonup v$ and $t \leftharpoonup w$ be two flows. Suppose we have chosen representatives of their renaming classes that have disjoint sets of variables. We define their* tensor product *as $(u \leftharpoonup v) \dot{\otimes} (t \leftharpoonup w) := u\bullet t \leftharpoonup v\bullet w$. The operation is extended to wirings by $(\sum_i f_i) \dot{\otimes} (\sum_j g_j) := \sum_{i,j} f_i \dot{\otimes} g_j$. Given two semirings $\mathcal{A}, \mathcal{B}$, we define $\mathcal{A} \dot{\otimes} \mathcal{B} := \{ \sum_i F_i \dot{\otimes} G_i \mid F_i \in \mathcal{A}\,,\, G_i \in \mathcal{B} \}$.*

The tensor product of two semirings is easily shown to be a semiring.

Example 23. $(\mathtt{f}(x)\bullet y \leftharpoonup y\bullet x) \dot{\otimes} (x \leftharpoonup \mathtt{g}(x)) = (\mathtt{f}(x)\bullet y)\bullet z \leftharpoonup (y\bullet x)\bullet \mathtt{g}(z)$

Notation. As the symbol $\bullet$, the $\dot{\otimes}$ operation is not associative. We carry on the convention for $\bullet$ and write it as *right associating*: $\mathcal{A} \dot{\otimes} \mathcal{B} \dot{\otimes} \mathcal{C} := \mathcal{A} \dot{\otimes} (\mathcal{B} \dot{\otimes} \mathcal{C})$.

Semirings can also naturally be associated to any set of closed terms or to the restriction to a certain set of symbols.

Definition 24. *Given a set of closed terms E, we define the following semiring $E^{\leftharpoonup} := \{ \sum_i t_i \leftharpoonup u_i \mid t_i, u_i \in E \}$. If $\mathtt{S}$ is a set of symbols and $\mathcal{A}$ a semiring, we write $\mathcal{A}^{\setminus \mathtt{S}}$ the semiring of wirings of $\mathcal{A}$, that do not use the symbols in $\mathtt{S}$.*

This operation yields semirings because composition of flows made of closed terms involves no actual unification: it is just equality of terms and therefore one never steps out of $E^{\leftharpoonup}$.

Finally, the unit $I = x \leftharpoonup x$ of $\mathcal{U}$ yields a semiring.

Definition 25 (unit semiring). *The* unit semiring *is defined as $\mathcal{I} := \{ 0\,, I \}$.*

3 Words and Observations

We define in this section the global framework that will be used later on to obtain the characterization of logarithmic space computation. In order to discuss the contents of this section, let us first define two specific semirings.

Definition 26 (word and observation semirings). *We fix two (disjoint) infinite sets of constant symbols* $\mathtt{P}$ *and* $\mathtt{S}$*, and a unary function symbol* $\mathtt{M}$*. We denote by* $\mathtt{M}(\mathtt{P})$ *the set of terms* $\mathtt{M}(\mathtt{p})$ *with* $\mathtt{p} \in \mathtt{P}$*. We define the following two semirings that are included in* $\mathcal{U}_b$*:*

- *The* word semiring *is the semiring* $\mathcal{W} := \mathcal{I} \dot{\otimes} \mathcal{I} \dot{\otimes} \mathtt{M}(\mathtt{P})^{\leftarrow}$*.*
- *The* observation semiring *is the semiring* $\mathcal{O} := \mathtt{S}^{\leftarrow} \dot{\otimes} \mathcal{U}_b{}^{\setminus \mathtt{P}}$*.*

Remark 27. The expression $\mathcal{I} \dot{\otimes} \mathcal{I} \dot{\otimes} \mathtt{M}(\mathtt{P})^{\leftarrow}$ may seem odd at first sight, as the intuition from algebra is that $\mathcal{I} \dot{\otimes} \mathcal{I} \simeq \mathcal{I}$. But remember that we are here in a *syntactical* context and therefore we need to be careful with things that can usually be treated "up to isomorphism", as it may cause some unifications to fail where they should not.

These two semirings will be used as parameters of a construction $\mathcal{M}_{\Sigma}(.)$ over an alphabet Σ (we suppose $\star \notin \Sigma$), that will define the representation of words and a notion of abstract machine, that we shall call observations.

Definition 28. *We fix the set of constant symbols* $\mathtt{LR} := \{\mathtt{L}, \mathtt{R}\}$*.*

Given a set of constant symbols Σ *and a semiring* $\mathcal{A}$ *we define the semiring* $\mathcal{M}_{\Sigma}(\mathcal{A}) := (\Sigma \cup \{\star\})^{\leftarrow} \dot{\otimes} \mathtt{LR}^{\leftarrow} \dot{\otimes} \mathcal{A}$*.*

In the following of this section, we will show how to represent lists of elements of Σ by wirings in the semiring $\mathcal{M}_{\Sigma}(\mathcal{W})$. Then, we will explain how the semiring $\mathcal{M}_{\Sigma}(\mathcal{O})$ captures a notion of abstract machine. In the last section of the paper we will explain further how observations and words interact, and prove that this interaction captures logarithmic space computation.

