Volume 3, Number 3
July 2016

Published by the Association for Computing Machinery
Special Interest Group on Logic and Computation

SIGLOG news

TABLE OF CONTENTS

Association for Computing Machinery

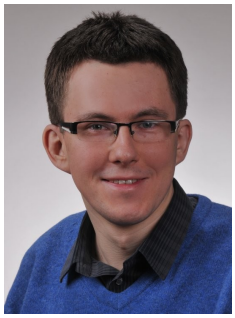
Advancing Computing as a Science & Profession

SIGLOG NEWS

Published by the ACM Special Interest Group on Logic and Computation

SIGLOG News (ISSN 2372-3491) is an electronic quarterly publication by the Association for Computing Machinery.

From the Editor



In this issue

— We have four technical columns!

  – Emmanuel Filiot and Pierre-Alain Reynier survey word functions and transducers in Mikołaj Bojańczyk's Automata column.

  – In the Complexity column edited by Neil Immerman, Toniann Pitassi and Iddo Tzameret discuss the latest results connecting proof complexity with algebraic complexity.

  – This year's Gödel Prize winners, Stephen Brookes and Peter O'Hearn, describe the origins of concurrent separation logic in our column on Semantics, edited by Mike Mislove and Prakash Panangaden.

  – Finally, Neha Rungta's column on Verification features an article on safety challenges posed by unmanned aircraft systems by César A. Muñoz, Aaron Dutle, Anthony Narkawicz, and Jason Upchurch.

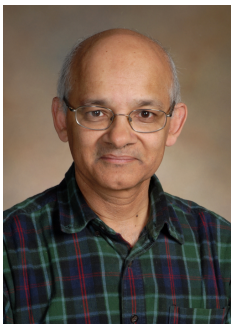
– In the Conference Reports section, edited by Jorge A. Pérez, Hugo Torres Vieira reports on DisCoTec 2016.

– As usual, we wrap up with the latest issue of SIGLOG Monthly, prepared by Daniela Petrişan.

SIGLOG News is still looking for a volunteer to coordinate a section on book reviews. Please email `editor@siglog.org` if you are interested.

Andrzej Murawski
University of Warwick
SIGLOG News Editor

Chair's Letter



Greetings and best wishes for a great summer. The Logic and Computation community has been particularly well recognized by recent awards. First, the inaugural Church Award was announced. The winners are Rajeev Alur and David Dill for the invention of timed automata. This work has been enormously influential in theory and practice. It is a staple of courses on verification and is part of many model-checking and verification systems. The award will be presented at the ACM-IEEE Symposium on Logic in Computer Science in New York in July. I remember meeting David Dill during ICALP in 1990 just before he was about to present the historic paper called "Automata for modeling real-time systems" which kicked off the subject. Of course, I had no idea I was about to see the start of something so enormously successful.

The Gödel Prize was awarded to two old friends of mine: Peter O'Hearn and Stephen Brookes for their work on concurrent separation logic. This issue's column on Semantics is an article by them on the genesis of this work. It is especially gratifying, as the Gödel Prize does not often go to logic related work. I remember O'Hearn as a clever but modest and unassuming graduate student. Steve Brookes and I were both on his PhD defence committee when he defended his thesis at Queen's University. It is wonderful to see the both of them recongnized in this way.

The SIGLOG elections are over; I thank everyone who voted and I especially thank Dale Miller for putting together a great slate of candidates. Every post was contested: this is a great sign of the vibrancy of the community. Three of the winners are incumbents: Alexandra Silva as Secretary, Luke Ong as vice-chair and myself as chair. Amy Felty is the newly elected Treasurer. We are, of course, gratified by the support we received and look forward to serving SIGLOG. I thank Natarajan Shankar for his dedicated service to SIGLOG as the Treasurer so far. It is also appropriate to remind the community that it was Shankar's inspired presentation at the SGB meeting that persuaded ACM to proceed with the chartering of SIGLOG.

Prakash Panangaden
McGill University
ACM SIGLOG Chair

AUTOMATA COLUMN

MIKOŁAJ BOJAŃCZYK, University of Warsaw
bojan@mimuw.edu.pl

A (word) transducer is an automaton with output, which defines a binary relation on words, including the important special case of a function from words to words. For the uninitiated, the transducer landscape can be a bit confusing, since there are several non-equivalent models (unlike for languages, where all reasonable models of finite automata are equivalent). In this column, Emmanuel Filiot and Pierre-Alain Reynier explain this landscape, with an emphasis on the connections with logic and algebra.

Transducers, Logic and Algebra for Functions of Finite Words

Emmanuel Filiot
Université Libre de Bruxelles

Pierre-Alain Reynier
Aix-Marseille Université

The robust theory of regular languages is based on three important pillars: computation (automata), logic, and algebra. In this paper, we survey old and recent results on extensions of these pillars to functions from words to words. We consider two important classes of word functions, the rational and regular functions, respectively defined by one-way and two-way automata with output words, called transducers.

1. INTRODUCTION

Important connections between computation, mathematical logic, and algebra have been established for regular languages of finite words. The class of regular languages corresponds to the class of languages recognized by finite automata, to the class of languages definable in monadic second-order logic (MSO) with one successor [Büchi 1960; Elgot 1961; Trakhtenbrot 1961], and to the class of languages whose syntactic monoid, a canonical monoid attached to every language, is finite (see for instance [Straubing 1994]). While automata are well-suited to study the algorithmic properties of regular languages, the algebraic view has provided effective characterizations of regular languages and its subclasses. Most notably, the problem of deciding whether a regular language is first-order definable amounts to checking whether its syntactic monoid, which is computable from any finite automaton recognizing the language, is aperiodic, which is decidable. See [Diekert et al. 2008] for a survey on first-order definable languages, and [Straubing 1994] for generalizations to other monoid varieties and fragments of MSO. In this paper, we want to survey some old and recent results that extend the three main pillars of language theory to functions of finite words.

At the computational level, word functions are defined by *transducers*, which extend automata with outputs on their transitions. They can be seen as Turing machines with a read-only input tape initially filled with the input word, and a write-only output tape on which to write the output word. We will consider two important classes of transducers: one-way and two-way transducers. For both classes, the output head is assumed to move only to the right, and for the class of one-way transducer, the input head is also restricted to move to the right. This yields the class of rational functions, long studied in the literature [Berstel and Boasson 1979], and the strictly more expressive class of regular functions, which has received more attention in the recent years.

At the logical level, we consider MSO transducers, as defined by Courcelle in the more general context of graph transformations [Courcelle 1994]. MSO transducers are based on MSO for words, and the main idea is to define the output word by some MSO

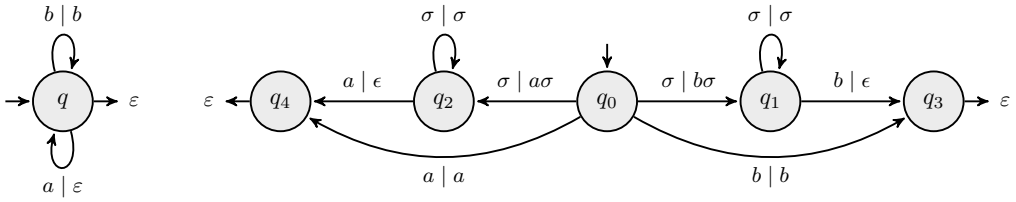


Fig. 1. One-way transducers T_0 and T_1 realizing f_{del} and f_{sw} respectively, where $\sigma \in \{a, b\}$.

interpretation on the input word. We present results establishing connections between one-way and two-way transducers and classes of MSO transducers.

At the algebraic level, we present a notion of syntactic congruence for functions introduced by Choffrut, which is of finite index iff the function is *sequential*, i.e. can be defined by some input-deterministic one-way transducer. We also present a generalization of this algebraic characterization to rational functions, a result due to Reutenauer and Schützenberger. We give an application of the latter characterization to the first-order definability problem for rational functions. For the more general class of regular functions, no algebraic characterization is known, but we present preliminary results towards it.

The paper is simply organized along the three pillars: transducers, logic, and algebra. We conclude with some extensions of the presented results and further research directions.

2. TRANSDUCERS

In this paper we assume Σ to be a finite alphabet and denote by Σ^* the set of finite words over Σ, and by ϵ the empty word. A *transduction* f is a relation on Σ^*. In this paper, we will be particularly interested in (partial) functions and for such objects, we denote by $\mathrm{dom}(f)$ their domain.

