# 5th International Workshop on Proof, Computation, Complexity

## PCC '06

The aim of PCC is to stimulate research in proof theory, computation, and complexity, focusing on issues which combine logical and computational aspects. Topics may include applications of formal inference systems in computer science, as well as new developments in proof theory motivated by computer science demands. Specific areas of interest are (non-exhaustively listed) foundations for specification and programming languages, logical methods in specification and program development, new developments in structural proof theory, and implicit computational complexity.

# 5th International Workshop on Proof, Computation, Complexity

# **PCC '06**

Ilmenau, July 24 - 25, 2006

Edited by

Karl-Heinz Niggl
TU Ilmenau, Germany

Reinhard Kahle
Universidade de Coimbra e CENTRIA, UNL, Portugal

Birgit Elbl
UniBw München, Germany

Universitätsverlag Ilmenau
2006

## Impressum

**Bibliographische Information der Deutschen Bibliothek**
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Angaben sind im Internet über http://dnb.ddb.de abrufbar.

Technische Universität Ilmenau/Universitätsbibliothek
**Universitätsverlag Ilmenau**
Postfach 10 05 65
98684 Ilmenau

**Herstellung und Auslieferung**
Verlagshaus Monsenstein und Vannerdat OHG
Am Hawerkamp 31
48155 Münster

ISBN 3-939473-01-4

# Abstracts

# On some small subrecursive hierarchies

**Mathias Barra**

Department of Mathematics, University of Oslo
Postboks 1053, Blindern, N-0316 Oslo, Norway
`georgba@math.uio.no`
`http://www.math.uio.no/~georgba`

***Abstract.***
This talk is to report on joint work with Lars Kristiansen [3], *The Small Grzegorczyk Classes and the Typed λ-Calculus*, and some related ideas. The article was presented at the CiE conference in Amsterdam 2005.
The class $\Delta_0^{\mathbb{N}}$ of rudimentary relations and the small relational Grzegorczyk classes $\mathcal{E}_\star^0$, $\mathcal{E}_\star^1$, $\mathcal{E}_\star^2$ attracted fairly much attention during the second half of the previous century, yet, the open problems imposed by these classes are still there for new generations to ponder on. It is well-known, and rather obvious, that $\Delta_0^{\mathbb{N}} \subseteq \mathcal{E}_\star^0 \subseteq \mathcal{E}_\star^1 \subseteq \mathcal{E}_\star^2$, but it is not known if any of the inclusions are strict, indeed it is open if the inclusion $\Delta_0^{\mathbb{N}} \subseteq \mathcal{E}_\star^2$ is strict.[1] Bel'tyukov [1] proved that $\mathcal{E}_\star^1 = \mathcal{E}_\star^2$ implies $\mathcal{E}_\star^0 = \mathcal{E}_\star^2$. Furthermore, we know that $\Delta_0^{\mathbb{N}} = \mathcal{E}_\star^0$ implies $\Delta_0^{\mathbb{N}} = \mathcal{E}_\star^2$. We do not know if this is proved anywhere else in the literature, but this is a straightforward corollary of some of the theorems in [3].
The open problems can be traced back to Grzegorczyk's seminal paper [2] from 1953, and it is fair to say that the problems belong to subrecursion theory, but they are closely related to those of complexity theory and computer science. Recalling that LINSPACE is the class of number-theoretic relations decidable by a deterministic Turing machine working in linear space, Ritchie [5] proved in 1963 that LINSPACE $= \mathcal{E}_\star^2$. Many of the other standard complexity classes, e.g. P and LOGSPACE, have since been characterised by subrecursive classes.
In [3] we introduce the $\mathcal{L}$-hierarchy:

$$\mathcal{L}^0 \subseteq \mathcal{L}^1 \subseteq \mathcal{L}^2 \subseteq \cdots \qquad \mathcal{L} = \bigcup_{i<\omega} \mathcal{L}^i$$

[1]For any set $\mathcal{X}$ of functions, $\mathcal{X}_\star$ denotes the set of relations whose characteristic function belong to $\mathcal{X}$.

which might shed some new light on the open problems described above.
A class in the $\mathcal{L}$-hierarchy is defined by a certain fragment of the $T^-$-calculus. Here $T^-$ is itself a fragment of the typed $\lambda$-calculus extended with zero-, recursor- and successor constants and appropriate reduction rules (Gödel's $T$). Full $T^-$ may be informally described as $T$ *without the successor*, and variations over this theme within various computational models have been the focus of research for Kristiansen and me for the last years. We have:

- $\Delta_0^{\mathbb{N}} \subseteq \mathcal{L}_\star^0 \subseteq \mathcal{E}_\star^0 \subseteq \mathcal{E}_\star^1 \subseteq \mathcal{L}_\star^1 \subseteq \cdots \subseteq \mathcal{L}_\star = \mathcal{E}_\star^2$
- $\forall i, j \in \mathbb{N}\big(\mathcal{L}^i = \mathcal{L}^j \iff \mathcal{L}_\star^i = \mathcal{L}_\star^j\big)$

Hence $\mathcal{E}_\star^0 \neq \mathcal{E}_\star^2$ if $\mathcal{L}^i \neq \mathcal{L}^j$ for some $i, j > 0$, and $\Delta_0^{\mathbb{N}} \neq \mathcal{E}_\star^2$ if $\mathcal{L}^0 \neq \mathcal{L}^j$ for some $j > 0$. No explicit bounds are embodied in the definition. Thus, we have a so-called implicit characterisation of $\mathcal{E}_\star^2$ (LINSPACE).
The proof relating the classes $\mathcal{L}_\star^i$ to the Grzegorczyk classes goes via a second hierarchy; the subrecursive Grzegorczyk-like $\mathcal{G}$. The classes $\mathcal{G}_\star^i$ are easily related to the $\mathcal{E}_\star^i$ classes, and the equality $\mathcal{L}_\star^i = \mathcal{G}_\star^i$ is then proved.
In [4] Kristiansen and Voda show that many well-known deterministic complexity classes can be characterised by fragments of $T^-$. Furthermore, a proof sketch based on Turing machines is given, showing that the *type 0 fragment* defining $\mathcal{L}$ captures LINSPACE, and thus $\mathcal{E}_\star^2$. In [3] we make no detours via Turing machines and give a more direct proof of the equality $\mathcal{L}_\star = \mathcal{E}_\star^2$.
Further ideas include extending and modifying the subrecursive hierarchy $\mathcal{G}$ to capture more complexity classes (also time classes), and to further analyse what happens at the bottom of the hierarchy, i.e. between $\Delta_0^{\mathbb{N}}$ and $\mathcal{E}_\star^0$.

[1] A. P. Bel'tyukov. *A machine description and the hierarchy of initial Grzegorczyk classes.* Zap. Naucn. Sem. Leninigrad. Otdel. May. Inst. Steklov. (LOMI) 88 (1979): 30–46, J. Soviet Math. 20

[2] A. Grzegorczyk. *Some classes of recursive functions.* Rozprawy Mat., 4 (1953):1–45

[3] L. Kristiansen and M. Barra. *The small Grzegorczyk classes and the typed $\lambda$-calculus.* New Computational Paradigms, Vol 3526 of LNCS, Springer Verlag, (2005):252–262

[4] L. Kristiansen and P.J. Voda. *Typed $\lambda$-calculi and computational complexity.* (Submitted.)

[5] R. W. Ritchie (1963). *Classes of predictably computable functions.* Trans. Am. Math. Soc. 106, (1963):139–173

# Strong normalisation via domain-theoretic computability predicates

**Ulrich Berger**

University of Wales, Swansea, UK
`u.berger@swansea.ac.uk`

***Abstract.***

It is well-known that intersection types on the one hand characterise the strongly normalisable lambda terms, and on the other hand give rise to a syntactically defined domain model for the lambda calculus. Recently, Coquand and Spiwack have combined and extended these facts to give a domain-theoretic criteria for strong normalisability for the lambda-calculus with recursively defined constants.
In the talk, I will show that their normalisation proof can also be carried out in an abstract axiomatic setting where the computability predicates are indexed by the elements of a (not syntactically presented) domain. The more abstract setting simplifies the proof considerably without sacrificing constructivity.

# Life after "life without cons"

**Guillaume Bonfante**

Loria, Calligramme project,
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France, and
École Nationale Supérieure des Mines de Nancy, INPL, France.
`Guillaume.Bonfante@loria.fr`

***Abstract.***

We propose two characterizations of complexity classes by means of programming languages. The first concerns LOGSPACE while the second leads to PTIME. The latter characterization shows that adding a choice command to a PTIME language (the language `WHILE` of Jones [6]) may not necessarily provide NPTIME computations. The result is close to Cook in [4] who used "auxiliary push-down automata". LOGSPACE is obtained through a decidable mechanism of tiering. It is based on an analysis of deforestation due to Wadler in [13]. We get also a characterization of NLOGSPACE.

The present work arises as a side effect of the much long-term research on quasi-interpretation analysis. Take a first order program. For instance,

**Example.**The following program computes "$\log_2(x+1)$":

$$\begin{array}{lrl} \texttt{log}(x) = & \textbf{case} & x\ \textbf{of} \\ & \mathbf{0} & \rightarrow \mathbf{0} \\ & \texttt{s}(x') & \rightarrow \texttt{incr}(\texttt{log}(\texttt{half}(x))) \end{array}$$

where `incr` corresponds to incrementation by one, and `half` to the "divide by two" function.

These rules are perfectly natural and we are looking for an ICC'like framework that fits with that kind of recursion. Coming back to the quasi-interpretation method, there is (at first sight) no hope, since the schema does not verify the subterm property (the redex is a subterm of the reduct).
The point is that we "have" to say at one point that `half` is a function whose results are sub-terms of the inputs. In that case, the analysis of [10,2,3] can

be carried on. So, the main purpose of the contribution is an ICC analysis of functions whose results are sub-terms of the inputs.

A major contribution in the area is due to Jones [6]. He considers programs where it is precluded to build intermediate data which would not be sub-term of the input. To do that, he considers a (imperative) programming language where the instruction $\mathtt{Z} := \mathtt{cons}(\mathtt{E}_1, \ldots, \mathtt{E}_k)$ is not authorized. According to whether we allow or not recursive calls, he gets a characterization of LOGSPACE or of PTIME. On the functional programming side, there is a strong link between the former analysis and the one of Wadler, known as "deforestation techniques", see [13]. The present contribution sheds some new light on the subject.

So, we propose a characterization of LOGSPACE. It is obtained with respect to a kind of tiering technique. This fruitful approach has been initially considered by Bellantoni and Cook in [1] and Leivant and Marion [7] who characterized PTIME. Leivant and Marion showed that such a stratification could be used for other complexity classes, see [8,9]. Following Bellantoni-Cook, Mairson and Neergaard [11] have shown what restrictions on the language B lead to LOGSPACE. Niggl and Wunderlich consider the crucial question of imperative programming [12].

The second characterization we propose deals with nondeterminism. We show, and the result is surprising, that adding some choice command in the language `WHILE` of Jones does not change the class of computed functions. Naively, one would have expected to characterize NPTIME. Indeed, if one considers — as does Jones — the class of functions computed in polynomial time in `WHILE`, one gets PTIME. Adding the choice command, one gets NPTIME. Since `WHILE`-cons-free programs characterize PTIME, adding the choice command "should" have resulted in NPTIME. It is not the case, and we show that such a system characterize PTIME. The result is all the more surprising as for the corresponding space characterization, that is of LOGSPACE, adding the choice command leads to the corresponding non-deterministic complexity class NLOGSPACE. We mention here the work of Cook [4] whose characterization of PTIME by means of auxiliary pushdown automata is in essence close to us. The main difference lies in the fact that we have an implicit call stack (for recursion) where Cook has an explicit one.

## `FOFP`

We define a generic first order functional programming language. The vocabulary $\Sigma = \langle \mathit{Cns}, \mathit{Op}, \mathit{Fct} \rangle$ is composed of three disjoint domains of symbols.

The set of programs is defined by the following grammar.

$$\begin{array}{lll}
\texttt{Programs} \ni \mathbf{p} & ::= & d_1, \cdots, d_m \\
\texttt{Definitions} \ni d & ::= & \texttt{f}(x_1, \cdots, x_n) = e^{\texttt{f}} \\
\texttt{Expression} \ni e & ::= & x \mid \mathbf{op}(e_1, \cdots, e_n) \mid \texttt{f}(e_1, \cdots, e_n) \\
& & \mid \mathbf{c}(e_1, \cdots, e_n) \\
& & \mid \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 \\
& & \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \\
& & \mid \mathbf{case}\ x_1, \cdots, x_n\ \mathbf{of}\ \overline{p_1} \to e_1 \ldots \overline{p_\ell} \to e_\ell \\
\texttt{Patterns} \ni p & ::= & x \mid \mathbf{c}(p_1, \cdots, p_n)
\end{array}$$

where $x \in \mathit{Var}$ is a variable, $\mathbf{c} \in \mathit{Cns}$ is a constructor, $\mathbf{op} \in \mathit{Op}$ is an operator, $\texttt{f} \in \mathit{Fct}$ is a function symbol, and $\overline{p_i}$ is a set of patterns.

**Definition.**We say that a program $\mathbf{p}$ is cons-free if the definitions do not use the rule $\mathbf{c}(e_1, \cdots, e_n)$ of the grammar. In other words, constructors occur only in patterns. The set of such cons-free programs is denoted $\texttt{FOFP}^{\text{cons-free}}$.