3.1 Representation of Words

We now show how one can represent words by wirings in $\mathcal{M}_{\Sigma}(\mathcal{W})$. We recall this semiring is defined as $\left((\Sigma \cup \{\star\})^{\leftarrow} \dot{\otimes} \mathtt{LR}^{\leftarrow}\right) \dot{\otimes} \mathcal{I} \dot{\otimes} \mathcal{I} \dot{\otimes} \mathtt{M}(\mathtt{P})^{\leftarrow}$.

The part $(\Sigma \cup \{\star\})^{\leftarrow} \dot{\otimes} \mathtt{LR}^{\leftarrow}$ deals with, and is dependent on, the alphabet Σ considered; this is where the input and the observation will interact. The two instances of the unit semiring $\mathcal{I}$ correspond to the fact that the word cannot affect parts of the observation that correspond to internal configurations. The last part, namely the semiring $\mathtt{M}(\mathtt{P})^{\leftarrow}$, will contain the *position constants* of the representation of words.

Notation. We write $t \leftrightharpoons u$ for $t \leftharpoonup u + u \leftharpoonup t$.

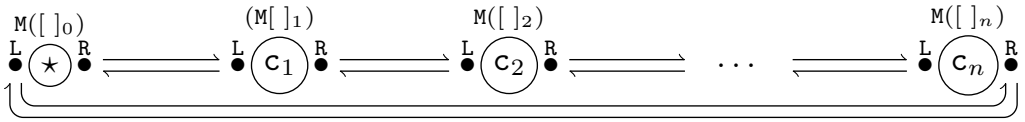
Definition 29 (word representations). *Let $W = \mathtt{c}_1, \dots, \mathtt{c}_n$ be a word over an alphabet Σ and $p = \mathtt{p}_0, \mathtt{p}_1, \dots, \mathtt{p}_n$ be pairwise distinct constant symbols.*

Writing $\mathtt{p}_{n+1} = \mathtt{p}_0$ and $\mathtt{c}_0 = \mathtt{c}_{n+1} = \star$, we define the representation *of W associated with $\mathtt{p}_0, \mathtt{p}_1, \dots, \mathtt{p}_n$ as the following wiring:*

$$\bar{W}_p = \sum_{i=0}^{n} \mathtt{c}_i \bullet \mathtt{R} \bullet x \bullet y \bullet \mathtt{M}(\mathtt{p}_i) \leftrightharpoons \mathtt{c}_{i+1} \bullet \mathtt{L} \bullet x \bullet y \bullet \mathtt{M}(\mathtt{p}_{i+1}) \tag{1}$$

We will write $\mathcal{R}(W)$ the set of representations of a given word W.

To better understand this representation, consider that each symbol in the alphabet Σ comes in two "flavors", *left* and *right*. Then, one can easily construct the "context" $\bar{W} = \sum_{i=0}^{n} \mathtt{c}_i \bullet \mathtt{R} \bullet x \bullet y \bullet \mathtt{M}([\,]_i) \leftrightharpoons \mathtt{c}_{i+1} \bullet \mathtt{L} \bullet x \bullet y \bullet \mathtt{M}([\,]_{i+1})$ from the list as the sums of the arrows in the following picture (where x and y are omitted):



Then, choosing a set $p = \mathtt{p}_0, \dots, \mathtt{p}_n$ of position constants, intuitively representing physical memory addresses, the representation $\bar{W}_p$ of a word associated with p is obtained by filling, for all $i = 0, \dots, n$, the hole $[\,]_i$ by the constant $\mathtt{p}_i$.

This abstract representation of words is not an arbitrary choice. It is inspired by the interpretation of lists in geometry of interaction.

Indeed, in System F, the type of binary lists corresponds to the formula $\forall X\ (X \Rightarrow X) \Rightarrow (X \Rightarrow X) \Rightarrow (X \Rightarrow X)$. Any lambda-term in normal form of this type can be written as $\lambda f_0 f_1 x.\, f_{\mathtt{c}_1} f_{\mathtt{c}_2} \cdots f_{\mathtt{c}_k} x$, where $\mathtt{c}_1 \cdots c_k$ is a word over $\{0, 1\}$. The GoI representation of such a lambda-term yields the abstract representation just defined[5]. Notice that the additional symbol $\star$ used to represent words corresponds to the variable x in the preceding lambda-term. Note also the fact that the representation of integer is *cyclic*, and that the symbol $\star$ serves as a reference for the starting/ending point of the word.

Let us finally stress that the words are represented as *deterministic* wirings. This implies that the restriction to deterministic observations will correspond to restricting ourselves to deterministic pointer machines. The framework, however, allows for a number of generalization and variants. For instance, one can define a representation of trees by adapting Theorem 29 in such a way that every vertex is related to its descendants; doing so would however yield non-deterministic wirings. In the same spirit, a notion of "one-way representations of words", defined by replacing the symbol $\leftrightharpoons$ by the symbol $\leftharpoonup$ in Eq. 1 of Theorem 29, could be used to characterize one-way multi-head automata.

3.2 Observations

We now define *observations*. We will then explain how these can be thought of as a kind of abstract machines. An observation is an element of the semiring

$$\mathcal{M}_\Sigma(\mathcal{O}) = (\Sigma \cup \{\star\})^{\leftharpoonup} \dot{\otimes} \mathtt{LR}^{\leftharpoonup} \dot{\otimes} (\mathtt{S}^{\leftharpoonup} \dot{\otimes} \mathcal{U}_b{}^{\setminus \mathtt{P}})$$

[5] A thorough explanation can be found in previous work by Aubert and Seiller [3].