Example 2.1. In this paper, we will use three running examples, the functions f_{del}, f_{sw} and f_{mir}, over the alphabet $\Sigma = \{a, b\}$. The function f_{del} erases the letters a of an input word: e.g. $f_{del}(abba) = bb$. The function f_{sw} puts the last symbol in a first position ('sw' stands for 'swap'). It is defined by $f_{sw}(\epsilon) = \epsilon$ and for all $u \in \Sigma^*$ and $\sigma \in \Sigma$, by $f_{sw}(u\sigma) = \sigma u$. Finally, the function f_{mir} copies an input word u and concatenates it with its mirror image $\overline{u}$, i.e. $f_{mir}(u) = u\overline{u}$ for all $u \in \Sigma^*$. E.g. $f_{mir}(ab) = abba$.

2.1. One-way transducers

One-way transducers are Turing machines with two left-to-right tapes. The first tape is read-only and contains the input word while the second tape is write-only and contains the output word. For example, to realize the function f_{del} for Example 2.1 by a one-way transducer, the machine will simply go through the input word, from left to right, and, for each input symbol σ, copy σ on the output tape iff σ is different from a.

Formally, a one-way transducer[1] is a tuple $T = (Q, I, F, \Delta, t)$ where Q is a finite set of states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, $\Delta \subseteq Q \times \Sigma \times \Sigma^* \times Q$ is a finite set of transitions, and $t : F \to \Sigma^*$ is a terminal output function. A transition $\gamma = (p, a, w, q) \in \Delta$ of T is represented as $p \xrightarrow{a|w} q$. Intuitively it means that if the

[1]More generally, a transducer is defined as a finite automaton over the monoid $\Sigma^* \times \Sigma^*$. Our definition corresponds to *real-time* transducers in the literature [Berstel and Boasson 1979]. For transductions which are functions, real-time transducers and transducers coincide in expressiveness, therefore we just call them transducers in this paper, as this paper is about functions.

transducer is in state p, and the next input letter is a, then it may go to state q and write the word w on the output tape. W.l.o.g., we assume that for all $p, q \in Q$, $a \in \Sigma$, there exists at most one $w \in \Sigma^*$ such that $(p, a, w, q) \in \Delta$. The class of these transducers is denoted NFT, standing for non-deterministic finite-state transducers.

Given a word $u = a_1 \dots a_n \in \Sigma^*$, a run of T from p to q on u is a sequence of states $\rho = (q_i)_{i=0..n}$ such that $q_0 = p$, $q_n = q$ and, for each $i \in \{1, \dots, n\}$, there is a transition of the form $q_{i-1} \xrightarrow{a_i|w_i} q_i$ in Δ. We say that a run $\rho = (q_i)_{i=0..n}$ is accepting if $q_0 \in I$ and $q_n \in F$. The output of such an accepting run ρ on u, denoted by $\mathrm{out}^u(\rho)$, is defined as the concatenation of the words produced by the transitions and the image of state q_n by the terminal function t, *i.e.* the finite word $w_1 \dots w_n t(q_n) \in \Sigma^*$.

The transduction defined (or realized) by T is the relation $[\![T]\!]$ composed of the pairs (u, w) such that there exists an accepting run ρ of T on u satisfying $w = \mathrm{out}^u(\rho)$.

An NFT T is *functional* if the transduction it defines is a function. The class of functional NFT is denoted by fNFT. The class of functions realized by fNFT is called the class of *rational functions*[2]. An NFT is *sequential* if the underlying input automaton, obtained by projecting away the output words on transitions, is deterministic. Observe that such transducers obviously recognize functions. The class of sequential NFT is denoted by SFT. The class of functions realized by SFT is called the class of *sequential functions*.

Example 2.2. We describe two examples on the alphabet $\Sigma = \{a, b\}$. The transducer T_0, depicted on the left of Figure 1, is functional, sequential, and realizes the function f_{del}. Its terminal function is constant equal to ϵ, and is depicted by a target free outgoing arrow. The transducer T_1, depicted on the right of Figure 1, is functional but not sequential. It realizes the function f_{sw}. Intuitively, the non-determinism is mandatory to guess initially the last letter of the input word.

2.2. Two-way transducers

Two-way transducers extend one-way transducers as follows: instead of being left-to-right, the input tape is two-way. As an example, this intuitively allows the transducer to move to the end of the input word, and to copy it on the output tape during a right-to-left traversal. This transformation thus maps every input word to its mirror image.

Formally, we will consider that two-way transducers take as input a word $u \in \Sigma^*$ surrounded by left and right end-markers, denoted by $\vdash$ and $\dashv$, which we suppose do not belong to the alphabet Σ. These end-markers allow the transducer to identify the beginning and end of the input word. We denote by $\Sigma_\vdash$ the alphabet $\Sigma \cup \{\vdash, \dashv\}$. A *two-way transducer* is a tuple $T = (Q, I, F, \Delta)$ where Q, I and F are as in one-way transducers. Δ is a finite set of transitions which are elements of the form $(p, a, w, q, m) \in Q \times \Sigma_\vdash \times \Sigma^* \times Q \times \{-1, +1\}$. Compared with one-way transducers, transitions contain now a component $m \in \{-1, +1\}$ which indicates the direction of the move (left or right). We also require that transitions reading the left end-marker always move to the right.

Note also that the definition does not include a terminal function. This function is indeed useless for two-way transducers as it can be simulated using the right end-marker.

The class of these transducers is denoted by 2NFT, standing for two-way non-deterministic finite-state transducers.

A *configuration* of a two-way transducer T is a pair $(q, i) \in Q \times (\mathbb{N} \setminus \{0\})$ where q is a state and i is a position on the input tape. Given an input word $u = a_1 \dots a_n \in \Sigma^*$, we

[2]They are called rational because they can be alternatively defined by the more general and classical notion of rational subsets of monoids, see [Berstel and Boasson 1979].

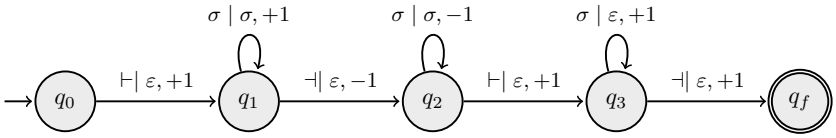


Fig. 2. A two-way transducer T_2 for f_{mir}.

will consider an execution of T on the input $\vdash u \dashv \in \Sigma_{\vdash}^*$. A *run* of T on the input $u' = \vdash u \dashv$ is a finite sequence of configurations $\rho = (p_1, i_1) \dots (p_m, i_m)$ such that $i_1 = 0$, $i_m = n+2$, and for all $k \in \{1, \dots, m-1\}$, $0 \leq i_k \leq n+1$ and [3] $(p_k, u'[i_k], w_k, p_{k+1}, i_{k+1} - i_k) \in \Delta$, for some word w_k. The output of such a run ρ on $\vdash u \dashv$, denoted by $\mathrm{out}^u(\rho)$, is defined as the concatenation of the words produced by the transitions, *i.e.* the finite word $w_1 \dots w_m \in \Sigma^*$.

This run is accepting if $p_1 \in I$ and $p_m \in F$. This means that for all the configurations but the last one, the reading head should be on a letter of the input u'. For the run to be accepting, the last configuration must correspond to the reading head being immediately after the end of the input word ($i_m = n + 2$).

As for NFT, the transduction defined by T is the relation $[\![T]\!]$ composed of the pairs (u, w) such that there exists an accepting run ρ of T on $\vdash u \dashv$ satisfying $w = \mathrm{out}^u(\rho)$.

As for one-way transducers, we will be interested in the subclasses of transducers *functional* and *sequential* transducers, denoted respectively by f2NFT and 2SFT, and defined respectively as the transducers whose semantics is a function, resp. the transducers whose underlying input (two-way) automaton is deterministic. The class of functions realized by f2NFT is called the class of *regular functions*. This terminology comes from the equivalence with a logical formalism, so-called MSO transducers, that will be presented in the next section.

Example 2.3. Consider the transducer T_2 for f_{mir} depicted on Figure 2. It maps every input word u to $u\bar{u}$, where $\bar{u}$ denotes the mirror image of u. The left and right end-markers are important here to allow the transducer to identify the beginning and end of the input word.

2.3. Landscape of transducer classes

Figure 3 gives the inclusion relations existing between the six classes of transducers we have defined so far. Let us first comment on these inclusions. Relations trivially extend functions. Rational functions strictly contain sequential functions, strictness being witnessed by the transduction f_{sw}. Regular functions strictly contain rational ones, as witnessed by f_{mir}. Last, unlike for one-way transducers, in presence of two-wayness, functional non-determinism does not increase expressive power (equality between f2NFT and 2SFT), as shown in [Engelfriet and Hoogeboom 2001]. This equivalence is effective.