**Definition.** A definition $\texttt{f}(x_1, \cdots, x_n) = e^{\texttt{f}}$ induces a relation on function symbols. Say that $\texttt{f}$ *calls* $\texttt{g}$, denoted $\texttt{f} \to \texttt{g}$, if $\texttt{g}$ appears in the body of $\texttt{f}$. The reflexive-transitive closure of $\to$ induces a pre-order on function symbols, denoted $\xrightarrow{*}$. The corresponding equivalence relation $\simeq$ is defined by $\texttt{f} \simeq \texttt{g} \Leftrightarrow (\texttt{f} \xrightarrow{*} \texttt{g} \wedge \texttt{g} \xrightarrow{*} \texttt{f})$. The corresponding strict partial order is denoted $\prec$. We have $\texttt{g} \prec \texttt{f} \Leftrightarrow (\texttt{f} \xrightarrow{*} \texttt{g} \wedge \neg(\texttt{f} \xrightarrow{*} \texttt{g}))$.

**Definition.**(Linear programs) Given a function symbol $\texttt{f}$, the level of an expression is given by the inductive rules:

- $\texttt{lvl}_{\texttt{f}}(x) = 0$,
- $\texttt{lvl}_{\texttt{f}}(\texttt{g}(e_1, \cdots, e_n)) = 1 + \sum_{k \leq n} \texttt{lvl}_{\texttt{f}}(e_k)$ where $\texttt{g} \simeq \texttt{f}$,
- $\texttt{lvl}_{\texttt{f}}(\texttt{g}(e_1, \cdots, e_n)) = \sum_{k \leq n} \texttt{lvl}_{\texttt{f}}(e_k)$ where $\texttt{g} \prec \texttt{f}$,
- $\texttt{lvl}_{\texttt{f}}(\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2) = \texttt{lvl}_{\texttt{f}}(e_1) + \texttt{lvl}_{\texttt{f}}(e_2)$,
- $\texttt{lvl}_{\texttt{f}}(\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3) = \texttt{lvl}_{\texttt{f}}(e_1) + \max(\texttt{lvl}_{\texttt{f}}(e_2), \texttt{lvl}_{\texttt{f}}(e_3))$,
- $\texttt{lvl}_{\texttt{f}}(\mathbf{case}\ \overline{x}\ \mathbf{of}\ \ \overline{p_1} \to e_1, \ldots, \overline{p_\ell} \to e_\ell) = \max(\texttt{lvl}_{\texttt{f}}(e_1), \ldots, \texttt{lvl}_{\texttt{f}}(e_k))$.

We say that a definition $\mathtt{f}(\overline{x}) = e^{\mathtt{f}}$ is linear if $\mathtt{lvl}_{\mathtt{f}}(e^{\mathtt{f}}) = 1$. A program is linear if any definition has level 1. The set of such programs is denoted $\mathtt{FOFP}^{lin}$.

**Theorem.** Decision problems decided by linear cons-free programs are exactly Logspace decision problems.

## Non-determinism

To WHILE, we add a new command choose. We propose non-confluence as a functional correspondence of this instruction.
We consider here some FOFP programs without the confluence property, that is, patterns may overlap each other. A normal form is one possible result of the computation. Following Grädel and Gurevich [5], the value of any term is the maximal normal form of the term (for a given order on terms). Notice that this includes the usual definition for decision problem by choosing **true** > **false**. We add the superscript $n$ to denote the fact that we include non-deterministic programs.

**Theorem.**

1. $\mathtt{WHILE}^{n-cons-free} = \mathtt{FOFP}^{n-lin-cons-free} = \text{NLogspace}$
2. $\mathtt{WHILE}^{n-rec-cons-free} = \mathtt{FOFP}^{\text{n-cons-free}} = \text{Ptime}$

The latter is surprising as it breaks a similarity (that holds for logspace):

$$\begin{tikzcd}
\mathtt{WHILE}^{n-rec-cons-free} \arrow[r, phantom, "\neq"] \arrow[d, "="] & \mathtt{WHILE}^{n-ptime} \arrow[r, phantom, "="] \arrow[d] & \text{NPtime} \arrow[d] \\
\mathtt{WHILE}^{rec-cons-free} \arrow[r, phantom, "="] & \mathtt{WHILE}^{ptime} \arrow[r, phantom, "="] & \text{Ptime}
\end{tikzcd}$$

This result is analogous to that of Cook [4] Th.2 p7. He gives a characterization of Ptime by means of auxiliary pushdown automata working in logspace, that is, a Turing Machine working in logspace plus an extra (unbounded) stack. It is also the case that the result holds whether or not the auxiliary pushdown automata is deterministic.

[1] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[2] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. On lexicographic termination ordering with space bound certifications. In *PSI 2001, Ershov Memorial Conference*, volume 2244 of *Lecture Notes in Computer Science*. Springer, Jul 2001.

[3] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-Interpretations and Small Space Bounds. In J. Giesl, editor, *Rewrite Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 150–164. Springer, Apr. 2005.

[4] S. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, January 1971.

[5] E. Grädel and Y. Gurevich. Tailoring recursion for complexity. *Journal of Symbolic Logic*, 60(3):952–969, Sept. 1995.

[6] N. D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theoretical Computer Science*, 228:151–174, 1999.

[7] D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19(1,2):167,184, September 1993.

[8] D. Leivant and J.-Y. Marion. Predicative functional recurrence and poly-space. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97, Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 369–380. Springer, Apr 1997.

[9] D. Leivant and J.-Y. Marion. A characterization of alternating log time by ramified recurrence. *Theoretical Computer Science*, 236(1-2):192–208, Apr 2000.

[10] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In *LPAR 2000*, volume 1955 of *Lecture Notes in Computer Science*, pages 25–42. Springer, Nov 2000.

[11] P. Neergaard. A functional language for logarithmic space. In L. Springer-Verlag, editor, *In Proc. 2nd Asian Symp. on Prog. Lang. and Systems (APLAS 2004)*, November 2004.

[12] K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear / polynomial space for imperative programs. *SIAM J. Computing*, 35(5):1122–1147, 2006. published electronically.

[13] P. Wadler. Deforestation: Transforming programs to eliminate trees. In *ESOP '88. European Symposium on Programming, Nancy, France, 1988 (Lecture Notes in Computer Science, vol. 300)*, pages 344–358. Berlin: Springer-Verlag, 1988.

# Terms and Operads

**Michael Brinkmeier**

Institute for Theoretical Computer Science
Technical University Ilmenau
`mbrinkme@tu-ilmenau.de`
`http://www.tu-ilmenau.de/fakia/mbrinkme.html`

***Abstract.***

The talk gives an outline of [2] in which a strong connection between $\lambda$-terms and operads is proved.
In short, an operad describes a family of composable operations with multiple inputs and one output, satisfying several intuitive properties like associativity of composition and permutability of the inputs. Speaking more algebraic, operads are a generalization of categories, allowing morphisms with multiple inputs.
If one compares the definition of operads with inductive definitions of terms over a signature, the parallels become apparent. Hence one may ask, whether these two formalisms are related with each other. The aim of [2] is to give an answer to this question. In fact, there it is proved that terms are isomorphic to free algebras over free operads, and that term graphs play a vital role in this relation.
In a second step, in [2] we extend the approach to simply typed $\lambda$-terms. We achieve this goal by adding function-types and *adjunctions*, inspired by the adjunctions of cartesian closed categories. One important (proposed) consequence is hidden behind the presented results. The rewriting of $\lambda$-terms and their term graphs can be realized as rewriting of "standard" terms and term graphs. This again would imply that all techniques and results about standard terms can be extended to $\lambda$-terms. Due to the length of the paper, we do not give a precise formal formulation of this "result" there. But the following survey may make the proposition more plausible.
The talk starts with a definition of *operads* and related concepts, like algebras and morphisms of operads and algebras. Basically, an operad is a family of (typed) composable operations. It describes the basic properties of the

composition and - in combination with algebras - allows a computational semantics. To model the use of variables, our notion of (extended) operads differs slightly from that introduced in [3] (and the related notion of PROPs in [1]). Firstly, we use types or colors, restricting the possible compositions. Secondly, we extend the permutation of inputs to arbitrary assignments, i.e. assigning an output to one or more inputs, or even none. These changes require adaptions of the notions of homomorphisms and algebras to the new kind of structure. Nonetheless, the basic approach and intuition behind our operads coincides with that of "classical" operads.

Rooted, directed, acyclic graphs (RDAGs) are used to describe the free operad generated by a (typed) signature. Since RDAGs are also used as term graphs, the connection between both approaches becomes apparent. In fact, the relations applied to the RDAGs to obtain the free operad correspond to sharing and garbage collection in the context of term graphs.

The connection is even stronger, since we can prove that the typed terms over a signature $\Sigma$ on a set $X$ of typed variables is an algebra over the free operad $\mathbf{F}\Sigma$ generated by $\Sigma$. It is even isomorphic (as an algebra) to the free algebra $\mathbf{F}\Sigma X$. Furthermore, we prove that the term substitution translates to the composition of operations in the operad.

The concept of relations on operads and the resulting quotient operads provide useful tools, which allow us to describe operads in terms of generators (operations of a signature) and relations (equations of terms). Again the connection to term graphs is apparent. The identification via an equivalence relation is comparable to rewrite rules for term graphs and the known computational and formal techniques may be applied.

Up to this point, nothing really new is presented. We just transfer the known concepts of signatures, terms, term graphs, rewriting (i.e. relations) etc. into the (more general) world of operads. But the introduction of an additional structure on operads, leads to $\lambda$-*operads*. These are operads with two additional features. They contain the operad of operads, providing us with types of functions, operations for the composition and evaluation of these and "reassignments" of their inputs/arguments. In addition they allow us to interpret terms as functions, making variables into inputs. This is done via a map $\lambda$, called *adjunction*[1]. Obviously, this resembles the $\lambda$-abstraction, but instead of binding only one variable, all free variables are bound at the same time. But our approach resembles the definition of functions in functional programming, where a term is simply reinterpreted as a function. Further-

[1]The name *adjunction* indicates, that it is a transfer of the concept of adjunction in categories to operads.

more, we describe the free $\lambda$-operad $\mathbf{F}_\lambda\Sigma$ generated by a typed signature $\Sigma$ as a quotient of the free operad $\mathbf{F}\Sigma$.
We prove that the set $\lambda\text{Terms}_{\Sigma,X}$ of $\lambda$-terms over a signature $\Sigma$ on a set $X$ of variables is an algebra of the free $\lambda$-operad $\mathbf{F}_\lambda\Sigma$ generated by $\Sigma$. Furthermore we prove, completely along the line of standard terms, that $\lambda\text{Terms}_{\Sigma,X}$ as an algebra is isomorphic to the free algebra $\mathbf{F}_\lambda\Sigma X$ generated by $X$. Moreover, we give a direct representation of the binding of one variable in the world of operads and prove its direct connection with the usual $\lambda$-abstraction.
As already mentioned, this approach allows $\lambda$-terms to be represented by standard term graphs over an enriched signature. This is implied by the fact that the free $\lambda$-operad can be represented as a quotient of the free operad generated by the induced $\lambda$-signature. Since the relations correspond to rules for term rewriting, this implies that the $\beta$- and $\eta$-reductions of $\lambda$-calculus correspond to (sequences of) "standard" rewrite rules. This is caused by the fact that it is not necessary to differentiate between free and bound variables, since only the free ones occur explicitly. The bound variables are completely contained in the types.
The motivation for this paper was the author's observation, that the formalism of simply typed terms can easily be embedded into the world of operads. As the work on this "translation" proceeded, more and more ideas of extensions evolved. The results of one of them, the extension of the formalism to $\lambda$-terms and the $\lambda$-calculus, is presented here. Others, like the incorporation of recursion as an operation on function types, the usage of polymorphic types via an additional "operad of types" may be worked out in detail in the future.

[1] J.M. Boardman and R.M.Vogt. *Homotopy invariant algebraic structures on topological spaces*, volume 347 of *Lecture Notes in Mathematics*. Springer, 1973.

[2] Michael Brinkmeier. Operads and terms. Technical report, Technical University Ilmenau, 2003.
`http://www.tu-ilmenau.de/fakia/mbrinkmepubs.html`.

[3] J.P. May. *The geometry of iterated loop spaces*, volume 271 of *Lecture Notes in Mathematics*. Springer, 1972.

[4] Martin Markl, Steve Shnider, and Jim Stasheff. *Operads in Algebra, Topology and Physics*, volume 96 of *Mathematical Surveys and Monographs*. AMS, 2002.

# An alternative correctness proof of a certification method for FPTIME

**Folke Eisterlehner**

(joint work with Karl-Heinz Niggl)
TU Ilmenau
`eisfol@web.de`
`niggl@tu-ilmenau.de`

***Abstract.***
Extensive research on implicit characterization of complexity classes has been done since Bellantoni and Cook investigated the concept of *normal* and *safe* variables in the context of *function algebras* [1]. In earlier work of Kristiansen and Niggl [2,3], a measure $\mu$ on *imperative programs* is defined that in particular certifies programs which run in polynomial time. However, that measure is not applicable to programs with "arbitrary" basic instructions, such as size-increasing instructions other than e.g. `push` in the context of stack programs, or assignment statements.
Recent work of Niggl and Wunderlich [4] extends these ideas to programs with arbitrary basic instructions. The pivot is a matrix calculus $\mathsf{M}$ for certifying "polynomial size-boundedness" that reflects the way program variables may interact during the execution of a program.
A similar approach by Jones and Kristiansen [5] aims at "*a better understanding of the relationship between syntactical constructions in natural programming languages and the computational resources required to execute the programs*" by introducing a proof calculus with an adequate imperative programming language for certifying *polynomial size-boundedness* of programs.
For a given program `C`, the proof calculus tries to keep track of how the variables used in `C` influence each other by stepwise constructing *certificates*, which mainly represent labeled, directed graphs built up from the program's variable identifiers.
For those certificates, restrictions are formulated which ensure *polynomial size-boundedness* for all variables used in the program.
Like the measure $\mu$ or the method $\mathsf{M}$, the proof calculus identifies *control circles* between variables as potential cause for super-polynomial growth.