Once again, the part of the semiring $(\Sigma \cup \{\star\})^{\leftarrow} \dot{\otimes} \mathtt{LR}^{\leftarrow}$ is dependent on the alphabet Σ considered and represents the point of interaction between the words and the machine. The semiring $\mathtt{S}^{\leftarrow}$ intuitively corresponds to the *states* of the observation, while the part ${\mathcal{U}_b}^{\setminus \mathtt{P}}$ forbids the machine to act non-trivially on the *position constants* of the representation of words. The fact that the machine cannot perform any operation on the memory addresses – the position constants – of the word representation explains why observations are naturally thought of as a kind of *pointer machines*.

Definition 30 (observation). *An* observation *is any element O of $\mathcal{M}_{\Sigma}(\mathcal{O})$.*

We can define the language associated to an observation. The condition of acceptance will be represented as the nilpotency of the product $O\bar{W}_p$ where $\bar{W}_p \in \mathcal{R}(W)$ represents a word W and O is an observation.

Definition 31 (language of an observation). *Let O be an observation on the alphabet Σ. We define the* language accepted by *O as*

$$\mathcal{L}(O) := \left\{ W \in \Sigma^* \;\middle|\; \forall p, \mathbf{Nil}(O\bar{W}_p) \right\}$$

One important point is that the semirings $\mathcal{M}_{\Sigma}(\mathcal{W})$ and $\mathcal{M}_{\Sigma}(\mathcal{O})$ are not completely disjoint, and therefore allow for non-trivial interaction of observations and words. However, they are sufficiently disjoint so that this computation does not depend on the choice of the representative of a given word.

Lemma 32. *Let W be a word, and $\bar{W}_p, \bar{W}_q \in \mathcal{R}(W)$. For every observation $O \in \mathcal{M}_{\Sigma}(\mathcal{O})$, $\mathbf{Nil}(O\bar{W}_p)$ if and only if $\mathbf{Nil}(O\bar{W}_q)$.*

Proof. As we pointed out, the observation cannot act on the position constants of the representations $\bar{W}_p$ and $\bar{W}_q$. This implies that for all integer k the wirings $(O\bar{W}_p)^k$ and $(O\bar{W}_q)^k$ are two instances of the same *context*, *i.e.* they are equal up to the interchange of the positions constants $\mathtt{p}_0, \dots, \mathtt{p}_n$ and $\mathtt{q}_0, \dots, \mathtt{q}_n$. This implies that $(O\bar{W}_p)^k = 0$ if and only if $(O\bar{W}_q)^k = 0$. □

Corollary 33. *Let O be an observation on the alphabet Σ. The set $\mathcal{L}(O)$ can be equivalently defined as the set*

$$\mathcal{L}(O) = \left\{ W \in \Sigma^* \;\middle|\; \exists p, \mathbf{Nil}(O\bar{W}_p) \right\}$$

This result implies that the notion of acceptance has the intended sense and is finitely verifiable: whether a word W is accepted by an observation O can be checked without considering all representations of W.

This kind of situation where two semirings $\mathcal{W}$ and $\mathcal{O}$ are disjoint enough to obtain Theorem 33 can be formalized through the notion of *normative pair* considered in earlier works [20,3,2].

4 Logarithmic Space

This section starts by explaining the computation one can perform with the observations, and prove that it corresponds to logarithmic space computation by showing how pointer machines can be simulated. Then, we will prove how the language of an observation can be decided within logarithmic space.

This section uses the complexity classes DLogspace and coNLogspace , as well as notions of completeness of a problem and reduction between problems. We use in Sect. 4.2 the classical theorem of coNLogspace -completeness of the acyclicity problem in directed graphs, and in Sect. 4.1 a convenient model of computation, two-ways multi-head finite automata [23], a generalization of automata also called "pointer machine". Note that the non-deterministic part of our results concerns coNLogspace , or equivalently NLogspace by the famous Immerman-Szelepcsényi theorem.

4.1 Completeness: Observations as Pointer Machines

Let $h_0, x, y \in \mathtt{V}$, $\mathtt{p}_0, \mathtt{p}_1, \mathtt{A}_0$ constants and $\Sigma = \{0, 1\}$, the excerpt of a dialogue in Figure 1 between an observation $O = o_1 + o_2 + \cdots$ and the representation of a word $\bar{W}_p = w_1 + w_2 + \cdots$ should help the reader to grasp the mechanism.

$$\star \bullet \mathtt{R} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(h_0) \leftharpoondown \star \bullet \mathtt{L} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(h_0) \qquad (o_1)$$
$$1 \bullet \mathtt{L} \bullet x \bullet y \bullet \mathtt{M}(\mathtt{p}_1) \leftharpoondown \star \bullet \mathtt{R} \bullet x \bullet y \bullet \mathtt{M}(\mathtt{p}_0) \qquad (w_1)$$
$$1 \bullet \mathtt{L} \bullet \mathbf{b} \bullet \mathtt{A}_0 \bullet \mathtt{M}(h_0) \leftharpoondown 1 \bullet \mathtt{L} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(h_0) \qquad (o_2)$$
$$\star \bullet \mathtt{R} \bullet x \bullet y \bullet \mathtt{M}(\mathtt{p}_0) \leftharpoondown 1 \bullet \mathtt{L} \bullet x \bullet y \bullet \mathtt{M}(\mathtt{p}_1) \qquad (w_2)$$