We also depict on this picture the decidability status of several subclasses decision problems. For instance, we have proven in [Filiot et al. 2013] that it is decidable, given a f2NFT, whether there exists an equivalent one-way transducer. This decidability result is indicated on the edge from f2NFT to fNFT. Concerning this result, the overall complexity of our decision procedure is non-elementary, and a recent work provided a 2EXPSPACE procedure for the subclass of sweeping transducers [Baschenis et al.

[3] We use the following notation: $u'[0]$ denotes symbol $\vdash$, $u'[i]$ with $1 \leq i \leq n$ denotes the i-th letter of u, and $u'[n+1]$ denotes symbol $\dashv$.

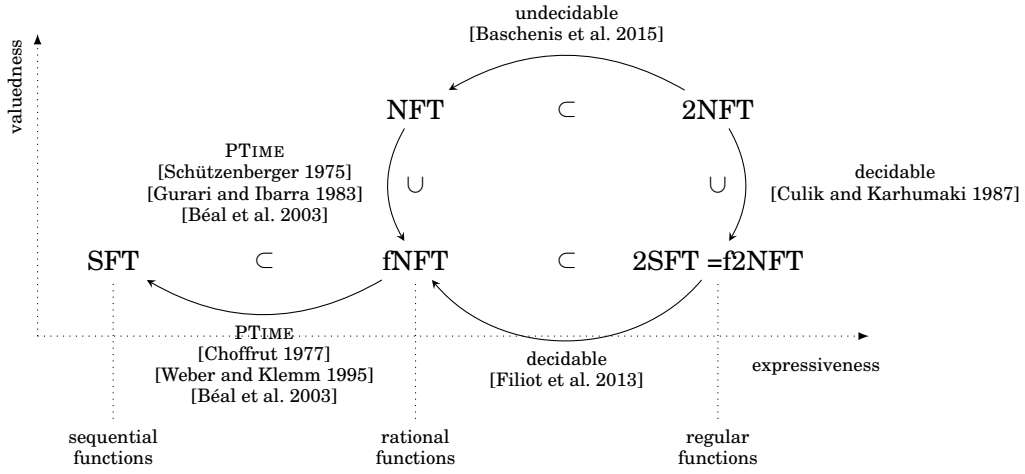


Fig. 3. A landscape of transducers of finite words.

2015]. Other decidability and undecidability results are depicted on the other edges in a similar way.

2.4. Equivalence problem

Checking whether two transducers are equivalent, *i.e.* whether they define the same transduction, is a natural and important problem, that has been studied by several authors. Unfortunately, this problem is already undecidable for NFT, as proven in [Griffiths 1968]. On the positive side, this problem is decidable for the classes of functional transducers we have presented (it is actually sufficient decide whether two functional transducers have the same domain, and then decide whether the disjoint union of the two transducers is functional), which explains the interest for these classes. The decidability frontier lies actually beyond functional transducers, as it encompasses the class of so-called *finite-valued transducers*, defined as the transducers for which there exists some natural number k such that, for every input word, the number of outputs associated with this input word is at most k. This decidability result has been proven for instance in [Culik II and Karhumäki 1986; Weber 1993; de Souza 2008] for finite-valued one-way transducers. In [Culik II and Karhumäki 1986], the authors claim that the equivalence problem is even decidable for the class of finite-valued two-way transducers.

3. LOGICS

A formalism based on monadic second-order logic (MSO) has been defined by [Courcelle 1994] to define transformations of graph structures. In this section, we cast this formalism to word to word functions, and refer the reader to [Courcelle and Engelfriet 2012] for more details and results about MSO transducers for general graph structures.

Words w are seen as logical structures on the domain $\{1, \dots, |w|\}$ over the signature consisting of unary predicates $a(x)$ for each symbol of Σ, and the binary predicate $x \preceq y$ for the total order on positions. Recall that MSO on words is the extension of first-order logic with quantification over sets of positions (see [Straubing 1994] for details). It is well-known by Büchi's theorem that a language is MSO-definable iff it is regular. We

present extensions of this fundamental result to word functions defined by one-way and two-way transducers.

3.1. MSO transducers

The definition of MSO transducers is technical and in this paper, we rather want to define them intuitively and with examples. We refer the interested reader to [Courcelle 1994; Engelfriet and Hoogeboom 2001; Filiot 2015] for detailed definitions.

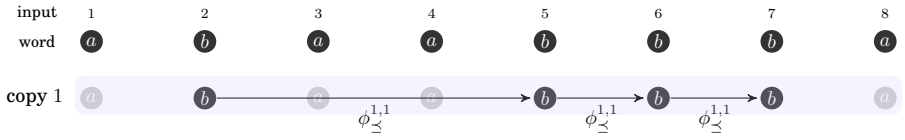
In an MSO transducer, the output word is defined as an MSO interpretation over a fixed number k of copies of the input word. Therefore, the nodes of the output word are copies 1 to k of the nodes of the input word. Output nodes are denoted x^c, for every copy c and input node x. For every copy c, only the nodes satisfying a given formula $\phi^c_{pos}(x)$ with one free first-order variable x are kept. For instance, assume one takes two copies of the input word, and in the first copy, one keeps all nodes x^1 such that x is labeled a, and in the second copy, one keeps all nodes x^2 such that x is labeled b. This is specified by the two formulas $\phi^1_{pos}(x) = a(x)$ and $\phi^2_{pos}(x) = b(x)$.

The output label and order predicates are defined by MSO formulas with one and two free first-order variables respectively, interpreted over the input structure. For instance, over the alphabet $\Sigma = \{a, b\}$, to set all the output labels to a, one just specifies the formulas $\phi^c_a(x) = \top$ and $\phi^c_b(x) = \bot$ for all copies c. The output order predicate relates input nodes of possibly different copies, and is therefore defined by formulas of the form $\phi^{c,d}_{\preceq}(x, y)$ for any copies $1 \leq c, d \leq k$.

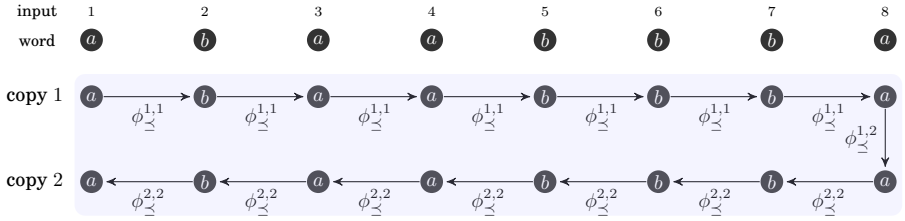
Finally, a closed MSO formula ϕ_{dom} defines the domain of the function. All in all, an MSO transducer over an alphabet Σ is a tuple

$$T = (k, \phi_{dom}, (\phi^c_{pos}(x))_{1\leq c\leq k}, (\phi^c_a(x))_{1\leq c\leq k, a\in\Sigma}, (\phi^{c,d}_{\preceq}(x, y))_{1\leq c,d\leq k})$$

The output structure may not be a word, but here we assume that an MSO transducer T outputs only word structures. It is a decidable property. Note that the length of output word of an input word of length M by an MSO transducer is bounded by kM, since one takes a fixed number k of copies of the input word.



(a) Transformation f_{del} defined by T_{del}



(b) Transformation f_{mir} defined by T_{mir}

Fig. 4. Functions defined by MSO-transducers

3.2. Examples of MSO transducers

As a first example, an MSO transducer T_{del} that realizes f_{del} is illustrated in Fig. 4(a) (only the successor relation is depicted). Input nodes filtered out by formulas $\phi^c_{pos}(x)$ are represented by fuzzy nodes. It is realized by the MSO transducer

$$T_{del}{=}(1, \phi_{dom} \equiv \top, \phi^1_{pos}(x) \equiv \neg a(x), (\phi^1_\sigma(x) \equiv \sigma(x))_{\sigma\in\Sigma}, \phi^{1,1}_{\preceq}(x,y) \equiv x \preceq y)$$

As a second example, consider the function f_{mir}. To realize f_{mir} with an MSO transducer (Fig. 4(b)), one needs two copies of an input word u. The labels are kept unchanged, however the order is reversed for the second copy. One also sets that all nodes of the first copy are smaller than the nodes of the second copy, as they appear before in the output word. Therefore, f_{mir} is realized by the transducer T_{mir} with $k = 2$ and

$$\begin{array}{cccccccc} \phi_{dom} & \equiv & \top & \phi^c_{pos}(x) & \equiv & \top & \phi^c_a(x) & = & a(x) & \phi^c_b(x) & = & b(x) \\ \phi^{1,1}_{\preceq}(x,y) & \equiv & x \preceq y & \phi^{1,2}_{\preceq}(x,y) & \equiv & \top & \phi^{2,1}_{\preceq}(x,y) & \equiv & \bot & \phi^{2,2}_{\preceq}(x,y) & \equiv & y \preceq x \end{array}$$

for all copies $c \in \{1, 2\}$. Note that given an input word u, the order on the output word positions is the binary relation $O = \{(i^c, j^d) \mid 1 \leq c, d \leq 2, i, j \in dom(u), u \models \phi^{c,d}_{\preceq}(i, j)\}$. Only the successor relation induced by O is depicted on the figure.