An essential achievement, however, is the distinction between different types of control circles, one of which is admissible for polynomial-size bounded programs. These control-circles correspond to circles in the graph represented by a program's certificate, and lead to difficulties in proving soundness of the calculus.
In our alternative correctness proof for loop statements, assuming program variables among $\mathtt{X_1}$ ,...,$\mathtt{X_n}$ , we identify all variables lying on such a control-circle by defining an equivalence relation for all $i, j \in \{1, \ldots, n\}$:

$$\mathtt{X_i} \overset{\alpha}{=} \mathtt{X_j} \iff i = j \text{ or } \mathtt{X_i} \text{ and } \mathtt{X_j} \text{ are lying on a circle, labeled with kind } \alpha \text{ only.}$$

Thus, the control graph is partitioned into it's strong components with respect to label $\alpha$.
We show that if the body of a loop statement has a certificate and satisfies the criteria formulated by the proof calculus, then every variable in the component $[\mathtt{X_i}]_\alpha$ can be bounded by a single polynomial $W_{[i]_\alpha}$. By defining a *partial ordering* "$\preceq$" over the strong components $[\mathtt{X_1}]_\alpha, \ldots, [\mathtt{X_n}]_\alpha$, soundness follows by induction.
Based on that simplified proof, the calculus is implemented such that both bounding polynomials are extracted and graphs representing certificates can be displayed.

[1] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of polynomial time. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, May 1992.

[2] L. Kristiansen and K.-H. Niggl. On the computational complexity of imperative programming languages. *Theor. Comput. Sci.*, 318(1-2):139–161, 2004.

[3] K.-H. Niggl. Control structures in programs and computational complexity. Habilitation Thesis, 2001, Ilmenau.
Available at `http://eiche.theoinf.tu-ilmenau.de/~niggl`.

[4] K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear / polynomial space for imperative programs. *SIAM Journal on Computing*, 35(5):1122–1147, March 2006.

[5] L. Kristiansen and N. D. Jones. The flow of data and the complexity of algorithms. In *New Computational Paradigms: First Conference on Computability in Europe, CiE 2005*, volume 3526 of *Lecture Notes in Computer Science*, pages 263–274, Amsterdam, The Netherlands, jun 8–12 2005. Springer-Verlag.

# $\lambda$-Calculus and Soft Linear Logic

**Marco Gaboardi**

(joint work with Simona Ronchi Della Rocca)
Dipartimento di Informatica, Università di Torino
`gaboardi@di.unito.it`
`ronchi@di.unito.it`

***Abstract.***
Soft Linear Logic (SLL) [2] is a subsystem of second-order linear logic with restricted rules for exponentials, which is complete for polynomial time algorithms, if cut elimination is the computation model.
A term calculus for SLL was proposed in [1]. We study instead the problem of assigning formulas of this logic to terms of pure lambda calculus, in such a way that the good computational properties of type assignment and the good complexity properties of SLL are preserved.
A standard assignment like the following[1], which we call SLL$_\lambda$,

$$\frac{}{x : U \vdash_L x : U}\ (Id) \qquad \frac{\Gamma, x : U \vdash_L M : V}{\Gamma \vdash_L \lambda x.M : U \multimap V}\ (\multimap R)$$

$$\frac{\Gamma \vdash_L M : U \quad \Delta, x : U \vdash_L N : V \quad \Gamma\#\Delta}{\Gamma, \Delta \vdash_L N[M/x] : V}\ (cut)$$

$$\frac{\Gamma \vdash_L M : U \quad x : V, \Delta \vdash_L N : Z \quad \Gamma\#\Delta \quad y \text{ fresh}}{\Gamma, y : U \multimap V, \Delta \vdash_L N[yM/x] : Z}\ (\multimap L)$$

$$\frac{\Gamma \vdash_L M : U}{!\Gamma \vdash_L M :!U}\ (sp) \qquad \frac{\Gamma, x_0 : U, ..., x_n : U \vdash_L M : V}{\Gamma, x :!U \vdash_L M[x/x_0, ..., x/x_n] : V}\ (m)$$

$$\frac{\Gamma, x : U[V/\alpha] \vdash M : Z}{\Gamma, x : \forall\alpha.U \vdash M : Z}\ (\forall L) \qquad \frac{\Gamma \vdash M : U}{\Gamma \vdash M : \forall\alpha.U}\ (\forall R)\ \alpha \text{ not free in } \Gamma$$

[1] $\Gamma\#\Delta$ denotes $dom(\Gamma) \cap dom(\Delta) = \emptyset$, $\alpha$ is not free in $\Gamma$ in rule ($\forall R$).

gives rise to some problems, in particular to the failure of subject-reduction. This means that there exists a term $M$ and a derivation $\Pi$ with conclusion $\Gamma \vdash_L M : A$ such that $M \to_\beta M'$ but we have no derivation with conclusion $\Gamma \vdash_L M' : A$. For example, consider the term $M \equiv y((\lambda z.sz)w)((\lambda z.sz)w)$ and a derivation $\Pi$ in SLL$_\lambda$ ending as follows:

$$\dfrac{\dfrac{}{s : B \multimap !A, w : B \vdash_L (\lambda z.sz)w : !A} \qquad \dfrac{\dfrac{}{y : A \multimap A \multimap B, r : A, m : A \vdash_L yrm : B}}{y : A \multimap A \multimap B, x : !A \vdash_L yxx : B}}{y : A \multimap A \multimap B, s : B \multimap !A, w : B \vdash_L y((\lambda z.sz)w)((\lambda z.sz)w) : B}$$

Clearly

$$y((\lambda z.sz)w)((\lambda z.sz)w) \to_\beta y(sw)((\lambda z.sz)w)$$

but unfortunately, there exists no derivation in SLL$_\lambda$ with conclusion:

$$y : A \multimap A \multimap B, s : B \multimap !A, w : B \vdash_L y(sw)((\lambda z.sz)w) : B$$

The problem is that if ! modality represents both a possible duplication and a performed duplication, then types like $B \multimap !A$ take something non-duplicated and return something which could be duplicated. In fact, for the term $(\lambda z.sz)w$ we deduce in $\Pi$ the type $!A$, and so we substitute it for the two occurrences of variable $x$ in the term $yxx$. Now what we expect is that we have performed a duplication, but instead we have performed sharing operation. Indeed the variables in the context of $s : B \multimap !A, w : B \vdash_L (\lambda z.sz)w : !A$ are not duplicated in the substitution.

To overcome the above problem, we propose a type system STA, inspired by the strict type assignment of intersection types, which preserves both the good properties of typed lambda calculus, such as subject reduction and normalization, and correctness and completeness for polynomial time computation.

The Soft Type Assignment System (STA) is obtained by restricting the set of formulas of SLL to the set $\mathbf{T}$ of *soft types*:

$$\begin{array}{ll} A ::= a \mid \sigma \multimap A & \qquad \text{(Linear Types)} \\ \sigma ::= A \mid !\sigma \mid \forall\alpha.\sigma & \end{array}$$

and by restricting the rules of SLL$_\lambda$:

$$\frac{}{x : A \vdash x : A}\ (Id) \qquad \frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\ (\multimap R) \qquad \frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M :!\sigma}\ (sp)$$

$$\frac{\Gamma \vdash M : A \quad \Delta, x : A \vdash N : \sigma \quad \Gamma\#\Delta}{\Gamma, \Delta \vdash N[M/x] : \sigma}\ (cut)$$

$$\frac{\Gamma \vdash M : \tau \quad x : A, \Delta \vdash N : \rho \quad \Gamma\#\Delta \quad y \text{ fresh}}{\Gamma, y : \tau \multimap A, \Delta \vdash N[yM/x] : \rho}\ (\multimap L)$$

$$\frac{\Gamma, x_1 : \tau, ..., x_n : \tau \vdash M : \sigma}{\Gamma, x :!\tau \vdash M[x/x_1, ..., x/x_n] : \sigma}\ (m) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma[\rho/\alpha] \vdash M : \mu}{\Gamma, x : \forall\alpha.\sigma \vdash M : \mu}\ (\forall L) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall\alpha.\sigma}\ (\forall R)\ \alpha \text{ not free in } \Gamma$$

For the above system, Subject Reduction Property holds, so for each term $M$ such that $\Gamma \vdash M : \mu$ and $M \to_\beta M'$, there exists a derivation for $\Gamma \vdash M' : \mu$. STA could seem weaker than SLL, in fact ($cut$) rule and ($\multimap L$) rule in STA can be viewed as restrictions to linear formulas of the ones in SLL$_\lambda$. Nevertheless, we have a generalized ($cut$) rule of the shape:

$$\frac{\Gamma \vdash M : \sigma \quad x : \sigma, \Delta \vdash N : \mu \quad \Gamma\#\Delta}{\Gamma, \Delta \vdash N[M/x] : \mu}\ (G - cut)$$

and furthermore, it can be shown that typability power of STA is equivalent to that one of SLL$_\lambda$.
SLL enjoys the good property that its proofs correspond to polynomial time algorithms, and that polynomial time algorithms can be represented by SLL proofs. The same can already be shown for STA. In particular, as in Lafont [2], we have that each term typable in STA is reducible to its normal form with a number of steps bounded by a measure which is exponential in the degree of the type derivation. So computing with input of fixed degree is polynomial in the dimension of the input.
Furthermore, as in Mairson and Terui [3], we have that STA with the only connective $\multimap$ and the quantifier $\forall$, and maintaining Lafont distinction between program as term typable with generic derivations (derivations where the rule ($m$) is not used) and data as term typable with homogeneous derivations (derivations where the rule ($m$) is used each time with the same rank), is complete for polynomial time algorithms.

Let as in [2] $N\langle P\rangle = \forall\alpha.(\alpha \multimap \alpha)^P \multimap \alpha \multimap \alpha$. Due to non-uniformity of soft types, to show completeness, we need to assign to the term $\lambda c_p.\lambda c_q.\lambda s.c_p(c_q s)$ representing multiplication the type $\mathbf{N}\langle P\rangle \multimap (\mathbf{N}\langle Q\rangle)^P \multimap \mathbf{N}\langle PQ\rangle$. This fact imposes a slight modification of the notion of representable function by allowing the possibility of defining functions from nonlinear types. Based on that modification, completeness is established.
So we have that STA is a type assignment system for pure lambda calculus with good computational and complexity properties.

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

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

[3] Harry G. Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. In *ICTCS*, pages 23–36, 2003.

# Strong WQO phase transitions[1]

**Lew Gordeev**

(joint work with A. Weiermann)
Tübingen-Utrecht
`gordeew@informatik.uni-tuebingen.de`
`weierman@math.uu.nl`

***Abstract.***

**Summary.** We elaborate phase transitions for Gordeev's well-quasi-ordering (called *well-partial-ordering* below, abbr.: *wpo*) results with respect to nested finite sequences and nested finite trees under the homeomorphic embedding with symmetrical gap condition. For every nested partial ordering in question, $\trianglelefteq$, we fix a natural extension of Peano Arithmetic, $\mathsf{T}$, that proves the corresponding 2-order sentence $SPQ\,(\trianglelefteq)$. Furthermore, we consider the appropriate parameterized 1-order *slow well-partial-ordering* sentence $SWPO\,(\trianglelefteq, r)$ with $r$ ranging over computable reals and show that for some computable real $\alpha$, the following holds.

1. If $r < \alpha$ then $SWPO\,(\trianglelefteq, r)$ is provable in $\mathsf{PA}$.
2. If $r > \alpha$ then $SWPO\,(\trianglelefteq, r)$ is not provable in $\mathsf{T}$.

In the limit cases we replace computable reals $r$ by computable functions $f : \mathbb{N} \to \mathbb{R}$ and prove analogous theorems.
These results strengthen both Kruskal-Friedman-Kriz wqo theorems and Weiermann's phase transition elaboration of basic Kruskal-Friedman-Schütte-Simpson cases.

## Preliminaries (1-D case)

### *Partial and linear well orderings*

- By $\trianglelefteq$ and $\leq$ we denote partial and linear countable well orderings (abbr.: *wpo* and *wo*), respectively. A wo $\mathcal{O} = (W, \leq)$ is called a *linearization* of a wpo $\mathcal{W} = (W, \trianglelefteq)$ iff $(\forall x, y \in W)\,(x \trianglelefteq y \to x \leq y)$. A

[1]Research supported by NWO grant 03-721.

wpo $\mathcal{W} = (W, \trianglelefteq)$ is called *enumerated* iff it is supplied with a bijection, also called *enumeration*, $\nu : \mathbb{N} \to W$. For any enumerated wpo $\mathcal{W} = (W, \nu, \trianglelefteq)$, we fix its lexicographical linearization $\mathcal{W}_\nu = (W, \leq_\nu)$ that is defined as follows [2]

$$\begin{aligned} W \times W \ni x \leq_\nu y :\Leftrightarrow (\forall i \in \mathbb{N}) (\nu(i) \trianglelefteq x \longleftrightarrow \nu(i) \trianglelefteq y) \vee (\exists i \in \mathbb{N}) \\ (\nu(i) \ntrianglelefteq x \wedge \nu(i) \trianglelefteq y \wedge (\forall j < i) (\nu(j) \trianglelefteq x \longleftrightarrow \nu(j) \trianglelefteq y)) \end{aligned}$$

- Let $\textsc{Seq}(X) := X^{<\omega} := X \cup X^2 \cup \cdots \cup X^m \cup \cdots = \bigcup_{m=1}^{\infty} \underbrace{X \times \cdots \times X}_{m}$ and for any $k > 0$ let $\textsc{Seq}_k(X) := X^{\leq k} := X \cup X^2 \cup \cdots \cup X^k \subset \textsc{Seq}(X)$. Let

$$\begin{aligned} \textsc{Seq}^0 &:= \{\bullet\}, \textsc{Seq}^1 := \textsc{Seq}\left(\textsc{Seq}^0\right) \cong \{\bullet, \bullet\bullet, \bullet\bullet\bullet, \cdots\}, \\ \textsc{Seq}^1_k &:= \textsc{Seq}_k\left(\textsc{Seq}^0\right) \cong \left\{\underbrace{\bullet, \bullet\bullet, \bullet\bullet\bullet, \cdots}_{k}\right\} \subset \textsc{Seq}^1, \cdots, \\ \textsc{Seq}^{d+1} &:= \textsc{Seq}\left(\textsc{Seq}^d\right), \\ \textsc{Seq}^{d+1}_k &:= \textsc{Seq}\left(\textsc{Seq}^d_k\right) \subset \textsc{Seq}^{d+1}, \cdots. \end{aligned}$$

  For any $x \in \textsc{Seq}^d$, we let $\varrho^d(x) := \min\left\{k \mid x \in \textsc{Seq}^d_k\right\}$.