By unification,

$$1 \bullet \mathtt{L} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(\mathtt{p}_1) \leftharpoondown \star \bullet \mathtt{L} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(\mathtt{p}_0) \qquad (w_1 o_1)$$
$$1 \bullet \mathtt{L} \bullet \mathbf{b} \bullet \mathtt{A}_0 \bullet \mathtt{M}(\mathtt{p}_1) \leftharpoondown \star \bullet \mathtt{L} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(\mathtt{p}_0) \qquad (o_2 w_1 o_1)$$
$$\star \bullet \mathtt{R} \bullet \mathbf{b} \bullet \mathtt{A}_0 \bullet \mathtt{M}(\mathtt{p}_0) \leftharpoondown \star \bullet \mathtt{L} \bullet \mathbf{init} \bullet \mathtt{A}_0 \bullet \mathtt{M}(\mathtt{p}_0) \qquad (w_2 o_2 w_1 o_1)$$

This can be understood as the small following dialogue:

o_1: [*Is in state* **init**] "I read $\star$ from left to right, what do I read now?"
w_1: "Your position was $\mathtt{p}_0$, you are now in position $\mathtt{p}_1$ and read 1."
o_2: [*Change state to* **b**] "I do an about-turn, what do I read now?"
w_2: "You are now in position $\mathtt{p}_0$ and read $\star$."

Fig. 1. The beginning of a dialogue between an observation and the representation of a word

We just depicted two transitions corresponding to an automata that reads the first bit of the word, and if this bit is a 1, goes back to the starting position, in state **b**. We remark that the answer of w_1 differs from the one of w_2: there

is no need to clarify the position (the variable argument of $\mathtt{M}$), since h_0 was already replaced by $\mathtt{p}_1$. Such an information is needed only in the first step of the computation: after that, the updates of the position of the pointer take place on the word side. We remark that neither the state nor the constant $\mathtt{A}_0$ is an object of dialogue.

Note also that this excerpt corresponds to a deterministic computation. In general, several elements of the observation could get unified with the current configuration, yielding non-deterministic transitions.

Multiple Pointers and Swapping. We now add some computational power to our observations by adding the possibility to handle several pointers. The observations will now use a k-ary function $\mathtt{A}_k$ that allows to "store" k additional positions in the variables $h_1, \dots, h_k$. This part of the observation is not affected by the word, which means that only one head (the *main pointer*) can move. The observation can exchange the position of the main pointer and the position stored in $\mathtt{A}_k$: we therefore refer to the arguments of $\mathtt{A}_k$ as *auxiliary pointers* that can become the main pointer at some point of the computation. This is of course strictly equivalent to having several heads with the ability to move.

Consider the following flow, that encodes the transition "if the observation reads $1\bullet\mathtt{R}$ in state $\mathbf{s}$, it stores the position of the main pointer (the variable h_0) at the i-th position in $\mathtt{A}_k$ and start reading the input with a new pointer":

$$\star\bullet\mathtt{R}\bullet\mathbf{s}'\bullet\mathtt{A}_k(h_1, \dots, h_0, \dots, h_k)\bullet\mathtt{M}(h_i) \leftharpoondown 1\bullet\mathtt{R}\bullet\mathbf{s}\bullet\mathtt{A}_k(h_1, \dots, h_i, \dots, h_k)\bullet\mathtt{M}(h_0)$$

Suppose that later on, when reading $0\bullet\mathtt{L}$ in state $\mathbf{r}$, we want to give back to that pointer the role of main pointer. That means to swap again the position of the variables h_0 and h_i, in order to store the position that was currently read and to restore the position that was "frozen" in $\mathtt{A}_k$.

$$\_\bullet\mathtt{L}\bullet\mathbf{r}'\bullet\mathtt{A}_k(h_1, \dots, h_i, \dots, h_k)\bullet\mathtt{M}(h_0) \leftharpoondown 0\bullet\mathtt{L}\bullet\mathbf{r}\bullet\mathtt{A}_k(h_1, \dots, h_0, \dots, h_k)\bullet\mathtt{M}(h_i)$$

The occurrence of $\mathtt{L}$ in the head of the previous flow reflects that we want to read the input from left to right, but the "_" slot cannot be a free variable, for that would break the safety of our clauses, the fact that all the variable of the head (the left-member) appears in the body (the right-member). So this slot should be occupied by the last value read by the pointer represented by the variable h_0, an information that should be encoded in the state $\mathbf{r}$[6].

Acceptance and Rejection. Remember (Theorem 33) that the language of an observation is the set of words such that the wiring composed of the observation applied to a representation of the word is nilpotent. So one could add a flow with the body corresponding to the desired situation leading to acceptance, and the head being some constant ACCEPT that appears in the body of no other flow, thus ending computation when it is reached. But in fact, it is sufficient not to add any flow: doing nothing is accepting!