3.3. Büchi theorems for word functions

Regular functions. The first automata-logic connection for word transformations has been given by Engelfriet and Hoogeboom for the class of regular functions:

THEOREM 3.1. *[Engelfriet and Hoogeboom 2001] A function f is regular iff it is realized by some MSO transducer.*

In order to prove the direction from 2SFT to MSO transducers, one builds an MSO transducer that, given an input word u, outputs a (linear) graph that represents the run of the 2SFT on u. The converse direction is more complex, and involves an extended model of two-way transducers which can perform "MSO jumps" $\varphi(x, y)$, where $\varphi(x, y)$ is an MSO formula that defines a function from x positions to y positions. Intuitively, the machine can move from position x to position y providing $\varphi(x, y)$ holds true. 2SFT with MSO jumps are then converted, based on the Büchi theorem, into 2SFT with regular look-around. These 2SFT can move only to positions which are in the 1- neighbourhood of the current position, but their move can be based on a regular property of the current prefix and suffix of the word. Finally, it is shown that 2SFT with regular look-around are equivalent to 2SFT.

Rational functions. Using an adequate restriction of MSO transducers, one can also prove a Büchi theorem for rational functions. Intuitively, an MSO transducer is *order-preserving* if in the graphical representation of the output, there is no right-to-left edge. For instance, considering the transformations depicted on Figure 4, the transformation f_{del} satisfies this property while the transformation f_{mir} does not. Formally, this property requires that for every input word u and for every $c, d \in \{1, \ldots, k\}$, if the formula $\phi^{c,d}_{\preceq}(x, y)$ evaluates to true in u, then $x \preceq y$ also evaluates to true. This property is decidable.

THEOREM 3.2. *[Bojanczyk 2014; Filiot 2015] A function f is rational iff it is realized by some order-preserving MSO transducer.*

The proof proceeds as follows: given a one-way transducer, the MSO transducer that builds as output the run of the one-way transducer naturally satisfies the order-preserving property. Conversely, the idea is to identify, for each position of the input

word, the subword of the output that corresponds to this position. One can then represent the transduction as a regular language, and make use of Büchi theorem.

The power of transducer-logic connections is illustrated by the following definability problem. As shown in [Filiot et al. 2013], it is decidable whether a deterministic two-way finite state transducer T is equivalent to a one-way functional finite state transducer. As a consequence of this result and of the two previous theorems, we obtain the following corollary:

COROLLARY 3.3. *Given an MSO-transducer, it is decidable whether it is equivalent to some order-preserving MSO-transducer.*

4. ALGEBRA

Similarly to regular languages, sequential and rational functions can be characterized by the index finiteness of well-chosen (canonical) congruences. For the class of regular functions, no algebraic characterizations are known but we present partial results based on algebraic properties of the transition structure of two-way transducers (their transition monoid). For the class of regular functions with *origin information*, i.e. regular functions extended with pointers from any output position to some input position (intuitively, the input position from which they originate), an algebraic characterization is known [Bojanczyk 2014]. We present origin information at the end of this section.

We recall that a left (resp. right) congruence on Σ^* is an equivalence relation $\equiv$ such that for all $u, v \in \Sigma^*$ and $\sigma \in \Sigma$, if $u \equiv v$ then $\sigma u \equiv \sigma v$ (resp. if $u \equiv v$ then $u\sigma \equiv v\sigma$). For $u \in \Sigma^*$, we denote by $[u]_\equiv$ the class of u, or just $[u]$ when it is clear from the context. A congruence $\equiv$ (left or right) has *finite-index* if its quotient $\Sigma^*/_\equiv$ has finitely many classes.

In this section, we also denote by $\preceq$ the prefix relation on words, and by $u \wedge v$ the longest common prefix of any two words u and v. Given a language $L \subseteq \Sigma^*$ and a word $u \in \Sigma^*$, we denote by $u^{-1}L$ the residual of L by u, i.e. the set of words w such that $uw \in L$. When $u \preceq v$, we denote by $u^{-1}v$ the unique word w such that $v = uw$.

4.1. Sequential functions

It is well-known that any regular language is recognized by a unique minimal complete deterministic automaton. Intuitively, for a language L, any two words u, v which behave equivalently with respect to continuations w ($uw \in L$ iff $vw \in L$) must reach the same state in a minimal deterministic automaton. This is captured by the right congruence $\equiv_L$: $u \equiv_L v$ if for all $w \in \Sigma^*$, $uw \in L$ iff $vw \in L$. The well-known Myhill-Nerode's theorem states that L is regular iff $\equiv_L$ has finite index. Moreover, the index of $\equiv_L$ is exactly the number of states of the minimal complete deterministic automaton recognizing L.

In this section, we explain how to extend the characterization of regular languages to sequential functions, an extension due to Choffrut [Choffrut 1979; Choffrut 2003]. In a minimal sequential transducer defining a function $f : \Sigma^* \to \Sigma^*$, two (input) words u, v must go to the same state if they behave the same w.r.t. $\mathrm{dom}(f)$ ($u \equiv_{\mathrm{dom}(f)} v$), but also w.r.t. the outputs, in the sense that the output produced when processing a continuation w should not depend on u and v. To capture this idea, one first defines a canonical way of producing the output, via a total function $\hat{f} : \Sigma^* \to \Sigma^*$ which, given a word u, outputs the longest common prefix of all words $f(uw)$ for all continuations $w \in u^{-1}\mathrm{dom}(f)$:

$$\hat{f}(u) = \bigwedge \{f(uw) \mid w \in u^{-1}\mathrm{dom}(f)\} \quad \text{where } \bigwedge \varnothing \text{ is set to } \epsilon.$$

Then, one defines the relation $\equiv_f$ by $u \equiv_f v$ if the following two statements hold:

$$(i)\ u \equiv_{\mathrm{dom}(f)} v \quad (ii)\ \forall w \in u^{-1}\mathrm{dom}(f),\ \hat{f}(u)^{-1} f(uw) = \hat{f}(v)^{-1} f(vw)$$

Note that $\hat{f}(u)^{-1}$ is necessarily a prefix of $f(uw)$, and similarly for $\hat{f}(v)^{-1}$, by definition of $\hat{f}$. It turns out that $\equiv_f$ is a right congruence, which allows one to construct a transducer T_f for f (possibly with infinitely many states). The set of states of T_f is $\Sigma^*/_{\equiv_f}$, with initial state $[\epsilon]$ and set of accepting states $\mathrm{dom}(f)/_{\equiv_f}$. The transitions are $([u], a, w, [ua])$ with $u \in \Sigma^*$, $a \in \Sigma$ and $w = \hat{f}(u)^{-1}\hat{f}(u\sigma)$. The terminal function is $t : [u] \mapsto \hat{f}(u)^{-1} f(u)$ for $u \in \mathrm{dom}(f)$.

Note that if $\equiv_f$ has finite index, then T_f is a proper sequential transducer with finitely many states, and therefore f is sequential. The converse is also true, and one gets a Myhill-Nerode theorem for sequential functions, due to Choffrut:

THEOREM 4.1. *[Choffrut 2003; Choffrut 1979] A function $f : \Sigma^* \to \Sigma^*$ is sequential iff $\equiv_f$ has finite index.*

Note that T_f is canonical. Moreover when f is sequential, then T_f is minimal, and any sequential transducer realizing f can be uniquely mapped to T_f, through a notion of transducer morphism, e.g. defined in [Choffrut 2003]. In that sense, T_f is unique.

4.2. Rational functions

In this section, we present an algebraic characterization, as well as a canonical transducer construction, for rational functions, due to Reutenauer and Schützenberger. This characterization is based on the model of *bimachines*, which are essentially sequential transducers extended with regular look-ahead. We adopt the latter view, which will hopefully allow us to give an intuitive explanation of Reutenauer and Schützenberger's result. This formalization also appeared in [Boiret et al. 2012].