- We define norm functions $\#^d : \textsc{Seq}^d \to \mathbb{N}$ by recursion on $d$ via

$$\#^{d+1}(\langle x_1, \cdots, x_m \rangle) := \sum_{i=1}^{m} \#^d(x_i)$$

  where $\#^0(\bullet) := 1$.

- For every $d, k > 0$, we fix an arbitrary primitive recursive enumeration $\nu_{d,k} : \mathbb{N} \to \textsc{Seq}^d_k$ (it might just as well be the corresponding nested lexicographical tuple-ordering).

## *Basic definition*

For every $d > 0$, define a wpo $\left(\textsc{Seq}^d, \trianglelefteq_d\right)$ and the corresponding wo $\left(\textsc{Seq}^d, \leq_d\right)$ by recursion as follows.

[2] For any wpo $W = (W, \trianglelefteq)$ denote by $o(\mathcal{W})$ the supremum of order types, i.e. set theoretical ordinals, of all linearizations of $W$. For all enumerated $W = (W, \nu, \trianglelefteq)$ considered below, $W_\nu = (W, \leq_\nu)$ has the order type $o(\mathcal{W})$. This conclusion fails for just arbitrary enumerated wpo; in general maximal linearizations don't admit explicit arithmetical definitions.

Case $d = 1$. Let $\leq_1 := \trianglelefteq_1$ where

$$\textsc{Seq}^1 \times \textsc{Seq}^1 \ni x \trianglelefteq_1 x' :\Leftrightarrow \#^1(x) \leq \#^1(x')$$

Case $d = 2$.

1. Define $\trianglelefteq_2$ by

$$\begin{aligned} \textsc{Seq}^2 \times \textsc{Seq}^2 \ni \langle x_1, \cdots, x_m \rangle \trianglelefteq_2 \langle x'_1, \cdots, x'_{m'} \rangle :\Leftrightarrow & \\ (\exists 1 \leq \xi(1) < \cdots < \xi(m) \leq m)\, (\forall 1 \leq i < m) & \\ \begin{pmatrix} x_i \trianglelefteq_1 x'_{\xi(i)} \wedge x_m \trianglelefteq_1 x'_{\xi(m)} \wedge \min\limits_{\leq_1} \{x_i, x_{i+1}\} \\ \leq_1 \min\limits_{\leq_1} \left\{ x'_{\xi(i)}, \cdots, x'_{\xi(i+1)} \right\} \end{pmatrix} & \end{aligned}$$

2. For every $k > 0$ take the enumeration $\nu_{2,k} : \mathbb{N} \to \textsc{Seq}^2_k$ and consider the corresponding "local" linearization $\left(\textsc{Seq}^2_k, \leq_{\nu_{2,k}}\right)$ of $\left(\textsc{Seq}^2_k, \trianglelefteq_2\right)$.
3. Define the desired "global" linearization $\leq_2$ by

$$\begin{aligned} \textsc{Seq}^2 \times \textsc{Seq}^2 \ni x \leq_2 x' :\Leftrightarrow & \\ \varrho^2(x) < \varrho^2(x') \vee \left(\varrho^2(x) = \varrho^2(x') \wedge x \leq_{\nu_{2,k}} x'\right) & \end{aligned}$$

Case $d \mapsto d+1$. Define $\trianglelefteq_{d+1}, \leq_{d+1} \subset \textsc{Seq}^{d+1} \times \textsc{Seq}^{d+1}$ analogously.

**Remark.** Note that all relations $\trianglelefteq_d, \leq_d$ are definable in the language of $\mathsf{PA}$.

## Basic results (1-D case)

For any wpo $\mathcal{W} = (W, \trianglelefteq)$, let $\mathsf{WPO}(\mathcal{W})$ be an abbreviation of "$\mathcal{W}$ *is a wpo*" in the form $(\forall f : \mathbb{N} \to W)\,(\exists i < j \in \mathbb{N})\,(f(i) \trianglelefteq f(j))$

**Theorem.**

1. $\mathsf{WPO}\left(\textsc{Seq}^2, \trianglelefteq_2\right)$ is not provable in $\mathsf{ACA}_0$.
2. $\mathsf{WPO}\left(\textsc{Seq}^3, \trianglelefteq_3\right)$ is not provable in $\Delta^1_1\mathsf{CA}$.
3. $(\forall d > 0)\,\mathsf{WPO}\left(\textsc{Seq}^d, \trianglelefteq_d\right)$ is provable in $\mathsf{ATR}$, but not in $\mathsf{ATR}_0$.

**Definition.** For any wpo $\mathcal{W} = (W, \trianglelefteq)$, norm function $\# : W \to \mathbb{N}$, $0 < d \in \mathbb{N}$ and $0 \leq r \in \mathbb{Q}$, let $\mathsf{SWP}\,(W, \trianglelefteq, \#, d, r)$ be the corresponding first order refinement of $\mathsf{WPO}\,(\mathcal{W})$

$$(\forall K \in \mathbb{N})\,(\exists M \in \mathbb{N})\,(\forall x_0, \cdots, x_M \in W)$$
$$((\forall i \leq M)\,(\#\,(x_i) \leq K + r \cdot \lceil \log_d (i+1) \rceil) \to (\exists i < j \leq M)\,(x_i \trianglelefteq x_j))$$

$\lceil \log_d (i+1) \rceil$ is also referred to as the $d$-adic length of $i$. In the sequel we often abbreviate $\lceil \log_d (i+1) \rceil$ by $|i|_d$.

**Theorem.**

1. If $r < 1$ then $\mathsf{PA} \vdash \mathsf{SWP}\left(\textsc{Seq}^2, \trianglelefteq_2, \#_2, 2, r\right)$.

2. If $r > 1$ then $\mathsf{PA} \nvdash \mathsf{SWP}\left(\textsc{Seq}^2, \trianglelefteq_2, \#_2, 2, r\right)$.

**Theorem.**

1. If $r < 1$ then $\mathsf{PA} \vdash \mathsf{SWP}\left(\textsc{Seq}^3, \trianglelefteq_3, \#_3, 3, r\right)$.

2. If $r > 1$ then $\Delta^1_1\mathsf{CA} \nvdash \mathsf{SWP}\left(\textsc{Seq}^3, \trianglelefteq_3, \#_3, 3, r\right)$.

**Definition.** For any $f, g : \mathbb{N} \to \mathbb{Q}$, let $I\,(x) := x$ and

$$f \prec^\infty g :\Leftrightarrow |\{x \in \mathbb{N} \mid f\,(x) < g\,(x)\}| = \infty$$

**Theorem.** For any computable function $f : \mathbb{N} \to \mathbb{Q}$ the following holds.

1. If $f \prec^\infty I$ then $\forall k\,(\exists d > k)\,\mathsf{PA} \vdash \mathsf{SWP}\left(\textsc{Seq}^d, \trianglelefteq_d, \#_d, d, f\,(d)\right)$.

2. If $I \prec^\infty f$ then:

   (a) $(\exists k)\,\mathsf{ATR}_0 \nvdash (\forall d > k)\,\mathsf{SWP}\left(\textsc{Seq}^d, \trianglelefteq_d, \#_d, d, f\,(d)\right)$,

   (b) $\mathsf{ATR} \vdash (\forall d > 0)\,\mathsf{SWP}\left(\textsc{Seq}^d, \trianglelefteq_d, \#_d, d, f\,(d)\right)$.

# $T^-$-Hierarchies and the Trade-off Theorem

**Lars Kristiansen**

Oslo University College, Faculty of Engineering,
PO Box 4, St. Olavs plass, NO-0130 Oslo, Norway, and
Department of Mathematics, University of Oslo
`larskri@iu.hio.no`
`http://www.iu.hio.no/~larskri`

***Abstract.***
I will discuss two hierarchies of unknown ordinal height. Many well-known deterministic complexity classes, e.g. LOGSPACE, P, PSPACE, LINSPACE and EXP, can be found in the hierarchies. These classes are defined by imposing explicit resource bounds on Turing machines, but note that the classes are not uniformly defined as some are defined by imposing *time* bounds, whereas others are defined by imposing *space* bounds. Small subrecursive classes can also be found in our hierarchies, e.g. the relational Grzegorczyk classes $\mathcal{E}^0_*$, $\mathcal{E}^1_*$ and $\mathcal{E}^2_*$. In contrast to a complexity class, a subrecursive class is defined as the least class containing some initial functions and closed under certain composition and recursion schemes. Some of the schemes might contain explicit bounds, but no machine models are involved.
The two hierarchies are induced by neat and natural fragments of a calculus based on finite types and Gödel's $T$, and all the classes in the hierarchies are uniformly defined without referring to explicit bounds. Thus, one should not expect the hierarchies to capture such a wide variety of classes, that is, both time classes, space classes and subrecursive classes. This indicates that a further investigation of the hierarchies might be rewarding, and perhaps shed light upon some of the notoriously hard open problems involving the classes captured by the hierarchies, e.g. maybe some of these problems turn out to be related in some unexpected way. Moreover, the ingredients of the theoretic framework nourishing the hierarchies are well-known and thoroughly studied in the literature, e.g. the ordinal numbers, the typed $\lambda$-calculi, cut-elimination, rewriting systems and Gödel's $T$. Advanced and well-proven techniques of mathematical logic and computability theory will thus be available facilitating the investigations.

I aim at a fairly nontechnical talk, and I will survey the research that led up to the hierarchies ([1,2,3,4,5]). Furthermore, I will discuss our trade-off theorem recently published in [7]. The theorem being an adaption of Schwichtenberg's well-known trade-off theorem to the setting of computational complexity, sheds light upon the hierarchies and makes them better understood. (Schwichtenberg shows how to eliminate higher type levels in definitions of primitive recursive functionals by means of transfinite recursion, see e.g. [8])

[1] L. Kristiansen. Neat function algebraic characterizations of LOGSPACE and LINSPACE. Computational Complexity **14**(1) (2005) 72–88

[2] L. Kristiansen. Complexity-Theoretic Hierarchies. In: CiE 2006: Logical Approaches to Computational Barriers. Volume 3988 of LNCS., Springer-Verlag (2006) 279-288

[3] L. Kristiansen, G. Barra. The small Grzegorczyk classes and the typed $\lambda$-calculus. In: CiE 2005: New Computational Paradigms. Volume 3526 of LNCS., Springer-Verlag (2005) 252–262

[4] L. Kristiansen, P. Voda. Complexity classes and fragments of C. Information Processing Letters **88** (2003) 213–218

[5] L. Kristiansen, P. Voda. The surprising power of restricted programs and Gödel's functionals. In CSL 2003: Computer Science Logic. Volume 2803 of LNCS., Springer (2003) 345–358

[6] L. Kristiansen, P. Voda. Programming languages capturing complexity classes. Nordic Journal of Computing **12** (2005) 1–27. (Special issue for NWPT'04.)

[7] L. Kristiansen, P. Voda. The trade-off theorem and fragments of Gödel's T. In: TAMC'06: Theory and Applications of Models of Computation. Volume 3959 of LNCS., Springer-Verlag (2006) 655–674

[8] H. Schwichtenberg. Classifying recursive functions. In Griffor, E., ed.: Handbook of computability theory. Elsevier (1996) 533–586

# Sign-sensitive Graph Representations of CNFs: Conflict graphs and resolution graphs

**Oliver Kullmann**[1]

Computer Science Department
University of Wales Swansea
Swansea, SA2 8PP, UK
`O.Kullmann@Swansea.ac.uk`
`http://cs.swan.ac.uk/~csoliver`

***Abstract.***

Graph representations of propositional formulas and constraint satisfaction problems have been investigated quite intensively over the last two decades (see [4,3] and Chapter 9 in [1]), but it seems that only the *variable interaction graphs* (or *Gaifman graphs*; the variables of the structure are the vertices, joined by an edge if occurring together in the same clause/constraint) and its relatives have been investigated. These graphs do not take the signs of the literals of clause-sets into account, but they consider only the variable structure. The basis of their exploitation is the trivial fact that if we have two variable-disjoint formulas $F_1, F_2$, then the satisfying assignments for $F_1 \wedge F_2$ are the compositions of the satisfying assignments for $F_1$ and $F_2$.

The basic graph representation of clause-sets taking signs into account is the *conflict graph*, which has the clauses as vertices, joined by an edge if there exists a clashing literal pair. This graph structure has been investigated in the context of satisfiability decision in [6,2]; different from the above exploitations, here linear algebra and not graph connectedness lies at the bottom (and actually the conflict *multi*graph is basic, taking into account the number of conflicts).

Now in [4] investigations have begun into the exploitation of the conflict graph and refinements for SAT decision in the same vein as for the variable interaction graph, namely by splitting graphs into connected components.