[6] That is, we should have states $\mathbf{r}_\star$, $\mathbf{r_0}$ and $\mathbf{r_1}$, and flows accordingly.

The real challenge is to reject a word: it means to loop. We cannot simply add the unit ($I := x \leftharpoonup x$) to our observation, since that would make our observation loop *for any input*. So we have to be more clever than that, and to encode rejection as a re-initialization of the observation: we want the observation to put all the pointers on $\star$ and to go back to an **init** state. So, a rejection is in fact a "perform for ever the same computation".

Suppose the main pointer was reading from right to left, that we are in state **b** and that we want to re-initialize the computation. Then, for every $\mathtt{c} \in \Sigma$, it is enough to add the transitions (go-back-c) and (re-init) to the observation,

$$\mathtt{c}\bullet\mathtt{L}\bullet\mathbf{b}\bullet\mathtt{A}(h_1, \dots, h_k)\bullet\mathtt{M}(h_0) \leftharpoonup \mathtt{c}\bullet\mathtt{R}\bullet\mathbf{b}\bullet\mathtt{A}(h_1, \dots, h_k)\bullet\mathtt{M}(h_0) \qquad \text{(go-back-c)}$$

$$\star\bullet\mathtt{R}\bullet\mathbf{init}\bullet\mathtt{A}(h_0, \dots, h_0)\bullet\mathtt{M}(h_0) \leftharpoonup \star\bullet\mathtt{R}\bullet\mathbf{b}\bullet\mathtt{A}(h_1, \dots, h_k)\bullet\mathtt{M}(h_0) \qquad \text{(re-init)}$$

Once the main pointer is back on $\star$, (re-init) re-initializes all the positions of the auxiliary pointers to the position of $\star$ and changes the state for **init**.

There is another justification for this design: as the observation and the representation of the word are sums, and as the computation is the application, any transition that can be applied will be applied, *i.e.* if the body of a flow of our observation and the head of a flow of the word can be unified, the computation will start in a possibly "wrong" initialization. That some of these incorrect runs accept for incorrect reason is no trouble, since only rejection is "meaningful" due to the nilpotency criterion. But, with this framework, an incorrect run will be re-initialized to the "right" initialization, and perform the correct computation: in that case, it will loop if and only if the input is rejected.

Two-Ways Multi-Heads Finite Automata and Completeness. The model we just developed has clearly the same expressivity as two-ways multi-head finite automata, a model of particular interest to us for it is well studied, tolerant to a lot of enhancements or restrictions[7] and gives an elegant characterization of DLogspace and NLogspace [23,28].

Then, by a plain and uniform encoding of two-ways multi-head finite automata, we get Theorem 34. That acceptance and rejection in the non-deterministic case are "reversed" (*i.e.* all path have to accept for the computation to accept) makes us characterize coNLogspace instead of NLogspace.

Note that encoding a *deterministic* automaton yields a wiring of the form of Theorem 11, which would be therefore a deterministic wiring.

Theorem 34. *If $L \in$ coNLogspace, then there is an observation O such that $\mathcal{L}(O) = L$. If moreover $L \in$ DLogspace, then O can be chosen deterministic.*

[7] In fact, most of the variations (the automata can be one-way, sweeping, rotating, oblivious, etc.) are studied in terms of number of states and additional heads needed to simulate a variation with another, but most of the time they keep characterizing the same complexity classes.

4.2 Soundness of Observations

We now use the results of Sect. 2.3 and Sect. 3.2 to design a procedure that decides whether a word belongs to the language of an observation within logarithmic space. This procedure will reduce this problem to the problem of testing the acyclicity of a graph, that is well-known to be tractable with logarithmic space resources.

First, we show how the computation graph of the product of the observation and the word representation can be constructed deterministically using only logarithmic space; then, we prove that testing the acyclicity of such a graph can be done within the same bounds. Here, depending on the shape of the graph (which is dependent in itself of determinism of the wiring, recall Theorem 21), the procedure will be deterministic or non-deterministic.

Finally, using the fact that logarithmic space algorithms can be composed [30, Fig. 8.10], Theorem 20 and Theorem 33, we will obtain the expected result:

Theorem 35. *If O is an observation, then $\mathcal{L}(O) \in$ coNLogspace. If moreover O is deterministic, then $\mathcal{L}(O) \in$ DLogspace.*

A Foreword on Word and Size. Given a word W over Σ, to build a representation $\bar{W}_p$ as in Theorem 29 is clearly in DLogspace: it is a plain matter of encoding. By Theorem 32, it is sufficient to consider a single representation. So for the rest of this procedure, we consider given $\bar{W}_p \in \mathcal{R}(W)$ and write $F := O\bar{W}_p$. The size of Σ is a constant, and it is clear that the maximal arity and the height of the balanced wiring F remain fixed when W varies. The only point that fluctuates is the cardinality of the set of symbols that occurs in F, and it is linearly growing with the length of W, corresponding to the number of position constants. In the following, any mention to a logarithmic amount of space is to be completed by "relatively to the length of W".

Building the Computation Graph. We need two main ingredients to build the computation graph (Theorem 19) of F: to enumerate the computation space $\mathbf{Comp}(F)$ (recall Theorem 16), and to determine whether there is an edge between two vertices.