Rational functions with infinite syntactic congruence. Let us see why the finiteness of $\equiv_f$ fails at characterizing rational (but non-sequential) functions. Over $\Sigma = \{a, b\}$, consider the function f_{sw} of Example 2.1. The congruence $\equiv_{\mathrm{dom}(f_{sw})}$ has only one class Σ^*. The function $\hat{f}_{sw}$ is constant equal to ϵ: for any $u \in \Sigma^*$, $\hat{f}_{sw}(u) \preceq (f_{sw}(ua) \wedge f_{sw}(ub)) = au \wedge bu = \epsilon$. Therefore, for all $u, v \in \Sigma^*$ and all $w \in \Sigma^*$, $\hat{f}_{sw}(u)^{-1} f_{sw}(uw) = \hat{f}_{sw}(u)^{-1}(v) f_{sw}(vw)$ iff $f_{sw}(uw) = f_{sw}(vw)$ iff $u = v$. Therefore, $\equiv_{f_{sw}}$ has infinite index.

Now, consider the following two restrictions on $\mathrm{dom}(f_{sw})$: $\mathrm{last}_a = \{ua \mid u \in \Sigma^*\}$ and $\mathrm{last}_b = \{ub \mid u \in \Sigma^*\}$, then the function f_{sw} is sequential on these two restricted domains. This suggests that modulo some information about the suffix (whether it ends with a or b in this case), the function is sequential. Unlike the latter example, the suffix information needed to be sequential may change along an input word. Consider for instance the iterated version of f_{sw} that we denote f^*_{sw}, which is defined over the alphabet $\Delta = \Sigma \cup \{\#\}$ by:

$$f^*_{sw} \ :\ u_1 \# u_2 \# \ldots \# u_n \mapsto f_{sw}(u_1) \# f_{sw}(u_2) \# \ldots \# f_{sw}(u_n) \qquad \text{where } u_i \in \Sigma^* \text{ and } n \geq 0$$

To realize f^*_{sw} in a sequential way, a transducer needs to know, when reading a symbol $\sigma \in \{a, b\}$, if it is the last letter of a block in $\{a, b\}^+$, and otherwise whether the last letter of this block is an a or a b. Modulo this look-ahead information, the function f^*_{sw} is sequential.

Sequentiality modulo regular look-ahead. Let us formalise the notion of sequentiality modulo (regular) look-ahead information. A *look-ahead information* $\mathcal{L}$ is a partition of Σ^* assumed to be the quotient of Σ^* by a left congruence of finite index, denoted by $\equiv_{\mathcal{L}}$. We denote by $[u]_{\mathcal{L}}$ the class of a word u. The *look-ahead extension* is

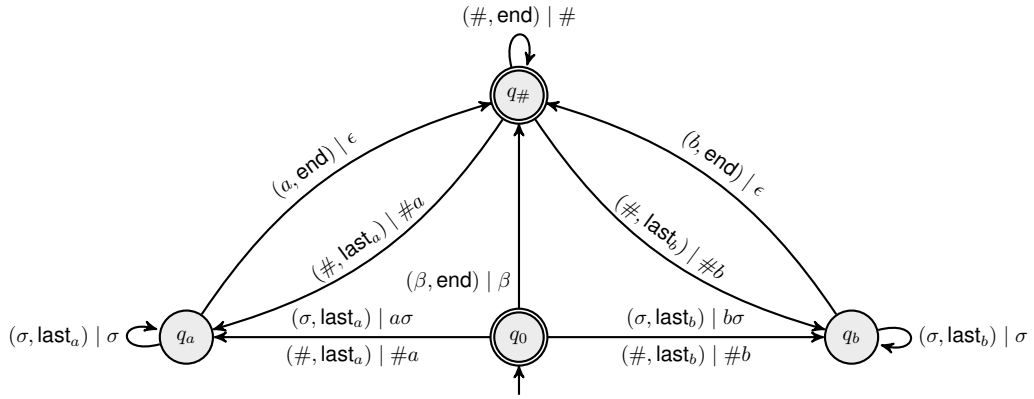


Fig. 5. Sequential transducer realizing $f^*_{sw}[\mathcal{L}]$, where $\sigma \in \{a, b\}$ and $\beta \in \{a, b, \#\}$.

the function $e_\mathcal{L} : \Sigma^* \to (\Sigma \times \mathcal{L})^*$ defined for all $u = \sigma_1 \dots \sigma_n \in \Sigma^*$ by $e_\mathcal{L}(u) = (\sigma_1, [\sigma_2 \dots \sigma_n]_\mathcal{L})(\sigma_2, [\sigma_3 \dots \sigma_n]_\mathcal{L}) \dots (\sigma_n, [\epsilon]_\mathcal{L})$. A function $f : \Sigma^* \to \Sigma^*$ is *sequential modulo* $\mathcal{L}$ (or $\mathcal{L}$-sequential) if the function denoted $f[\mathcal{L}] : (\Sigma \times \mathcal{L})^* \to \Sigma^*$ defined on $e_\mathcal{L}(\mathrm{dom}(f))$ by $f[\mathcal{L}](e_\mathcal{L}(u)) = f(u)$, is sequential.

Note that a function f over an alphabet Σ is sequential iff it is $\{\Sigma^*\}$-sequential. As a second example, the function f^*_{sw} is $\mathcal{L}$-sequential for $\mathcal{L} = \{\mathsf{end}, \mathsf{last}_a, \mathsf{last}_b\}$ where $\mathsf{end} = \epsilon + \#\Delta^*$, and for $\sigma \in \{a, b\}$, $\mathsf{last}_\sigma = (a + b)^*\sigma(\epsilon + \#\Delta^*)$. The function $f^*_{sw}[\mathcal{L}]$ is realized by the sequential transducer of Fig. 5. Note that this transducer also checks that the look-ahead annotations are correct, i.e. it ensures that the domain is exactly $e_\mathcal{L}(\mathrm{dom}(f^*_{sw}))$. For instance, whenever the information end is read, the transducer moves to state $q_\#$, from which only $\#$ can be read. This ensures that the annotation by look-ahead information end is correct. The only way to access state q_a is when reading the look-ahead information last_a, and the only way to leave q_a is when reading (a, end), thus ensuring that the last letter of the block is indeed a. Since the transducer is sequential, has domain $e_\mathcal{L}(\mathrm{dom}(f^*_{sw}))$, and the look-ahead annotation is unique for each word, projecting away the look-ahead information on this transducer yields an unambiguous transducer realizing f^*_{sw}.

A canonical look-ahead. It is known [Elgot and Mezei 1965] that any rational function $f : \Sigma^* \to \Sigma^*$ equals the composition of two functions $r : \Sigma^* \to \Delta^*$ and $\ell : \Delta^* \to \Sigma^*$ for some intermediate alphabet Δ, i.e. $f = \ell \circ r$, where ℓ is sequential and r is right-sequential, i.e. the function $m \circ r \circ m$ is sequential, where m is the function that mirrors input words. In other words, r can be realized by a sequential transducer that processes input words backwards and produces output words backwards. Moreover, T can be chosen to be letter-to-letter, i.e. produce exactly one output symbol per input symbol, and hence T is nothing else than an automaton that computes some look-ahead information (the output symbols). We can rephrase this decomposition result in terms of $\mathcal{L}$-sequentiality:

THEOREM 4.2. *[Elgot and Mezei 1965] f is rational iff it is $\mathcal{L}$-sequential for some $\mathcal{L}$ of finite index.*

The strong result of Reutenauer and Schützenberger is precisely to show that, in the latter proposition, $\mathcal{L}$ can be chosen in a canonical way, that depends only on f, denoted $\mathcal{L}_f$. The idea is to identify suffixes u and v that have only a bounded "difference" on the outputs $f(wu)$ and $f(wv)$ for all w such that $wu, wv \in \mathrm{dom}(f)$. To quantify this

effect, they use the notion of *delay distance* between words, defined by $d(t_1, t_2) = |t_1| + |t_2| - 2|t_1 \wedge t_2|$. In other words, this distance only counts what remains when the longest common prefix of t_1 and t_2 has been cut out. Then, two words u, v are equivalent for $\equiv_{\mathcal{L}_f}$ if (i) for all $w \in \Sigma^*$, $wu \in \mathrm{dom}(f)$ iff $wv \in \mathrm{dom}(f)$, and (ii) the set $\{d(f(wu), f(wv)) \mid wu, wv \in \mathrm{dom}(f)\}$ is finite. It turns out that $\equiv_{\mathcal{L}_f}$ is a left congruence, and it is of finite index when f is rational.