[1]Supported by EPSRC Grant GR/S58393/01

### The resolution graph

Modulo pure literals (literals occurring in only one sign), the conflict graph offers the same splitting opportunities as the variable interaction graph (whether nevertheless the conflict graphs offer better *algorithmic* possibilities has to be investigated). Stronger splitting opportunities are given by the *resolution graph*, which eliminates all edges from the conflict graph where the corresponding resolvent would be tautological (i.e., where at least two conflicts exist). Here already the proof, that the connected components of the resolution graph can be treated independently for SAT solving, is not completely trivial (the proof was given in [5]).
In my talk I want to give an introduction to the subject, and report on progress (and difficulties encountered) regarding the following problems:

1. A clause-set is satisfiable if and only if every connected component of the resolution graph is satisfiable; however, given satisfying assignments for the components, we do not know how to compute efficiently a satisfying assignment for the whole clause-set.

2. How do resolution trees interact with the components of the resolution graph? We know that if a resolution refutation of the whole clause-set exists, then there is one which uses only the clauses of one single component — how to find such single-component refutations efficiently?

3. The variable interaction graph has a compact representation by the variable hypergraph (and its dual). Can we find similar (generalised) compact representations, which can be exploited by graph-theoretical algorithms, for the conflict graph and the resolution graph?

4. For bounded clause-length and bounded treewidth of the resolution graph, the satisfiability problem is decidable in polynomial time (and via self-reduction then we can also find a satisfying assignment in polynomial time). What about fixed parameter-tractability? Is the bounded clause-length necessary? What about hypertree decompositions, or other forms of decompositions? What about harder problems like counting satisfying assignments, or solving MAXSAT?

[1] Rina Dechter. *Constraint Processing.* Morgan Kaufmann, San Francisco, 2003. ISBN 1-55860-890-7; QA76.612.D43 2003.

[2] Nicola Galesi and Oliver Kullmann. Polynomial time SAT decision, hypergraph transversals and the hermitian rank. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing 2004*, volume 3542 of *Lecture Notes in Computer Science*, pages 89–104, Berlin, 2005. Springer. ISBN 3-540-27829-X.

[3] Georg Gottlob, Martin Grohe, Nysret Musliu, Marko Samer, and Francesco Scarcello. Hypertree decompositions: Structure, algorithms, and applications. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science: 31st International Workshop (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science (LNCS)*, pages 1–15. Springer, 2005. ISBN 3-540-31000-2.

[4] Marijn Heule and Oliver Kullmann. Decomposing clause-sets: Integrating DLL algorithms, tree decompositions and hypergraph cuts for variable- and clause-based graph representations of CNF's. Technical Report CSR 2-2006, University of Wales Swansea, Computer Science Report Series, March 2006 (`http://www-compsci.swan.ac.uk/reports/2006.html`).

[5] Oliver Kullmann. Obere und untere Schranken für die Komplexität von aussagenlogischen Resolutionsbeweisen und Klassen von SAT-Algorithmen. Master's thesis, Johann Wolfgang Goethe-Universität Frankfurt am Main, April 1992. (Upper and lower bounds for the complexity of propositional resolution proofs and classes of SAT algorithms (in German); Diplomarbeit am Fachbereich Mathematik).

[6] Oliver Kullmann. The combinatorics of conflicts between clauses. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 426–440, Berlin, 2004. Springer. ISBN 3-540-20851-8.

# Interpretations methods for proving complexity upper bounds

**Jean-Yves Marion**

Loria, Carte project, and
Ecole Nationale Supérieure des Mines de Nancy, INPL,
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
`Jean-Yves.Marion@loria.de`
`www.loria.fr/~marionjy`

***Abstract.***

Interpretation methods of proving complexity upper bounds provide a static analysis of first order functional program resources. It gives a way to determine a bound on the running time or space usage of programs or yet the number of processors in a parallel computation. Those methods give rise to characterizations of many complexity classes, however, they are designed so as to identify a broad class of algorithms computing functions in a certain complexity class, in particular algorithms using design principles such as greedy, divide and conquer, and dynamic programming. Interpretation methods also provide complexity proof certificates. The talk will focus on two methods. The first one is based on Quasi-interpretations, and the second one on Sup-interpretations, which refines the first one.

The overall objective is to provide machine independent characterizations of functional complexity classes. This line of research was initiated by Cobham [11]. One major motivation of this research line is to provide a static analysis of the computational resources needed to run a program. Such analysis should guarantee the amount of memory, time or processors, which are necessary to execute a program on all inputs.

*Implicit computational complexity (ICC)* proposes syntactic characterizations of complexity classes which lean on data ramification principles like safe recursion [5], lambda-calculus [20] or data tiering [19]. It is worthwhile to discuss the two main difficulties we have to face in order to provide a compelling resource static analysis. The first is that the method should capture a broad class of programs in order to be useful. From a theoretical perspective, this means that we are trying to identify a large class of programs

computing functions in certain complexity classes. Unlike the traditional approach, which is extensional in that the focus is on capturing all functions of a complexity class, our approach is rather intentional. This change of view is difficult because we have to keep in mind that the set of polynomial time programs is $\Sigma_2$-complete. The second difficulty is related to the complexity of the static analysis suggested. The resource analysis procedure should be decidable and easily checkable. But inversely, a too "easy" resource analysis procedure won't, certainly, delineate a meaningful class of programs.
There are at least four directions inspired by ICC approaches which are related to our topic and that we briefly review. The first direction deals with linear type systems in order to restrict computational time, and began with seminal work of Girard [14] which defined Light Linear Logic. We mention here some recent works of Baillot-Terui [4], Lafont[18], or yet Coppola-Ronchi [12]. The second direction is due to Hofmann, e.g. see [15], and independently [6], who introduced a resource atomic type, the diamond type, into the linear type system for higher order functional programming. Unlike the two former approaches, the third one considers imperative programming languages, and is developed by Kristiansen-Niggl [17], Kristiansen-Jones [16], or Niggl-Wunderlich [26], and Marion-Moyen[25,23].
Lastly, the fourth approach is the one on which we focus on in this talk. That approach is concerned with term rewriting systems and interpretation methods of proving complexity bounds based on [21]. Those methods consist in giving an interpretation to computed functions, which provides an upper bound on function output size, and analyse the program data flow in order to measure the program complexity. We have developed two kinds of interpretation methods of proving complexity. The first direction concerns Quasi-interpretations, which is surveyed in [7]. The second direction is the *sup-interpretation method* introduced in [24].
The main features of interpretation methods of proving complexity bounds are the following.

1. Interpretation methods are applicable to broad classes of algorithms, like greedy algorithms, dynamic programming [22,8,9], and are capable of dealing with nonterminating programs [24].

2. Resource verification of bytecode programs is obtained by compiling first order functional and reactive programs. See for example [3,2,13].

3. There are heuristics to determine program complexity. See [1,10]

[1] R. Amadio. Max-plus quasi-interpretations. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003, Valencia, Spain, June 10-12, 2003, Proceedings*, volume 2701 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2003.

[2] R. Amadio and S. Dal-Zilio. Resource control for synchronous cooperative threads. In *Concur*, pages 68–82, 2004.

[3] R.M. Amadio, S. Coupet-Grimal, S. Dal Zilio, and L. Jakubiec. A functional scenario for bytecode verification of resource bounds. In *CSL*, 2004. to appear.

[4] P. Baillot and K. Terui. A feasible algorithm for typing in elementary affine logic. In Springer, editor, *TLCA*, volume 3461 of *LNCS*, pages 55–70, 2005.

[5] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[6] S. Bellantoni, K-H Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1-3):17–30, 2000.

[7] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretation: a way to control ressources. *survey submitted, revision.* `http://www.loria/~marionjy`.

[8] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. On lexicographic termination ordering with space bound certifications. In *PSI 2001, Akademgorodok, Novosibirsk, Russia, Ershov Memorial Conference*, volume 2244 of *LNCS*. Springer, Jul 2001.

[9] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations and small space bounds. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2005.

[10] G. Bonfante, J.-Y. Marion, J.-Y. Moyen, and R. Péchoux. Synthesis of quasi-interpretations. *Workshop on Logic and Complexity in Computer Science, LCC2005, Chicago*, 2005. `http://www.loria/~pechoux`.

[11] A. Cobham. The intrinsic computational difficulty of functions. In *Conf. on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, 1962.

[12] P. Coppola and S. Ronchi Della Rocca. Principal typing for lambda calculus in elementary affine logic. *Fundamenta Informaticae*, 65(1-2):87–112, 2005.

[13] S. Dal-Zilio and R. Gascon. Resource bound certification for a tail-recursive virtual machine. In *APLAS 2005*, volume 3780 of *LNCS*, pages 247–263, 2005.

[14] J.-Y. Girard. Light linear logic. *Inf. and Comp.*, 143(2):175–204, 1998. pésenté à LCC'94, LNCS 960.

[15] M. Hofmann. The strength of Non-Size Increasing computation. In *Proceedings of POPL'02*, pages 260–269, 2002.

[16] L. Kristiansen and N.D. Jones. The flow of data and the complexity of algorithms. In *New Computational Paradigms*, number 3526 in LNCS, pages 263–274, 2005.

[17] L. Kristiansen and K.-H. Niggl. On the computational complexity of imperative programming languages. *TCS*, 318(1–2):139–161, 2004.

[18] Y. Lafont. Soft linear logic and polynomial time. *TCS*, 318:163–180, 2004.

[19] D. Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

[20] D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19(1,2):167,184, September 1993.

[21] J.-Y. Marion. Analysing the implicit complexity of programs. *Inf. and Comp.*, 183:2–18, 2003.

[22] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France*, volume 1955 of *LNCS*, pages 25–42. Springer, Nov 2000.

[23] J.-Y. Marion and J.-Y. Moyen. Heap analysis for assembly programs. Technical report, Loria, 2006.

[24] J.-Y. Marion and R. Pechoux. Resource analysis by sup-interpretation. In *FLOPS 2006*, volume 3945 of *LNCS*, pages 163–176, 2006.

[25] J.-Y. Moyen. *Analyse de la complexité et transformation de programmes.* Thèse d'université, Nancy 2, Dec 2003.

[26] K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear / polynomial space for imperative programs. *SIAM J. on Computing*, 35(5):1122–1147, 2006.

# Resources Control Graphs

**Jean-Yves Moyen**

LIPN-CNRS – université Paris 13
99 avenue J.-B. Clément
F-93430 Villetaneuse
`Jean-Yves.Moyen@lipn.univ-paris13.fr`

***Abstract.***

Resources Control Graphs are a tool to perform some kind of abstract interpretation of programs in order to study their properties such as termination or complexity. The key idea is to work over the Control Flow Graph of a program and add some kind of dynamic information to it that can be seen as a way to approximate the different configurations reached during computation. Thus, any actual computation can be followed on the Resources Control Graph, and studying properties of the Resources Control Graph gives information about the properties of the program itself.

Let us consider a program in some kind of imperative language that we won't need to describe in details here. In order to build its Control Flow Graph (CFG), we start by finding some control points in the program. Then, the CFG is a directed graph with one vertex per control point and one edge between $a$ and $b$ if there exists a computation of the program where the control flow goes between the control point corresponding to $a$ to the control point corresponding to $b$. In most cases, that simply means that each control point is linked to the following one, except for tests where control can go to either branch.

In order to illustrate this, let us consider as a toy example the following program, computing addition of two unary numbers and quite similar to what can be done with a counter machine:

$$\begin{array}{lcl} 0 & : & \texttt{if } x = 0 \texttt{ jmp } 4; \\ 1 & : & x--\,;\ 2\ :\ y++; \\ 3 & : & \texttt{jmp } 0\ ;\ 4\ :\ \texttt{end}; \end{array}$$

Its CFG is displayed on Figure 1. Here, the control points correspond to the labels of the program. The useful property of CFG is that any execution of

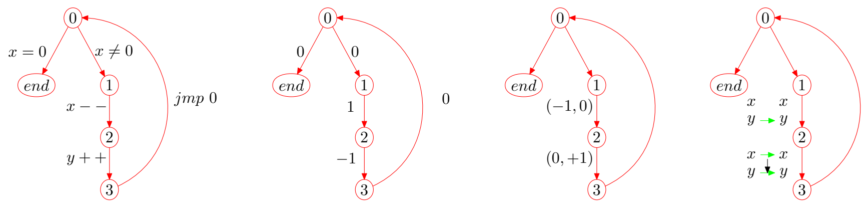

Figure 1: Control Flow Graph and Resources Control Graphs of the addition.

the program can be mapped onto a path of the CFG. The converse, however, is not true because paths of the CFG are somewhat non-deterministic in the sense that any branch of the test can be followed without taking into account the actual value of variables. For example, the computation of $3 + 5$ corresponds to the path 0123012301230`end`.

Any execution of the program can be seen as a sequence of states, each state containing both the current label and the current memory. A path in the CFG, on the other hand, only contains the sequences of vertexes, that is labels. Thus, the memory of the machine is completely forgotten in the CFG. Now, in order to study more precisely properties of programs, we want to add (part of) the forgotten information into the CFG, thus turning it into a *Resources Control Graph* (RCG).

So, instead of considering simply vertexes, we will now consider *configurations*. A configuration is a couple $(v, x)$ where $v$ is a vertex and $x$ ranges over a (potentially infinite) set called the set of *valuations*. On the more general case, valuations represent exactly the current memory, but depending on the property under consideration, it is possible to restrict the set of valuations.

Now, the notion of path in the graph turns into a notion of *walks*. A walk is a sequence of configurations $(v_1, x_1), \dots, (v_n, x_n)$. The corresponding sequence of vertexes $v_1, \dots, v_n$ forms a path. Each new valuation $x_{i+1}$ is computed from the preceding one and the edge followed by the walk.

So, to each edge of the RCG, that is to each instruction of the program, we need to define a function for computing new valuations when going through that edge. Here, again, depending on the approximation of memory chosen for the valuations, we may have different functions.

Let us illustrate this by considering only the total space usage of the program. Since our toy example works with unary numbers, the size needed to store a value $n$ is $n$ itself. So, at each state, the total space usage is $x + y$, the sum of the values of both registers. Since we chose this as an approximation of memory, valuations will be integers.