By Theorem 18, $\mathrm{card}(\mathbf{Comp}(F))$ is polynomial in the size of W. Hence, given a balanced wiring F, a logarithmic amount of memory is enough to enumerate the members of $\mathbf{Comp}(F)$, that is the vertices of $\mathbf{G}(F)$.

Now the second part of the construction of $\mathbf{G}(F)$ is to determine if there is an edge between two vertices. Remember that there is an edge from $\mathbf{u} = u \leftharpoonup \star$ to $\mathbf{v} = v \leftharpoonup \star$ in $\mathbf{G}(F)$ if $\mathbf{v} \in F\mathbf{u}$. So one has to scan the members of $F = O\bar{W}_p$: if there exists $(t_1 \leftharpoonup t_2)(t'_1 \leftharpoonup t'_2) \in F$ such that $(t_1 \leftharpoonup t_2)(t'_1 \leftharpoonup t'_2)(u \leftharpoonup \star) = v \leftharpoonup \star$, then there is an edge from $\mathbf{u}$ to $\mathbf{v}$. To list the members of F is in DLogspace, but unification in general is a difficult problem (see Sect. 1.2). The special case of matching can be tested with a logarithmic amount of space:

Theorem 36 (matching is in DLogspace [12, p. 49]). *Given two terms t and u such that either t or u is closed, deciding if they are matchable is in* DLogspace.

Actually, this result relies on a subtle manipulation of the representation of the terms as *simple directed acyclic graphs* [4], where the variables are "shared". Translations between this representation of terms and the usual one can be performed in logarithmic space [12, p. 38].

Deciding if $\mathbf{G}(F)$ is Acyclic We know thanks to Theorem 20 that answering this question is equivalent to deciding if F is nilpotent. We may notice that $\mathbf{G}(F)$ is a directed, potentially unconnected graph of size $\mathrm{card}(\mathbf{Comp}(F))$.

It is well-know that testing for acyclicity of a directed graph is a coNLogspace [24, p. 83] problem. Moreover, if F is deterministic (which is the case when O is), then $\mathbf{G}(F)$ has out-degree bounded by 1 (Theorem 21) and one can test its acyclicity without being non-deterministic: it is enough to list the vertices of $\mathbf{Comp}(F)$, and for each of them to follow $\mathrm{card}(\mathbf{Comp}(F))$ edges and to test for equality with the vertex picked at the beginning. If a loop is found, the algorithm rejects, otherwise it accepts after testing the last vertex. Only the starting vertex and the current vertex need to be stored, which fits within logarithmic space, and there is no need to do any non-deterministic transitions.

5 Conclusion

We presented the unification semiring, a construction that can be used both as an algebraic model of logic programming and as a setting for a dynamic model of logic. Within this semiring, we were able to identify a class of wirings that have the exact expressive power of logarithmic space computation.

If we try to step back a little, we can notice that the main tool in the soundness proof (Sect. 4.2) is the computation graph, defined in Sect. 2.3. More precisely, the properties of this graph, notably its cardinality (that turns out to be polynomial in the size of the input), allow to define a decision procedure that needs only logarithmic space. The technique is modular, hence not limited to logarithmic space: identifying other conditions on wirings that ensure size bounds on the computation graph would be a first step towards the characterization of other space complexity classes.

Concerning completeness, the choice of encoding pointer machines (Sect. 4.1) rather than log-space bounded Turing machines was quite natural. Balanced wirings correspond to the idea of computing with pointers: manipulation of data without writing abilities, and thus with no capacity to store any information other than a fixed number of positions on the input.

By considering other classes of wirings or by modifying the encoding it might be possible to capture other notions of machines characterizing some complexity classes: we already mentioned at the end of Sect. 3.1 a modification of the representation of the word that would model one-way finite automata.

The relation with proof theory needs to be explored further: the approach of this paper seems indeed to suggest a sort of "Curry-Howard" correspondence for logic programming.

As Sect. 1.1 highlighted, there are many notions that might be transferable from one field to the other, thanks to a common setting provided by geometry of interaction and the unification semiring. Most notably, the notion of nilpotency (on the proof-theoretic side: strong normalization) corresponds to a variant of boundedness of logic programs, a property that is usually hard to ensure.

Another direction could be to look for a proof-system counterpart of this work: a corresponding "balanced" logic of logarithmic space.

Acknowledgments. The authors would like to thank the anonymous referees for helpful suggestions and comments.