Example 4.3. As an example, consider again the function f^*_{sw} defined before. The look-ahead given before is actually the canonical one, i.e. $\mathcal{L}_{f^*_{sw}} = \{\mathsf{end}, \mathsf{last}_a, \mathsf{last}_b\}$. Indeed, let $\sigma, \beta \in \{a, b\}$, $u, v \in \{a, b\}^*$ and $u', v' \in \mathsf{end}$. Then, any w such that $wu\sigma u' \in \mathrm{dom}(f^*_{sw})$ and $wv\beta v' \in \mathrm{dom}(f^*_{sw})$ is of the form $w_1 w_2$ where $w_1 \in \Delta^*\#$ and $w_2 \in \Sigma^*$. Then, $\{d(f^*_{sw}(w_1 w_2 u\sigma u'), f^*_{sw}(w_1 w_2 v\beta v')) \mid w_1 \in \Delta^*\#, w_2 \in \Sigma^*\} = \{d(f^*_{sw}(w_1)\sigma w_2 u f^*_{sw}(u'), f^*_{sw}(w_1)\beta w_2 v f^*_{sw}(v')) \mid w_1 \in \Delta^*\#, w_2 \in \Sigma^*\}$, which is finite iff $\beta = \sigma$. This gives the two classes last_a and last_b. Similarly, it is possible to check that all words in the set end are equivalent.

We can now state Reutenauer and Schützenberger's result:

THEOREM 4.4. *[Reutenauer and Schützenberger 1991] Let $f : \Sigma^* \to \Sigma^*$. The following three statements are equivalent:*

(1) f is rational,
(2) $\mathcal{L}_f$ has finite index and f is $\mathcal{L}_f$-sequential,
(3) $\mathcal{L}_f$ and $\equiv_{f[\mathcal{L}_f]}$ have finite index.

When f is given by a transducer, a canonical sequential transducer for $f[\mathcal{L}_f]$ can be constructed effectively. Projecting its input symbols on Σ (i.e. discarding the look-ahead information), one gets a canonical unambiguous transducer T_f realizing f.

Bimachines. The latter theorem was originally presented based on the notion of bimachines. A bimachine is made of a right deterministic automaton R reading input words from right to left, a deterministic automaton L, and an output function ω which takes states of L, states of R and symbols in Σ as arguments, and produces a word. The output produced at position i in a word $w = \sigma_1 \dots \sigma_n$ is $\omega(l, \sigma_i, r)$ where l is the state of L reached after processing the prefix $\sigma_1 \dots \sigma_i$ (if it exists), and r is the state of R reached after reading the suffix $\sigma_i \dots \sigma_n$. Reutenauer and Schützenberger's result is precisely to show that for any rational function f, there exists a canonical right automaton R_f, a left automaton $L[R_f]$ (which is canonical once the right automaton is fixed), and an output function ω, which defined a canonical bimachine for f. Translated in the look-ahead framework, R_f defines the look-ahead information $\mathcal{L}_f$, and by taking the product of R_f and $L[R_f]$, one obtains a sequential transducer realizing $f[\mathcal{L}_f]$.

4.3. First-order definable functions

If only first-order formulas are allowed in the definition of (order-preserving) MSO-transducers, one gets the subclass of (order-preserving) FO-transducers. A function is (order-preserving) first-order definable if it is definable by an (order-preserving) FO-transducer. First-order definable languages L are characterized by languages having aperiodic syntactic congruence[4] $\equiv_L$, which yields a decision procedure for automata: minimize the automaton and check the aperiodicity of its transition congruence. We present similar results for rational functions and partial results for functions definable by two-way transducers.

[4]A congruence $\equiv$ on Σ^* is aperiodic if there exists $n_0 \in \mathbb{N}$ such that for all $u, v \in \Sigma^*$ and $n \geq n_0$, $u^n \equiv v^n$ iff $u^{n+1} \equiv v^{n+1}$.

The *transition congruence* $\equiv_A$ of an automaton A is defined by $u \equiv_A v$ if for all states p, q, it holds $p \xrightarrow{u} q$ iff $p \xrightarrow{v} q$. The transition congruence of a transducer is the transition congruence of its underlying (input) automaton, and a transducer is aperiodic if its transition congruence is aperiodic. It is known that a rational function f is order-preserving first-order definable iff it is realized by some aperiodic transducer [Filiot et al. 2016]. Moreover, a function f is realizable by some aperiodic transducer iff the canonical left congruence $\mathcal{L}_f$ and the right congruence $\equiv_{f[\mathcal{L}_f]}$ are both aperiodic, which is decidable when f is given by some transducer. Therefore, one gets the following theorem:

THEOREM 4.5. *[Lhote 2015; Filiot et al. 2016] It is decidable whether a transducer defines an order-preserving first-order function.*

The problem of deciding whether a transducer is equivalent to some aperiodic and functional one has been generalized to arbitrary congruence varieties in [Filiot et al. 2016]. It is shown in this case that the congruences $\mathcal{L}_f$ and $\equiv_{f[\mathcal{L}_f]}$ may not be in the varieties, even if some transducer realizing the function is, but at least one pair of congruences $\mathcal{L}$ and $\equiv_{f[\mathcal{L}]}$ is in the variety, where those pairs are taken in a finite computable set of congruences.

No such results are known for functions definable by deterministic two-way transducers, mainly because for such functions the existence of a canonical device is still open. Nevertheless, the notion of transition congruence can be extended to two-way automata and therefore to two-way transducers. Roughly, it identifies words with the same state behaviors, where the behaviors are either left-to-right, right-to-left, left-to-left and right-to-right. For instance, left-to-right behaviors are exactly as for classical one-way automata, and a pair of states (p, q) if a left-to-left behavior of a word u if there is a run that enters u from the left in state p, and leaves u to the left in state q.

THEOREM 4.6. *[Carton and Dartois 2015] A function f is first-order definable iff it is realized by some aperiodic two-way transducer.*

In [Bojanczyk 2014], a *transduction with origin* is a function from words u to pairs (v, o), where v is a word, and o is an origin mapping that sends any position of v to a position of u, the position from which "it has been created". Most transducer models, including MSO- and FO-transducers, implicitly bear origin information. We denote by $[\![T]\!]_o$ the origin transduction defined by a transducer. For regular transductions with origin, an algebraic characterization is known (see [Bojanczyk 2014]), based on which first-order definability can be decided:

THEOREM 4.7. *[Bojanczyk 2014] Given a two-way transducer T realizing a transduction with origin $[\![T]\!]_o$, it is decidable whether there exists an FO-transducer T' such that $[\![T]\!]_o = [\![T']\!]_o$.*

5. EXTENSIONS AND PERSPECTIVES

Functions of finite words enjoy multiple presentations by means of transducers, logic and algebra. We have presented some old and recent results using these different tools, as well as some nice equivalence results. In this conclusion, we present a recent alternative automaton model, as well as some extensions to other structures than finite words.

Streaming String Transducers. Recently, an alternative model of transducers has been introduced in [Alur and Černý 2011], named streaming string transducers (SST for short). Intuitively, it consists in a deterministic one-way automaton extended with a finite number of registers, valued with finite words. These registers are updated along

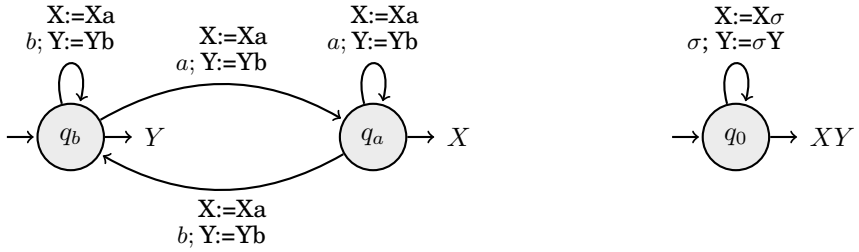


Fig. 6. Two streaming string transducers.

transitions but never tested, and they are used to define the output of a run. Two examples of SST are depicted on Figure 6. The left SST realizes the function mapping any word of the form $u\sigma$ to $\sigma^{|u|}$, and ϵ to itself, while the right SST realizes the function f_{mir}.