Now, we need to have functions associated to each edge that mimic, on the level of the approximation, the behaviour of the corresponding instruction. Both branches of the test, as well as the jump, keep the memory in the same state. Hence, the corresponding function is the identity. The $y$++ instruction increases the total space usage by one. Hence the corresponding function is $\lambda x.x+1$. The $x--$ instruction decreases the total space usage by one *only if* $x \neq 0$ and causes an error otherwise. However, it is more convenient to consider that the decrease always occurs, that is associate to it the function $\lambda x.x - 1$, and carry the error on the level of the RCG.

Now, we can notice that all the functions associated to edges have the form $\lambda x.x + \alpha$ where $\alpha$ is some constant. Hence, it is more convenient to only represent them by the constant $\alpha$. The corresponding RCG is thus a classical weighted graph as displayed on Figure 1.

Again, we have the same simulation property. That is, any execution of the program corresponds to a walk in the RCG. For example, the execution of $3 + 5$ corresponds to the walk (0, 8), (1, 8), (2, 7), (3, 8), (0, 8), . . . , (0, 8), (`end`, 8).

But now, problems may occur because we've chosen to carry the errors of the subtraction on the RCG. Consider the following: $(0, 0), (1, 0), (2, -1)$. It is a walk that obviously cannot correspond to any execution because space usage cannot be negative. At this point, we would like the valuation to be restricted to only *positive* integers. However, this is not really satisfactory from a theoretical point of view, since it is more convenient to have the set of valuations closed by the functions associated to each edge.

So, in order to circumvent this problem, we split the set of valuations and only consider some of them to be *admissible*. Similarly, a configuration will be admissible only if its valuation is admissible and a walk is admissible if all configurations in it are. Admissible valuations are those who really correspond to an approximation of the memory, not those added by the closure. Now, the previously mentioned walk is not admissible and can thus be banned.

The important point is that executions of the program are now mapped onto *admissible walks* only. So by only considering admissible walks, we do not miss anything. Of course, there is still the problem of detecting non-admissible walks.

Let us have a more closer look at the set of admissible walks in our case. Since the RCG does not contain any cycle of strictly positive weight (this is decidable in polynomial time via Bellman-Ford algorithm), any configuration $(v, x)$ reached in a walk starting from $(v_0, x_0)$ will have $x \leq x_0$. We say that the RCG is *resource aware*. Since, when mapping an execution of the program onto a walk of the RCG, the valuation corresponds to the total space usage, this means that when executing the program, one will never need more space than what is initially allocated. That is, the program is *Non-Size Increasing* [2].

Of course, only considering the total space usage is a big approximation performed on memory. One may want to do something closer to the actual memory. Typically, one may want to consider the space usage of each variable independently. In this case, instead of a single integer, the valuations will range over vectors of integers with as many components as the number of variables, given a suitable enumeration of the variables. In our case, there are two variables and we will consider the vector $(|x|, |y|)$.

What will be the weight corresponding to the instructions in this case? Again, both branches of the test and the jump do not change the variables and hence have the identity as a weight. The $x{-}{-}$ instruction decreases the size (value) of the first component without changing the size of the second component, hence its weight is $\lambda x.x + (-1, 0)$ (where vector-addition is performed component-wise). Similarly, the $y{+}{+}$ instruction has weight $\lambda x.x + (0, +1)$. Again, since weights all have the form $\lambda x.x + \alpha$ for some constant $\alpha$, we can identify them with $\alpha$, and the resulting RCG is a Vector Addition System with States (VASS) [5]. The VASS for the addition is displayed on Figure 1.

Once more, every execution of the program can be mapped onto an admissible walk of the VASS. Here, the set of admissible valuations is exactly $\mathbb{N}^n$. For example, the execution of $3 + 5$ corresponds to the walk (0, (3, 5)), (1, (3, 5)), (2, (2, 5)), (3, (2, 6)), (0, (2, 6)), ..., (2, (0, 7)), (3, (0, 8)), (0, (0, 8)), (`end`, (0, 8)).

But now, if we look at the VASS, we can see that it admits no infinite admissible walk. The existence of an infinite admissible walk for a VASS is a decidable property. Since every execution is mapped onto an admissible walk, the absence of infinite admissible walk means that there are no infinite executions of the program, that is, it is uniformly terminating.

However, using VASS as RCG to model programs is still not very convenient because one cannot model that way several useful instructions. Among other things, a usual copy instruction like $x := y$ cannot be represented that way. However, keeping the idea of having vectors representing the size of each variable as valuations, it then becomes quite easy to perform such copies by a matrix multiplication.

This is a rather appealing direction since matrices multiplications have already been used to study termination by Abel and Altenkirch [1] in their re-reading of the Size Change Principle [3]. And actually, it turns out that the SCP can indeed be expressed in terms of RCG where the weights are the size change graphs. The corresponding RCG for our addition program is depicted on Figure 1. The resulting termination criterion is equivalent to the so called "graph algorithm" of Lee, Jones and Ben Amram.
Using Abel and Altenkirch ideas, one can also express each size change graph (in the RCG) by a matrix. This leads to some kind of Matrices Multiplication Systems with States. But matrices multiplication has already been used by Niggl and Wunderlich [4] in order to detect polynomial time computations. Apparently, Niggl and Wunderlich criterion should also be expressible in terms of RCG.
So, the main interest of RCG is to be able to express within the same formalism several different existing program analysis. Moreover, some program transformation techniques such as the deforestation also seem to be easy to represent using Resource Control Graphs. This might lead to the building of some kind of global tool to perform program analysis.

[1] A. Abel and T. Altenkirch. A Predicative Analysis of Structural Recursion. *Journal of Functional Programming*, 12(1):1–41, January 2002.

[2] M. Hofmann. Linear types and Non-Size Increasing polynomial time computation. In *Proceedings of the Fourteenth IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 464–473, 1999.

[3] C. S. Lee, N. D. Jones, and A. Ben-Amram. The Size-Change Principle for Program Termination. In *POPL'01*, volume 28, pages 81–92. ACM press, January 2001.

[4] K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear / polynomial space for imperative programs. *SIAM Journal on Computing*, 35(5):1122–1147, March 2006. published electronically.

[5] C. Reutenauer. *Aspects mathématiques des réseaux de Petri.* Masson, 1989.

# Interpolation in Horn logic

**Gerard Renardel de Lavalette**

Gerard R. Renardel de Lavalette
Department of Mathematics and Computing Science
University of Groningen, the Netherlands
`g.r.renardel.de.lavalette@rug.nl`

***Abstract.***

In this note, we investigate interpolation in the Horn fragment of propositional infinitary logic, where conjunctions of arbitrary size are allowed. We use abstract derivations that are related to the derivations for equational logic we introduced in [2] to prove interpolation for equational logic. In the future, we intend to combine the methods of [2] and this note in a proof of interpolation for conditional equational logic and other logics.

**1. Horn formulae and interpolation.** Let $\mathsf{PR}$ be a collection of propositional atoms, denoted by $p, q, r, \ldots$. We consider the collection $\mathsf{Horn}$ of formulae $\varphi = \varphi_I$ defined by

$$\varphi_I = \bigwedge_{i \in I} (\bigwedge P_i \to p_i)$$

where $I$ is some index set with $p_i \in \mathsf{PR}$ and $P_i \subseteq \mathsf{PR}$ for every $i \in I$. The subformulae $\bigwedge P_i \to p_i$ are called the *rules* of $\varphi$: if $P_i = \emptyset$, then $p_i$ is called a *fact* of $\varphi$, otherwise $\bigwedge P_i \to p_i$ is a proper rule. Observe that, in general, we do not require $I$ and the $P_i$ to be finite. We identify $\varphi_\emptyset$ with $\mathsf{T}$. $\mathsf{atom}(\varphi)$ denotes the collection of atoms occurring in $\varphi$.
$\mathsf{Horn}$ satisfies (polynomial) interpolation if, for every $\varphi_1, \varphi_2, \varphi_3 \in \mathsf{Horn}$ with $\varphi_1, \varphi_2 \vdash \varphi_3$, there is an interpolant $\psi \in \mathsf{Horn}$ (with size polynomial in $\varphi_1, \varphi_2, \varphi_3$), i.e.

$$\varphi_1 \vdash \psi \qquad \psi, \varphi_2 \vdash \varphi_3 \qquad \mathsf{atom}(\psi) \subseteq \mathsf{atom}(\varphi_1) \cap (\mathsf{atom}(\varphi_2) \cup \mathsf{atom}(\varphi_3))$$

It is an easy exercise to show that this is equivalent to the apparently weaker version where $\varphi_3$ is an atom (use that $\mathsf{Horn}$ is closed under conjunction and under taking subformulae). Since $\mathsf{Horn}$ is not closed under $\rightarrow$, the present formulation of interpolation is stronger than the version without $\varphi_2$. $\psi$ is called a *uniform* interpolant if it does not depend on $\varphi_2$ and $\varphi_3$, but only on $\varphi_1$ and the set of atoms allowed in $\psi$. In the sequel, we sketch a proof that

$\mathsf{Horn}$ satisfies polynomial interpolation.

**2. Two unsuccessful attempts.** The first idea that comes to mind for proving interpolation is the Maehara-Schütte method (after [5], [6]) that extracts an interpolant from a cut-free derivation in a sequent calculus. Unfortunately, this does not work for $\mathsf{Horn}$: there is e.g. a cut-free derivation of $p \wedge (q \rightarrow r), p \rightarrow q \vdash r$ that yields the interpolant $(p \rightarrow q) \rightarrow r$ which is not in $\mathsf{Horn}$.
A second attempt starts with F. Ville's observation (see [3](Ch. 1, exercise 2)) that, in classical propositional logic, uniform interpolation can be proved using the fact that

$$\varphi(p) \vdash \varphi(\varphi(\mathsf{T}))$$

(all occurrences of $p$ are shown in $\varphi(p)$). An interpolant $\psi$ for $\varphi_1, \varphi_2, \varphi_3$ is obtained as follows: start with $\varphi_1$ and eliminate one by one all atoms that occur in $\varphi_1$ but not in $\varphi_2$ or $\varphi_3$. It is not evident that this will work for $\mathsf{Horn}$, since it is not closed under substitution, but it turns out that the uniform interpolant obtained this way is logically equivalent to a Horn formula. However, the size of a uniform interpolant is not polynomial in the size of $\varphi_1$. This is shown in the following example. Let

$$\varphi_n = (\bigwedge_{i<n} p_i \rightarrow p) \wedge \bigwedge_{i<n} (q_i \rightarrow p_i) \wedge \bigwedge_{i<n} (r_i \rightarrow p_i)$$

then the uniform interpolant $\psi_n$ for $\varphi_n$ with

$$\mathsf{atom}(\psi_n) \subseteq \{p, q_0, \ldots, q_{n-1}, r_0, \ldots, r_{n-1}\}$$

is

$$\psi_n = \bigwedge_{I \subseteq \{0,\ldots,n-1\}} ((\bigwedge_{i \in I} q_i \wedge \bigwedge_{i<n, i \notin I} r_i) \rightarrow p)$$

It is evident that the size of $\varphi_n$ is linear in $n$, while the size of $\psi_n$ is exponential.

**3. Abstract derivations.** An (abstract) derivation is a structure $D = \langle K, B, R, \sigma\rangle$ with $K \neq \emptyset$, $B \subseteq K$, $R \subseteq K^2$ is a wellfounded relation, and $\sigma : K \to \mathsf{PR}$. $B, R$ represent the Horn formula $\varphi_{B,R}$ defined by

$$\varphi_{B,R} = \bigwedge_{k\in B} \sigma(k) \wedge \bigwedge_{k\in \mathsf{rg}(R)} (\bigwedge_{lRk} \sigma(l) \to \sigma(k))$$

Given $L \subseteq K$, the collection $\mathsf{cons}(L)$ of direct consequences of $L$ in $D$ is defined as $\mathsf{cons}(L) = \{k \in \mathsf{rg}(R) \mid Rk \subseteq L\}$ ($Rk$ is shorthand for $\{l \mid lRk\}$). The collection of nodes that are *derivable* in $D$ from $L$ is defined by $\mathsf{der}(L) = \mu M.L \cup B \cup \mathsf{cons}(M)$. We say that $D$ represents a derivation of $\varphi_{B,R} \vdash \varphi_I$ if the following holds:

$$\forall i \in I \exists k \in K \exists L \subseteq K(\sigma(k) = p_i \ \&\ \sigma[L] = P_i \ \&\ k \in \mathsf{der}(L))$$

i.e. for every rule $\bigwedge P_i \to p_i$ in $\varphi_I$ there are a node $k$ representing $p_i$ and a set of nodes $L$ representing $P_i$ such that $k \in \mathsf{der}(L)$.
Now let $\varphi_1, \varphi_2, \varphi_I \in \mathsf{Horn}$ satisfy $\varphi_1, \varphi_2 \vdash \varphi_I$, and let $D = \langle K, B, R, \sigma\rangle$ be a derivation representing this. So there are $B_1, B_2, R_1, R_2$ satisfying $B = B_1 \cup B_2$, $R = R_1 \cup R_2$, $\mathsf{rg}(R_1) \cap \mathsf{rg}(R_2) = \emptyset$, $\varphi_1 = \varphi_{B_1,R_1}$, $\varphi_2 = \varphi_{B_2,R_2}$, and for every $i \in I$ there are $k_i \in K$ and $L_i \subseteq K$ with $\sigma(k_i) = p_i$, $\sigma[L_i] = P_i$ and $k_i \in \mathsf{der}(L_i)$. We indicate how an interpolant for $\varphi_1, \varphi_2 \vdash \varphi_I$ can be found. Define $C \subseteq K$ and $S \subseteq K^2$ by

$$\begin{array}{rcl} C & = & B_1 \cap \mathsf{dom}(R_2) \\ S & = & R_1^+ \cap (B_2 \cup \mathsf{rg}(R_2)) \times (\mathsf{dom}(R_2) \cup \{k_i \mid i \in I\}) \end{array}$$

where $R_1^+$ denotes the transitive closure of $R_1$. We claim that $\varphi_{C,S}$ is an interpolant for $\varphi_1, \varphi_2 \vdash \varphi_I$ ($\varphi_1 \vdash \varphi_{C,S}$ follows from $\varphi_1 = \varphi_{C_1,R_1} \vdash \varphi_{C_1,R_1^+}$; the proof of $\varphi_{C,R}, \varphi_2 \vdash \varphi_I$ is more involved). However, we know nothing about the size of $\varphi_{C,S}$, only that it is polynomial (in fact at most quadratic) in the size of the derivation $D$. So the question is: what can we say about the size of $D$?