References

1. Asperti, A., Danos, V., Laneve, C., Regnier, L.: Paths in the lambda-calculus. In: LICS, pp. 426–436. IEEE Computer Society (1994)
2. Aubert, C., Bagnol, M.: Unification and logarithmic space. In: Dowek, G. (ed.) RTA-TLCA 2014. LNCS, vol. 8560, pp. 77–92. Springer, Heidelberg (2014)
3. Aubert, C., Seiller, T.: Characterizing co-nl by a group action. Arxiv preprint abs/1209.3422 (2012)
4. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 445–532. Elsevier and MIT Press (2001)
5. Baillot, P., Mazza, D.: Linear logic by levels and bounded time complexity. Theoret. Comput. Sci. 411(2), 470–503 (2010)
6. Baillot, P., Pedicini, M.: Elementary complexity and geometry of interaction. Fund. Inform. 45(1-2), 1–31 (2001)
7. Bellia, M., Occhiuto, M.E.: N-axioms parallel unification. Fund. Inform. 55(2), 115–128 (2003)
8. Ben-Amram, A.M.: What is a "pointer machine"? Science of Computer Programming 26, 88–95 (1995)
9. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable functions in ASP: Theory and implementation. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 407–424. Springer, Heidelberg (2008)
10. Dal Lago, U., Hofmann, M.: Bounded linear logic, revisited. Log. Meth. Comput. Sci. 6(4) (2010)
11. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. 33(3), 374–425 (2001)
12. Dwork, C., Kanellakis, P.C., Mitchell, J.C.: On the sequential nature of unification. J. Log. Program. 1(1), 35–50 (1984)
13. Dwork, C., Kanellakis, P.C., Stockmeyer, L.J.: Parallel algorithms for term matching. SIAM J. Comput. 17(4), 711–731 (1988)
14. Gaboardi, M., Marion, J.Y., Ronchi Della Rocca, S.: An implicit characterization of pspace. ACM Trans. Comput. Log. 13(2), 18:1–18:36 (2012)
15. Girard, J.Y.: Linear logic. Theoret. Comput. Sci. 50(1), 1–101 (1987)
16. Girard, J.Y.: Geometry of interaction 1: Interpretation of system F. Studies in Logic and the Foundations of Mathematics 127, 221–260 (1989)
17. Girard, J.Y.: Towards a geometry of interaction. In: Gray, J.W., Ščedrov, A. (eds.) Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference held, June 14-20. Categories in Computer Science and Logic, vol. 92, pp. 69–108. AMS (1989)

18. Girard, J.Y.: Geometry of interaction III: Accommodating the additives. In: Girard, J.Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic. London Math. Soc. Lecture Note Ser., vol. 222, pp. 329–389. CUP (1995)
19. Girard, J.Y.: Light linear logic. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 145–176. Springer, Heidelberg (1995)
20. Girard, J.Y.: Normativity in logic. In: Dybjer, P., Lindstrm, S., Palmgren, E., Sundholm, G. (eds.) Epistemology versus Ontology. Logic, Epistemology, and the Unity of Science, vol. 27, pp. 243–263. Springer (2012)
21. Girard, J.Y.: Three lightings of logic. In: Ronchi Della Rocca, S. (ed.) CSL. LIPIcs, vol. 23, pp. 11–23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
22. Hillebrand, G.G., Kanellakis, P.C., Mairson, H.G., Vardi, M.Y.: Undecidable boundedness problems for datalog programs. J. Log. Program. 25(2), 163–190 (1995)
23. Holzer, M., Kutrib, M., Malcher, A.: Multi-head finite automata: Characterizations, concepts and open problems. In: Neary, T., Woods, D., Seda, A.K., Murphy, N. (eds.) CSP. EPTCS, vol. 1, pp. 93–107 (2008)
24. Jones, N.D.: Space-bounded reducibility among combinatorial problems. J. Comput. Syst. Sci. 11(1), 68–85 (1975)
25. Laurent, O.: A token machine for full geometry of interaction (extended abstract). In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 283–297. Springer, Heidelberg (2001)
26. Lierler, Y., Lifschitz, V.: One more decidable class of finitely ground programs. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 489–493. Springer, Heidelberg (2009)
27. Ohkubo, M., Yasuura, H., Yajima, S.: On parallel computation time of unification for restricted terms. Tech. rep., Kyoto University (1987)
28. Pighizzini, G.: Two-way finite automata: Old and recent results. Fund. Inform. 126(2-3), 225–246 (2013)
29. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM 12(1), 23–41 (1965)
30. Savage, J.E.: Models of computation - exploring the power of computing. Addison-Wesley (1998)
31. Schöpp, U.: Stratified bounded affine logic for logarithmic space. In: LICS, pp. 411–420. IEEE Computer Society (2007)
32. Seiller, T.: Interaction graphs: Multiplicatives. Ann. Pure Appl. Logic 163, 1808–1837 (2012)

UNIVERSITÉ PARIS 13
Laboratoire d'Informatique de Paris-Nord
UMR CNRS 7030

Villetaneuse, le 27 novembre 2014

Recommandation pour Clément Aubert

Clément Aubert a été moniteur à l'IUT de Villetaneuse de septembre 2010 à août 2013, au département Réseaux et Télécommunications que je dirigeais jusqu'en décembre 2011.

Lorsque Clément Aubert a rejoint le département, sa formation initiale ne l'avait pas préparé aux enseignements dans lesquels il serait amené à intervenir au sein de notre département. Il a par conséquent dû s'investir fortement dans la préparation de ses séances.

Pendant les trois années de monitorat, les enseignements confiés à Clément Aubert ont évolué avec une technicité croissante, commençant par du soutien en mathématiques via une plateforme en ligne, de l'encadrement de projets informatiques, puis des travaux dirigés et travaux pratiques d'algorithmique et programmation ainsi que de bases de données, et enfin un module d'administration système et réseaux (sous linux) comportant une très grande majorité de travaux pratiques. Cette progression lui a permis d'apprendre les éléments-clés de ces domaines qu'il ne connaissait au départ pas ou peu.