Interestingly, it has been proven that a simple restriction of this model, so-called copyless SST (updates should make a linear use of registers) exactly coincides with the class of regular functions [Alur and Černý 2010]. Intuitively, this means that the model allows to transfer the complexity of runs of two-way transducers to the updates of variables. Similarly, a simple restriction of SST (called right-appending SST) coincides with the class of rational functions, namely that in which updates are of the form $X := Yu$, where X and Y are registers, and u is a finite word. This model has been applied to the verification of list-processing programs [Alur and Černý 2011], and implemented [Alur et al. 2015].

Concerning first-order definable transformations, a result similar to Theorem 4.6 has been provided for streaming string transducers in [Filiot et al. 2014; Dartois et al. 2016b], using an adequate notion of transition monoid of an SST.

Another natural problem for this model is the notion of register complexity of a function, *i.e.* the minimal number of registers needed to realize a regular function. This problem has attracted a lot of attention recently. In [Daviaud et al. 2016], a solution is proposed for the class of right-appending SST using a generalization of the twinning property, a tool that has been introduced in [Choffrut 1977] to characterize sequential functions among rational ones. In [Baschenis et al. 2016], the authors consider also a register minimization problem but for the class of non-deterministic SST where concatenations of registers are forbidden in the register updates.

Infinite words. The determinization problem, i.e. deciding whether a transducer defines a sequential function, has been extended to infinite word transducers with Büchi acceptance condition in [Béal and Carton 2002]. Correspondence between MSO and SST on infinite strings have been shown in [Alur et al. 2012].

Trees and nested words. Tree transducers have been studied in numerous works, in particular by Joost Engelfriet and Sebastian Maneth, and we will not give here an exhaustive list. However, in order to echo the transducer-logic connection that we have presented in this paper, let us mention that it has been in shown in [Engelfriet and Maneth 2003] that MSO-definable tree transducers exactly coincide with Macro Tree Transducers that are linear-size increase, *i.e.* the size of the output tree is always linear in the size of the input tree. Let us also mention the survey [Maneth 2015] about the equivalence problem for tree transducers.

Recently, we have also considered transducers whose inputs are tree linearizations represented as nested words [Filiot et al. 2010]. We have defined so called visibly push-

down transducers and have proven that several positive results of one-way transducers are preserved by this model. Recently, we have shown that MSO-definable nested word-to-word transformations and a two-way model of visibly pushdown transducers are equivalent [Dartois et al. 2016a].

Some perspectives. For the class of regular functions, we have seen logical and automata models. An interesting and challenging research direction is to build an algebraic framework for regular transductions.

On the logical side, MSO transducers can be transformed into two-way or streaming transducers in non-elementary time. This blow-up is not avoidable, which raises the question of having MSO expressive logics well-suited to define transductions, and that enjoy better complexities.

Finally, an interesting direction, which generalizes the classical Church synthesis problem, is that of sequential uniformisation. The problem is to decide whether from a given word relation, one can extract a function such that (i) it has the same domain as the relation, (ii) it is included in the relation, and (iii) it belongs to some class of functions with interesting properties. The relation can be thought of as a set of good behaviours of a system, and the function as the behaviour of the synthesised system. The required properties of the targeted class of functions depend on the applications. For instance, one may target sequential functions for memory efficiency. For rational relations and the class of sequential functions, this problem is undecidable, but decidable when restricted to finite-valued rational relations or deterministic rational relations [Filiot et al. 2016].

Acknowledgments

We thank the editorial board of ACM SIGLOG newsletter for giving us the opportunity to present these exciting results. We would like to warmly thank Nathan Lhote for his careful reading on the algebra section.

This work was partially supported by the French ExStream project (ANR-13- JS02-0010), the Belgo-French PHC project VAST (35961QJ) funded by Campus France and WBI, the ARC project Transform (Federation Wallonia-Brussels) and the Belgian FNRS CDR project Flare. Emmanuel Filiot is research associate at F.R.S.-FNRS.

REFERENCES

Rajeev Alur and Pavol Černý. 2010. Expressiveness of streaming string transducers. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS) (LIPIcs)*, Vol. 8. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 1–12.

Rajeev Alur and Pavol Černý. 2011. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proc. of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*. ACM, 599–610.

Rajeev Alur, Loris D'Antoni, and Mukund Raghothaman. 2015. DReX: A Declarative Language for Efficiently Evaluating Regular String Transformations. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 125–137.

Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. 2012. Regular Transformations of Infinite Strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. 65–74.

Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. 2015. One-way definability of sweeping transducers. In *35th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015 (LIPIcs)*, Vol. 45. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 178–191.

Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. 2016. Minimizing resources of sweeping and streaming string transducers. In *In Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP'16) (LIPIcs)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. To appear.

Marie-Pierre Béal and Olivier Carton. 2002. Determinization of transducers over finite and infinite words. *Theor. Comput. Sci.* 289, 1 (2002), 225–251.

Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. 2003. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science* 292, 1 (2003), 45–63.

Jean Berstel and Luc Boasson. 1979. Transductions and context-free languages. *Ed. Teubner* (1979), 1–278.

Adrien Boiret, Aurélien Lemay, and Joachim Niehren. 2012. Learning Rational Functions. In *Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings (Lecture Notes in Computer Science)*. Springer, 273–283.

Mikolaj Bojanczyk. 2014. Transducers with Origin Information. In *41st Internationl Colloquium on Automata, Languages, and Programming (ICALP) (LNCS)*, Vol. 8573. Springer, 26–37.

J. R. Büchi. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6, 1–6 (1960), 66–92.

Olivier Carton and Luc Dartois. 2015. Aperiodic Two-way Transducers and FO-Transductions. In *Computer Science Logic (CSL) (LIPIcs)*, Vol. 41. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 160–174.

Christian Choffrut. 1977. Une Caractérisation des Fonctions Séquentielles et des Fonctions Sous-Séquentielles en tant que Relations Rationnelles. *Theor. Comput. Sci.* 5, 3 (1977), 325–337.

Christian Choffrut. 1979. A Generalization of Ginsburg and Rose's Characterization of G-S-M Mappings. In *Automata, Languages and Programming, 6th Colloquium, Graz, Austria, July 16-20, 1979, Proceedings*. 88–103.

Christian Choffrut. 2003. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.* 292, 1 (2003), 131–143.

Bruno Courcelle. 1994. Monadic Second-Order Definable Graph Transductions: A Survey. *Theor. Comput. Sci.* 126 (1994), 53–75.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Encyclopedia of mathematics and its applications, Vol. 138. Cambridge University Press.

K. Culik and J. Karhumaki. 1987. The Equivalence Problem for Single-Valued Two-Way Transducers (on NPDT0L Languages) is Decidable. *SIAM J. Comput.* 16, 2 (1987), 221–230.

Karel Culik II and Juhani Karhumäki. 1986. The Equivalence of Finite Valued Transducers (On HDT0L Languages) is Decidable. *Theor. Comput. Sci.* 47, 3 (1986), 71–84.

Luc Dartois, Emmanuel Filiot, Pierre-Alain Reynier, and Jean-Marc Talbot. 2016a. Two-Way Visibly Pushdown Automata and Transducers. In *Proc. 31st Annual IEEE Symposium on Logic in Computer Science (LICS'16)*. IEEE Computer Society. To appear.

Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. 2016b. Aperiodic String Transducers. In *Proc. 20th International Conference on Developments in Language Theory (DLT 2016) (Lecture Notes in Computer Science)*. Springer. To appear.

Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. 2016. A Generalized Twinning Property for Minimisation of Cost Register Automata. In *Proc. 31st Annual IEEE Symposium on Logic in Computer Science (LICS'16)*. IEEE Computer Society. To appear.

Rodrigo de Souza. 2008. On the Decidability of the Equivalence for k-Valued Transducers. In *Developments in Language Theory, 12th International Conference, DLT 2008, Kyoto, Japan, September 16-19, 2008. Proceedings (Lecture Notes in Computer Science)*, Vol. 5257. Springer, 252–263.

Volker Diekert, Paul Gastin, and Manfred Kufleitner. 2008. A Survey on Small Fragments of First-Order Logic over Finite Words. *Int. J. Found. Comput. Sci.* 19, 3 (2008), 513–548.

C. C. Elgot. 1961. Decision Problems of Finite Automata Design and Related Arithmetics. *In Transactions of the American Mathematical Society* 98, 1 (1961), 21–51.

C. C. Elgot and J. E. Mezei. 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development* 9 (1965), 47–68.

Joost Engelfriet and Hendrik Jan Hoogeboom. 2001. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.* 2, 2 (2001), 216–254.