**4. Minimal derivations.** A minimal derivation is a derivation where $\sigma$ is injective, i.e. different nodes represent different atoms. So we may identify $K$ with a subset of $\mathsf{PR}$, and we can skip $\sigma$. The size of a minimal derivation representing $\varphi_1, \varphi_2 \vdash \varphi_3$ with $K = \mathsf{atom}(\varphi_1, \varphi_2, \varphi_3)$ is linear in the size of $\varphi_1, \varphi_2 \vdash \varphi_3$. We claim that any derivation $D = \langle K, B, R, \sigma\rangle$ representing $\varphi_1, \varphi_2 \vdash p$ can be transformed in a minimal derivation $D' = \langle \sigma[K], \sigma[B], R'\rangle$

that satisfies

$$\forall q \in \mathsf{rg}(R')\exists k \in K(q = \sigma(k) \,\&\, R'q = \sigma[Rk])$$

and hence represents $\varphi_1, \varphi_2 \vdash p$, too. $R'$ is obtained as $\bigcup_{\alpha \leq \kappa} R'_\alpha$ where $\alpha$ runs over the ordinals and $\kappa$ is the cardinality of $\mathsf{PR}$; we use that $\mathsf{der}(\emptyset) = \bigcup_{\alpha \leq \kappa} \mathsf{cons}^\alpha(\emptyset)$ (approximation of least fixpoint from below). For the definition of $R'_{\alpha+1}$, we need a choice function $\lambda$ for the fact that

$$\begin{aligned}&\forall \alpha \leq \kappa \;\; \forall p \in \sigma[\mathsf{cons}^{\alpha+1}(\emptyset)] - \sigma[\mathsf{cons}^{\alpha}(\emptyset)] \;\; \exists k \in \mathsf{rg}(R)\\ &\qquad (\sigma(k) = p \,\&\, Rk \subseteq \mathsf{cons}^\alpha(\emptyset))\end{aligned}$$

so $\lambda(\alpha, p) \in \mathsf{rg}(R) \,\&\, \sigma(\lambda(p,\alpha)) = p \,\&\, R\lambda(p,\alpha) \subseteq \mathsf{cons}^\alpha(P)$. Now $R'_{\alpha+1}$ is defined by

$$R'_{\alpha+1} = \{(q,p) \mid p \in \sigma[\mathsf{cons}^{\alpha+1}(\emptyset)] - \sigma[\mathsf{cons}^{\alpha}(\emptyset)], q \in \lambda(\alpha,p)\}$$

Using minimal derivations, we can obtain polynomial interpolation: if $\varphi_1, \varphi_2 \vdash \varphi_I$ then $\varphi_1, \varphi_2 \wedge \bigwedge P_i \vdash p_i$ for all $i \in I$; via minimal derivations we obtain polynomial interpolants $\psi_i$, and their conjunction $\bigwedge_{i\in I} \psi_i$ is a polynomial interpolant for $\varphi_1, \varphi_2 \vdash \varphi_I$.
This ends our proof sketch. Our interpolation theorem generalizes Theorem 10 in [1]. Moreover, the interpolant we found satisfies the Lyndon conditions (after [4])

$$\begin{aligned}\mathsf{atom}^+(\psi) &\subseteq \mathsf{atom}^+(\varphi_1) \cap (\mathsf{atom}^-(\varphi_2) \cup \mathsf{atom}^+(\varphi_3))\\ \mathsf{atom}^-(\psi) &\subseteq \mathsf{atom}^-(\varphi_1) \cap (\mathsf{atom}^+(\varphi_2) \cup \mathsf{atom}^-(\varphi_3))\end{aligned}$$

where $\mathsf{atom}^+(\varphi_I) = \{p_i \mid i \in I\}$, $\mathsf{atom}^-(\varphi_I) = \bigcup\{P_i \mid i \in I\}$.

[1] E. Dahlhaus, A. Israeli, and J.A. Makowsky. On the existence of polynomial time algorithms for interpolation problems in propositional logic. *Notre Dame Journal of Formal Logic*, 29:497–509, 1988.

[2] Gerard R. Renardel de Lavalette. Abstract derivations, equational logic and interpolation (extended abstract). In Paola Bruscoli, François Lamarche, and Charles Stewart, editors, *Structures and Deduction — the Quest for the Essence of Proofs*, pages 173–188. Technische Universität Dresden, Fakultät Informatik, 2005. (see also `http://www.cs.rug.nl/~grl/pub/lisbon2005full.pdf`).

[3] G. Kreisel and J.-L. Krivine. *Elements of Mathematical Logic (Model Theory)*. North-Holland, Amsterdam, 1967.

[4] R.C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific Journal of Mathematics*, 9:129–142, 1959.

[5] S. Maehara. On the interpolation theorem of Craig. *Sugaku*, 12:235–237, 1960. (Japanese).

[6] Kurt Schütte. Der Interpolationssatz der intuitionistischen Prädikatenlogik. *Mathematische Annalen*, 148:192–200, 1962.

# Implicit characterizations of FPTIME and NC revisited

**Henning Wunderlich**

(joint work with Karl-Heinz Niggl)
Technische Universität Ilmenau,
Institut für Theoretische Informatik,
Helmholtzplatz 1, 98684 Ilmenau,
`henning.wunderlich@tu-ilmenau.de`

***Abstract.***

In implicit computational complexity, much attention has been payed to the complexity classes FPTIME and NC, e.g. see [3,5,2]. This talk presents simplified or improved, and partly corrected well-known implicit characterizations of the complexity classes FPTIME and NC. The core of the present research is to simplify the required simulations of various bounded recursion schemes in the corresponding implicit framework, and moreover, to develop those simulations in a more uniform way. Furthermore, we establish a new ground type function algebraic characterization of NC, which might be of help to resolve the open problem of characterizing NC through higher types. The starting point is a simplified proof that Cobham's class Cob [7] characterizing FPTIME is contained in the function algebra BC of Bellantoni and Cook [3]. That every function in Cob can be simulated in BC essentially rests on two findings:

(S1) For every function $f \in$ Cob one can construct a function $f' \in$ BC, called *simulation* of $f$, and a polynomial $p_f$ we call *witness* for $f$ such that

$$f(\vec{x}) = f'(w;\ \vec{x}) \text{ whenever } |w| \geq p_f(|\vec{x}|).$$

(S2) Every polynomial $p$ can be *length-bounded* by a function $W_p \in$ BC, that is, $|W_p(\vec{x})| \geq p(|\vec{x}|)$.

The proof that every $f \in$ Cob can be simulated in BC is then concluded as follows: The safe-to-normal property built into the class BC allows one to write both $f'$ and $W_p$ as functions $\mathrm{SN}(f')(w, \vec{x};\ )$ and $\mathrm{SN}(W_p)(\vec{x};\ )$ of normal

variables only. Therefore, (S1) and (S2) imply that $f$ can be defined in BC by safe composition:

$$f(\vec{x}) \quad = \quad \mathrm{SN}(f')(\mathrm{SN}(W_p)(\vec{x};\ ), \vec{x};\ )$$

In each simulation, we will concentrate on the crucial statement corresponding to (S1). As for (S1), all cases are obvious, except for the case $f = \mathrm{BRN}(g, h_0, h_1, j)$, where a difficult simulation and proof was given in [3]. The difficulty mainly arises because of an unnatural choice of a *case* function defined as

$$\mathrm{case}(\ ; x, \mathit{even}, \mathit{odd}) := \begin{cases} \mathit{even} & x \text{ is even} \\ \mathit{odd} & x \text{ is odd.} \end{cases}$$

When replacing function case by the function bcase (for binary case) which naturally springs from (bounded) recursion on notation, that is,

$$\mathrm{bcase}(\ ; x, \mathit{zero}, \mathit{even}, \mathit{odd}) := \begin{cases} \mathit{zero} & x = 0 \\ \mathit{even} & x > 0 \text{ and } x \text{ is even} \\ \mathit{odd} & x > 0 \text{ and } x \text{ is odd} \end{cases}$$

then a simulation $f'$ can be constructed, the correctness of which is immediate from its definition. To see this, let $g', h'_0, h'_1 \in \mathrm{BC}$ be given by the induction hypothesis. For legibility, given an input $y = (b_{l-1} \cdots b_0)_2$, with $b_{l-1} \cdots b_0$ being the binary representation of $y$, we write $y\{i\}$ for the *y-section* $(b_{l-1} \cdots b_i)_2$. Thus, on input $y = (b_{l-1} \cdots b_0)_2$ we want to simulate

$$\begin{array}{rcll} f(y, \vec{a}) & = & f(y\{0\}, \vec{a}) & \\ & = & h_{b_0}(y\{1\}, \vec{a}, & \text{step 1} \\ & & \quad \ddots & \vdots \\ & & \qquad h_{b_{i-1}}(y\{i\}, \vec{a}, & \text{step i} \\ & & \qquad\quad \ddots & \vdots \\ & & \qquad\qquad h_{b_{l-1}}(0, \vec{a}, & \text{step l} \\ & & \qquad\qquad\quad g(\vec{a})) \cdots ) \cdots ) & \text{step l+1} \end{array}$$

step by step via $f'$, where $f'$ should be of the form $f'(w;\ y, \vec{a}) := \hat{f}(w, w;\ y, \vec{a})$. The function $\hat{f}(\hat{w}, w;\ y, \vec{a})$ is defined by safe recursion on notation (srn) on $\hat{w}$, being of the form $\hat{f} := \mathrm{srn}(0, \hat{h}, \hat{h})$. The new recursion parameter $\hat{w}$ must be introduced because $y$, being the recursion parameter of $f$, must be in a safe position in $f'$. Inspecting the required step by step simulation, we see that $\hat{h}(\hat{w}, w;\ y, \vec{a}, v)$ must simulate *step* $\mathrm{S}(\hat{w}, w) := |w| \dot{-} |\hat{w}|$ in the recursion of $f(y, \vec{a})$. Using the given simulations $g', h'_0, h'_1$ we conclude that

$$\hat{h}(\hat{w}, w;\ y, \vec{a}, v) = \begin{cases} h'_{y\{\mathrm{S}(\hat{w},w)\dot{-}1\}}(w;\ y\{\mathrm{S}(\hat{w}, w)\}, \vec{a}, v) & \text{if } 1 \leq \mathrm{S}(\hat{w}, w) \leq |y| \\ g'(w;\ \vec{a}) & \text{if } \mathrm{S}(\hat{w}, w) > |y|. \end{cases}$$

Defining $\ominus(u;\ v) := \mathrm{p}^{|u|}(v)$ by srn on $u$, we obtain the BC function

$$Y(\hat{w}, w;\ y) := \ominus(\ominus(\hat{w};\ w);\ y) = \mathrm{p}^{|w| \dot{-} |\hat{w}|}(y) = \mathrm{p}^{\mathrm{S}(\hat{w},w)}(y) = y\{\mathrm{S}(\hat{w}, w)\}.$$

Now, observing that for $y = (b_{l-1} \cdots b_0)_2$,

$$b_{\mathrm{S}(\hat{w},w)\dot{-}1} = 0 \iff Y(\mathrm{s}_1(\hat{w}), w;\ y) \text{ is even}$$

the required function $\hat{h}$ can be defined as follows

$$\begin{aligned} \hat{h}(\hat{w}, w;\ y, \vec{a}, v) := \mathrm{bcase}(\ ;\ & Y(\mathrm{s}_1(\hat{w}), w;\ y), \\ & g'(w, \vec{a}), \\ & h'_0(w;\ Y(\hat{w}, w;\ y), \vec{a}, v), \\ & h'_1(w;\ Y(\hat{w}, w;\ y), \vec{a}, v)) \end{aligned}$$

As mentioned above, correctness of this simulation follows immediately from its definition, leading to the following variant of the Bellantoni/Cook theorem:

$$\mathrm{FPTIME} = [0, \mathrm{s}_b, \mathrm{p}, \pi_i^{m,n}, \mathrm{bcase}; \mathrm{scomp}, \mathrm{srn}]$$

Now we consider characterizations of the complexity class NC. In [5] Clote established NC = CLO for the function algebra

$$\mathrm{CLO} := [0, \mathrm{S}_b, \Pi_i^m, \mathrm{LEN}, \mathrm{BIT}, \#; \mathrm{COMP}, \mathrm{CRN}, \mathrm{WBRN}].$$