Pour acquérir les connaissances et compétences nécessaires, après avoir étudié les feuilles d'exercices et corrigés qui lui étaient fournis, et réalisé les travaux pratiques, Clément Aubert n'a pas hésité à demander des explication complémentaires à ses collègues. Ainsi, dans le module d'administration système et réseaux que j'enseigne, il a atteint un degré de maîtrise certain qui l'a amené à proposer des sujets d'examen.

De plus, Clément Aubert a activement participé aux diverses commissions pédagogiques, fournissant un avis pertinent sur nos étudiants.

Enfin, en ce qui concerne ses recherches, je peux attester des mêmes qualités de pédagogie, rigueur et collaboration. Lors d'exposés, j'ai pu constater qu'il savait valoriser ses travaux, la thématique dans laquelle il s'inscrit, et en motiver les perspectives.

Clément Aubert est un collègue de grande qualité, que je recommande *très vivement*. L'université qui le recrutera ne pourra que se féliciter de ce choix.

Laure Petrucci
Directrice du LIPN
Chef de département R&T (2009-2011)

INSTITUT GALILÉE : 99, Avenue J.-B. Clément • 93430 Villetaneuse (France)
Téléphone : (+ 33) 01 49 40 35 90 • Fax : (+ 33) 01 48 26 07 12

Villetaneuse, le 30 novembre 2014

Réseaux & Télécommunications

Objet : Recommandation de Clément Aubert

M. Clément Aubert a été Moniteur durant trois ans (2010-2013) au département Réseaux et Télécommunications de l'IUT de Villetaneuse, dont je suis responsable. Il y a effectué un service de 64h eq TD par an, réparti comme suit :

- 18 heures, TP - TD, sem. 1, AA1 - Apprendre autrement : sensibilisation aux TICE et au travail en autonomie sur la plate-forme WIMS, exercices de mathématiques (responsable : F. Benkhaldoun) ;
- 41 heures, TP - TD - CM, sem. 2, AA2 - Apprendre autrement : apprentissage de la gestion de projets, en C, PHP, MySQL et Java (responsable : F. Benkhaldoun) ;
- 54 heures, TP-TD, sem. 1, 2011-2012, I3 - Algorithmique et programmation en C (responsable : C. Coti).
- 52 heures, TP, sem. 2, 2011-2013, R3 - Administration des systèmes d'exploitation réseaux (responsable : L. Petrucci).
- 26 heures, TP, sem. 2, I4 - Bases de données (responsable : J.-M. Barrachina).

Clément a assuré ses enseignements avec un très grand sérieux et enthousiasme. Il faut souligner que l'informatique n'était pas sa spécialité initiale, et qu'il a su se former rapidement pour arriver à enseigner dans des disciplines techniques, qui demandent un temps de préparation important. Son travail et ses qualités humaines et pédagogiques ont été très appréciés des étudiants comme de tous ses collègues.

Je pense que Clément Aubert est un enseignant-chercheur très compétent et agréable, et *recommande très vivement* sa qualification dans la section Informatique (27) du CNU.

Emmanuel Viennet
Chef du Département R&T,
Professeur des Universités

Institut de Technologie de Villetaneuse – Université Paris 13
99, avenue Jean-Baptiste Clément – F 93430 Villetaneuse – Téléphone **+33** (0) **49 40 37 16**
Télécopie **+33** (0)**1 49 40 20 16** - E-mail **secrgtr@iutv.univ-paris13.fr** - Web **www.iutv.univ-paris13.fr**

Camille Coti
Université Paris 13, IUT de Villetaneuse,
99 avenue Jean-Baptiste Clément,
93 430 Villetaneuse
01 49 40 36 93

Villetaneuse, le 28 novembre 2014

Objet : recommandation de Clément Aubert en vue de son inscription sur la liste de qualification aux fonctions de Maître de Conférences

Madame, Monsieur,

Je suis Maître de Conférences à l'IUT de Villetaneuse. J'ai eu l'occasion de travailler avec Clément Aubert en 2011-2012, lorsque j'étais responsable du module d'introduction à l'algorithmique et à la programmation en langage C. Clément était alors doctorant allocataire-moniteur.

Il a eu la charge d'un groupe de TD-TP. Cette année-là, j'ai mis en place ce cours pour la première fois en le montant intégralement. Dans un premier temps, Clément s'est montré particulièrement utile à la préparation du module en tant que relecteur des supports de cours et des sujets de TD et TP, apportant bon nombre de suggestions utiles.

Au cours du module, il a ensuite été un excellent chargé de TD-TP. Il connaissait très bien la matière enseignée, et a été très à l'écoute des questions des étudiants. J'ai eu un bon retour des étudiants sur cet aspects, les étudiants étant particulièrement élogieux de sa disponibilité et de ses explications techniques. Enfin, il a été efficace sur la correction des copies et des TP notés.

Par conséquent, je ne peux que recommander chaleureusement l'inscription de Clément Aubert sur la liste de qualification aux fonctions de Maître de Conférences en section 27 en étant totalement confiante quant à ses qualités d'enseignant.

Avec l'expression de mon entière considération,

Camille Coti
Maître de Conférences,
IUT de Villetaneuse, Université Paris 13