In [4], Bellantoni noticed that CLO = CLO′ for the function algebra

$$\mathrm{CLO}' := [0, \mathrm{S}_b, \Pi_i^m, \mathrm{LEN}, \mathrm{BIT}, \#; \mathrm{COMP}, \mathrm{CRN}, \mathrm{WBRN}'],$$

where WBRN′ is WBRN but based on Bellantoni's *half* function $H$, and sketched the proof in a footnote. As the proof sketch does not give the desired result, we give a corrected proof of CLO = CLO′. Furthermore, we consider the following function algebras:

$$\begin{aligned} \mathrm{2CLO} &:= [0, \mathrm{s}_b, \pi_i^{m,n}, \mathrm{len}, \mathrm{bit}, \#_{\mathrm{Bel}}, \mathrm{case}; \mathrm{scomp}, \mathrm{scrn}, \mathrm{slr}] \\ \mathrm{2NC} &:= [0, \mathrm{s}_b, \pi_i^{m,n}, \mathrm{len}, \mathrm{bit}, \#_{\mathrm{Bel}}, \mathrm{case}, \mathrm{half}, \mathrm{drop}; \\ &\qquad \mathrm{scomp}, \mathrm{scrn}_{\mathrm{AJST}}, \mathrm{slr}] \\ \mathrm{2NC}' &:= [0, \mathrm{s}_b, \pi_i^{m,n}, \mathrm{len}, \mathrm{bit}, \mathrm{sm}, \#_{\mathrm{AJST}}, \mathrm{case}, \mathrm{half}, \mathrm{drop}; \\ &\qquad \mathrm{scomp}, \mathrm{scrn}_{\mathrm{AJST}}, \mathrm{slr}] \\ \mathrm{NC}' &:= [0, \mathrm{s}_b, \pi_i^{m,n}, \mathrm{len}, \mathrm{bit}, \mathrm{sm}, \#_{\mathrm{AJST}}, \mathrm{case}, \mathrm{msp}; \\ &\qquad \mathrm{scomp}, \mathrm{scrn}', \mathrm{slr}] \end{aligned}$$

2CLO was defined in [4], 2NC was implicitly defined in [1]. The idea to split the smash function $\#_{\mathrm{Bel}}$ into two parts can be found in [2]. We call this algebra 2NC′. The class NC′, treated in [10], contains fewer base functions, and uses the following variant of concatenation recursion on notation $f = \mathrm{scrn}'(h_0, h_1)$:

$$\begin{array}{rcl} f(0, \vec{x};\ \vec{y}) & = & 0 \\ f(\mathrm{s}_b(\ ; y), \vec{x};\ \vec{y}) & = & \mathrm{s}_{h(\vec{x};\ \mathrm{s}_b(\ ;y), \vec{y}) \bmod 2}(f(y, \vec{x};\ \vec{y})) \end{array}$$

Unlike the scheme in [2], $\mathrm{s}_b(\ ; y)$ appears in a safe position in $h$, which is more restrictive. Again, replacing function case by its natural variant bcase, and proceeding as in the above described alternative characterization of FPTIME, we show that all these function algebras characterize NC. For this, we embed NC = CLO′ in a uniform way into 2CLO, ..., NC′ (now with bcase) using simplified simulations. As above, correctness of those simulations can be seen right from their definitions.

[1] K. Aehlig, J. Johannsen, H. Schwichtenberg, and S. Terwijn. Linear ramified higher type recursion and parallel complexity. Technical Report 17, Mittag-Leffler-Institut, 2000/2001.

[2] K. Aehlig, J. Johannsen, H. Schwichtenberg, and S. Terwijn. Linear ramified higher type recursion and parallel complexity. In Kahle et al. [8], pages 1–21.

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

[4] S. J. Bellantoni. *Predicative Recursion and Computational Complexity.* PhD thesis, Graduate Department of Computer Science, University of Toronto, 1992.

[5] P. Clote. Sequential, machine-independent characterizations of the parallel complexity classes *alogtime*, *ac*$^k$, *nc*$^k$ and *nc*. In *MSI Workshop on Feasible Mathematics.* Birkhäuser, 1989.

[6] P. Clote. Computational models and function algebras. In Leivant [9], pages 98–130.

[7] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. North Holland, 1964.

[8] R. Kahle, P. Schroeder-Heister, and R. F. Stärk, editors. *Proof Theory in Computer Science, International Seminar, PTCS 2001, Dagstuhl Castle, Germany, October 7-12, 2001, Proceedings*, volume 2183 of *Lecture Notes in Computer Science.* Springer, 2001.

[9] D. Leivant, editor. *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13-16 October 1994*, volume 960 of *Lecture Notes in Computer Science*. Springer, 1995.

[10] H. Wunderlich. Syntaktische Charakterisierungen effizient berechenbarer Funktionen, 2003. Diploma thesis.

# Substructural Logics in Natural Deduction

**Ernst Zimmermann**
ernstzimmermann@de.ibm.com

***Abstract.***

**Substructures in Sequents and Deductions**

From the beginning, substructural logics were almost exclusively devoted to proof theoretic structures of the calculus of sequents. Structural rules like weakening and contraction are intimately linked to sequents, and so it was to be expected that investigations from various directions and with various motivations would undertake the task to define logics below the use of structural rules in the calculus of sequents.
The use and the importance of calculi of sequents in the realm of substructural logic contrast with the weight which the founder of the subject gave to it as a whole. Gentzen spoke about deductions as the form of natural logical and mathematical reasoning, and about sequents only as a form of a heuristic metamathematical means.
Disregarding any attitude towards the one or the other type of logical reasoning, one decisive difference should be apparent when the two types of calculi are compared. Sequents allow proof theoretic interpretations of intuitionistic logic and of classical logic in a simple form and to full generality, whereas deductions seem to be restricted in a certain sense to intuitionistic logic. Clearly, deductions allow interpretations of classical logic too, but only with a loss of some highly desired proof theoretic properties like subformula property.
Against the background of the situation sketched, the present talk does the following. Since structural rules like weakening and contraction are easily deducible in the natural deduction of intuitionistic logic, the talk explores what natural deduction without weakening and contraction looks like. This exploration leads to a certain fragment of linear logic, which can be extended either by an explicit contraction rule to relevant logic or by an explicit weakening rule to BCK logic.

Nevertheless, the present investigations are limited to substructural logics of intuitionistic logic, since in the opinion of the author this is the realm of logic described by means of natural deduction. But in this realm it is shown that the well-known normalisation methods and normalisation results of Prawitz for intuitionistic predicate logic can be fully generalised to fragments of intuitionistic linear, relevant and BCK predicate logic. These fragments include usual implication and falsum and additive connectives conjunction and disjunction, which obey contexts of assumptions. Given these connectives, extensions to quantifiers are trivial. To make conversions work for conjunction and disjunction, completely symmetric rules are defined: the well-known pairs of symmetric rules for conjunction elimination and disjunction introduction, but even pairs of symmetric rules for conjunction introduction and disjunction elimination, where equal contexts of assumptions are explicitely mentioned and partially discharged. As a consequence, normalisation procedures like conversions and permutations have to be worked out in more detail, i.e. usual conversions and permutations have to be refined. The pairs of symmetric rules for context sharing connectives presented in this talk deviate from other rules already present in the literature (Negri, 2002, Martins / Martins, 2004). These symmetric rules for connectives which share contexts of assumptions and the explicit rules for weakening and contraction show that substructural logics are a subject not reserved for the calculus of sequents.

## Rules and Calculi

The *language of propositional logic.* There are propositional variables $\mathbf{P}_k$ for $k \in \omega$. $\bot$ is a constant; $\to, \wedge, \vee$ are binary connectives; $(,)$, the parentheses, are auxiliary symbols.
The *formulas*: $\mathbf{P}_k$ are formulas; $\bot$ is a formula; $(A \to B), (A \wedge B), (A \vee B)$ are formulas, if $A$ and $B$ are formulas.

## An Intuitionistic Linear Logic ILL

$A \qquad BR$

$$\dfrac{\begin{matrix}\vdots & \vdots \\ A & A \to B\end{matrix}}{B} \to E \qquad\qquad \dfrac{\begin{matrix}A^u \\ \vdots \\ B\end{matrix}}{A \to B} \to Iu$$

$$\dfrac{\begin{matrix}\vdots \\ A \wedge B\end{matrix}}{A} \wedge E \qquad\qquad \dfrac{\begin{matrix}\vdots \\ A \wedge B\end{matrix}}{B} \wedge E$$

$$\dfrac{\begin{matrix}\Gamma\\ \vdots\\ A\end{matrix} \quad \begin{matrix}[\Gamma]^v\\ \vdots\\ B\end{matrix}}{A\wedge B}\ \wedge Iv \qquad\qquad \dfrac{\begin{matrix}[\Gamma]^v\\ \vdots\\ A\end{matrix} \quad \begin{matrix}\Gamma\\ \vdots\\ B\end{matrix}}{A\wedge B}\ \wedge Iv$$

$$\dfrac{\begin{matrix}\vdots\\ A\end{matrix}}{A\vee B}\ \vee I \qquad\qquad \dfrac{\begin{matrix}\vdots\\ B\end{matrix}}{A\vee B}\ \vee I$$

$$\dfrac{\begin{matrix}\\ \vdots\\ A\vee B\end{matrix} \quad \begin{matrix}A^u \quad [\Gamma]^w\\ \vdots\\ C\end{matrix} \quad \begin{matrix}B^v \quad \Gamma\\ \vdots\\ C\end{matrix}}{C}\ \vee Euvw$$

$$\dfrac{\begin{matrix}\\ \vdots\\ A\vee B\end{matrix} \quad \begin{matrix}A^u \quad \Gamma\\ \vdots\\ C\end{matrix} \quad \begin{matrix}B^v \quad [\Gamma]^w\\ \vdots\\ C\end{matrix}}{C}\ \vee Euvw$$

$$\dfrac{\begin{matrix}\vdots\\ \bot\end{matrix}}{A}\ \bot R \qquad\qquad \dfrac{\begin{matrix}\vdots\\ \bot\end{matrix} \quad \begin{matrix}\vdots\\ A\end{matrix}}{\bot}\ W\bot$$

$\Gamma$ are multisets, $A^w$ singletons.

## An Intuitionistic Relevant Logic IRL

ILL Rules in the following interpretation: $\Gamma, A^w$ are sets, $A^w \neq \emptyset$.

## An Intuitionistic Relevant Logic $\mathbf{IRL_C}$

ILL Rules and Contraction Rule:

$$\dfrac{\begin{matrix}A^u \quad A\\ \vdots\\ B\end{matrix}}{B}\ Cu$$

**An Intuitionistic BCK Logic IBL**

ILL Rules in the following interpretation: $\Gamma$ are multisets, $A^w$ singletons or $A^w = \emptyset$.

**An Intuitionistic BCK Logic IBL$_\mathbf{W}$**

ILL Rules and Weakening Rule:

$$\dfrac{\begin{matrix}\vdots & \vdots \\ B & A\end{matrix}}{B}\ W$$

[1] K. Dosen, P. Schroeder-Heister (Eds.), 1993, *Substructural Logics.* Clarendon Press, Oxford.

[2] G. Gentzen, 1934/35, *Untersuchungen ueber das logische Schliessen I, II.* In: Mathematische Zeitschrift, 39. 176-210, 405-431.

[3] J.-Y. Girard, 1987, *Linear Logic.* In: Theoretical Computer Science, 50, 1-102.

[4] L.R. Martins, A.T. Martins, 2004, *Natural Deduction and Weak Normalisation of Full Linear Logic.* In: Logic Journal of IGPL, 12, 601-625.

[5] S. Negri, 2002, *A normalizing system of natural deduction for intuitionistic linear logic.* In: AML, 41, 789-810.

[6] H. Ono, Y. Komori, 1985, *Logics without Contraction Rule.* In: JSL. 169-201.

[7] D. Prawitz, 1965, *Natural Deduction. A Proof-Theoretical Study.* Almquist and Wiksell, Stockholm.

[8] D. Prawitz, 1971, *Ideas and Results in Proof Theory.* In: J.E. Fenstad (Ed.), Proc. of 2-nd Scand. Logic Symposium. Amsterdam. 235-307.

[9] N. Tennant, 2004, *Relevance in Reasoning.* In: S. Shapiro (Ed.), Handbook of the Philosophy of Logic and Mathematics, Oxford.

[10] A.S. Troelstra, 1992, *Lectures on linear logic.* CSLI Lecture Notes 29. Stanford, California.

[11] A.S. Troelstra, 1995, *Natural deduction for intuitionistic linear logic.* In: APAL.

[12] A.S. Troelstra, H. Schwichtenberg, 1996, *Basic Proof Theory.* Cambridge University Press, Cambridge.

## Monday, July 24th

| | |
|---|---|
| 9:15 | *Registration & Opening*<br><br>Jean-Yves Marion<br>*Interpretations methods for proving complexity upper bounds* |
| | *Coffee Break* |
| 10:45 | Guillaume Bonfante<br>*Life after “life without cons”*<br><br>Jean-Yves Moyen<br>*Resources Control Graphs* |
| | *Lunch Break* |
| 13:45 | Ernst Zimmermann<br>*Substructural Logics in Natural Deduction*<br><br>Marco Gaboardi<br>$\lambda$-*Calculus and Soft Linear Logic* |
| | *Coffee Break* |
| 15:45 | Oliver Kullmann<br>*Sign-sensitive Graph Representations of CNFs:*<br>*Conflict graphs and resolution graphs*<br><br>Michael Brinkmeier<br>*Terms and Operads* |

## Tuesday, July 25th

| | |
|---|---|
| 9:30 | Ulrich Berger<br>*Strong normalisation via domain-theoretic computability predicates* |
| | *Coffee Break* |
| 10:45 | Lew Gordeev<br>*Strong WQO phase transitions*<br>Gerard Renardel de Lavalette<br>*Interpolation in Horn logic* |
| | *Lunch Break* |
| 13:45 | Henning Wunderlich<br>*Implicit characterizations of* FPTIME *and* NC *revisited*<br>Folke Eisterlehner<br>*An alternative correctness proof of a certification method for* FPTIME |
| | *Coffee Break* |
| 15:45 | Mathias Barra<br>*On some small subrecursive hierarchies*<br>Lars Kristiansen<br>$\boldsymbol{T}^{-}$*-Hierarchies and the Trade-off Theorem* |
| 17:15 | *Closing* |