Joost Engelfriet and Sebastian Maneth. 2003. Macro Tree Translations of Linear Size Increase are MSO Definable. *SIAM J. Comput.* 32, 4 (2003), 950–1006.

Emmanuel Filiot. 2015. Logic-Automata Connections for Transformations. In *Logic and Its Applications (ICLA)*. Springer, 30–57.

Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. 2016. First-order definability of rational transductions: an algebraic approach. In *Logic in Computer Science (LICS)*. IEEE.

Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. 2013. From Two-Way to One-Way Finite State Transducers. In *Logic in Computer Science (LICS)*. IEEE, 468–477.

Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. 2016. On Equivalence and Uniformisation Problems for Finite State Transducers. In *ICALP*. To appear.

Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. 2014. First-order Definable String Transformations. In *Foundation of Software Technology and Theoretical Computer Science, (FSTTCS) (LIPIcs)*, Vol. 29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 147–159.

Emmanuel Filiot, Jean-Franois Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. 2010. Properties of Visibly Pushdown Transducers. In *Proc. 35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10) (Lecture Notes in Computer Science)*, Vol. 6281. Springer, 355–367. DOI:http://dx.doi.org/10.1007/978-3-642-15155-2_32

T. V. Griffiths. 1968. The Unsolvability of the Equivalence Problem for Lambda-Free Nondeterministic Generalized Machines. *J. ACM* 15, 3 (1968), 409–413. DOI:http://dx.doi.org/10.1145/321466.321473

Eitan M. Gurari and Oscar H. Ibarra. 1983. A note on finite-valued and finitely ambiguous transducers. *Mathematical systems theory* 16, 1 (1983), 61–66.

Nathan Lhote. 2015. Towards an algebraic characterization of rational word functions. *CoRR* abs/1506.06497 (2015). http://arxiv.org/abs/1506.06497

Sebastian Maneth. 2015. A Survey on Decidable Equivalence Problems for Tree Transducers. *Int. J. Found. Comput. Sci.* 26, 8 (2015), 1069–1100. DOI:http://dx.doi.org/10.1142/S0129054115400134

Christophe Reutenauer and Marcel-Paul Schützenberger. 1991. Minimization of Rational Word Functions. *SIAM J. Comput.* 20, 4 (1991), 669–685.

Marcel Paul Schützenberger. 1975. Sur les relations rationnelles. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages, Kaiserslautern, May 20-23, 1975 (Lecture Notes in Computer Science)*, Vol. 33. Springer, 209–213.

Howard Straubing. 1994. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, Basel and Berlin.

Boris Avraamovich Trakhtenbrot. 1961. Finite automata and logic of monadic predicates (in Russian). *Dokl. Akad. Nauk SSSR* 140 (1961), 326–329.

Andreas Weber. 1993. Decomposing Finite-Valued Transducers and Deciding Their Equivalence. *SIAM J. Comput.* 22, 1 (1993), 175–202.

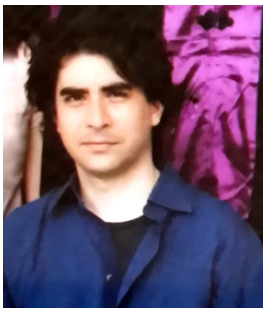
Andreas Weber and Reinhard Klemm. 1995. Economy of Description for Single-Valued Transducers. *Information and Computation* 118, 2 (1995), 327–340.

COMPLEXITY COLUMN

NEIL IMMERMAN, University of Massachusetts Amherst
immerman@cs.umass.edu

Hilbert's Nullstellensatz says roughly that there does not exist a solution of the polynomial equations $f_1(\overline{x}) = f_2(\overline{x}) = \cdots = f_m(\overline{x}) = 0$ iff there does exist a linear combination of these polynomials, $g_1(\overline{x})f_1(\overline{x}) + \cdots + g_m(\overline{x})f_m(\overline{x})$ that is identically equal to 1. I first heard the idea of using this in a proof system, in a talk by Toniann Pitassi at a 1996 DIMACS workshop that Phokion Kolaitis and I organized. Pitassi and Iddo Tzameret have now given us a clear explanation of dramatic progress in this area along with rich interconnections with deep work in algebraic complexity.

Algebraic Proof Complexity: Progress, Frontiers and Challenges



Toniann Pitassi
Department of Computer Science
University of Toronto

Iddo Tzameret
Department of Computer Science
Royal Holloway
University of London

We survey recent progress in the proof complexity of strong proof systems and its connection to algebraic circuit complexity, showing how the synergy between the two gives rise to new approaches to fundamental open questions, solutions to old problems, and new directions of research. In particular, we focus on tight connections between proof complexity lower bounds (namely, lower bounds on the size of proofs of certain tautologies), algebraic circuit lower bounds, and the Polynomial Identity Testing problem from derandomization theory.

1. INTRODUCTION

Propositional proof complexity aims to understand and analyze the computational resources required to prove propositional tautologies, in the same way that circuit complexity studies the resources required to compute boolean functions. A central question in the area asks whether every boolean tautology has a short propositional proof. Here, a propositional proof system can take many forms. One such proof system is the resolution refutation system whose proof-search algorithm constitutes the basis of current state of the art industrial-level SAT solvers (this thread of research was recently covered in SigLog; see Nordström [Nordström 2015]). For resolution and its *weak* extensions, strong lower bounds are known since Haken [Haken 1985]. But the major open questions in proof complexity, those originating from boolean circuit complexity and complexity class separations, such as P vs. NP, are about the length of *much stronger* proof systems than resolution, and these stronger systems will be the focus of this survey.

The prototypical strong proof system is the standard Hilbert-style propositional proof system, called *Frege proof system*, in which a proof starts from a fixed finite set of axioms and derives new propositional *formulas* using a fixed set of sound derivation rules. Establishing any super-polynomial size lower bound on such proofs (in terms of the size of the formula proved) is a major open problem in proof complexity, and a fundamental question in complexity theory.

The seminal work of Cook and Reckhow [Cook and Reckhow 1979] showed that in its strongest form, proof-size lower bound questions relate directly to fundamental hardness questions in computational complexity: establishing super-polynomial lower bounds for *every* propositional proof system would separate NP from coNP (and thus also P from NP).

The aim of this survey is to outline a currently thriving research direction, connecting algebraic circuit complexity to proof complexity. The prominent goal of this approach is the

quest for lower bounds on strong proof systems; other important aspects are connections to derandomization theory and application to feasible mathematics. The survey is meant to give some basic background in propositional proof complexity and describe the algebraic approach to proof complexity.

In what follows, Section 2 gives the basic definitions of algebraic circuits, propositional proof systems and algebraic proof systems, and a quick survey of the background results. Section 3 is devoted to the Ideal Proof System (IPS). In this section we show that IPS is closely connected to the Extended Frege proof system, and show that superpolynomial lower bounds for IPS proofs imply algebraic circuit lower bounds. Section 4 is devoted to the study of the non-commutative IPS. In this section we show that non-commutative IPS is equivalent (up to quasi-polynomial factors) to the Frege system and discuss the ramifications of this equivalence. Section 5 is dedicated to lower bounds for restricted subsystems of IPS. Section 6 discusses connections between algebraic proof complexity and the polynomial identity testing (PIT) problem and the use of structural results on algebraic circuits in proof complexity, as well as application to feasible mathematics. Finally we conclude with a discussion and open problems in Section 7.

2. BASIC CONCEPTS

For a natural number we let $[n] = \{1, \ldots, n\}$. Let $\mathbb{F}$ be a field. Denote by $\mathbb{F}[x_1, \ldots, x_n]$ the ring of (commutative) polynomials with coefficients from $\mathbb{F}$ and variables $x_1, \ldots, x_n$. In this survey, unless otherwise stated, we treat polynomials as *formal* linear combination of monomials, where a monomial is a product of variables. Hence, when we talk about the *zero polynomial* we mean the polynomial in which the coefficients of all monomials are zero (it can happen that over, say, $GF(2)$, $x^2 + x$ computes the zero *function*, but it is *not* the zero polynomial, because it has two nonzero monomial coefficients). Similarly, two polynomials are said to be *identical* if they have precisely the same monomial coefficients. The *degree* of a polynomial (or total degree) is the maximal sum of variable powers in a monomial with a nonzero coefficient in the polynomial. If the power of each variable in every monomial is at most 1 we say that the polynomial is *multilinear*. We write $\mathsf{poly}(n)$ to denote a polynomial growth in n, namely a function that is upper bounded by $n^{O(1)}$, and $\mathsf{qpoly}(n)$ to denote a quasi-polynomial growth in n, that is, $n^{\log^{O(1)} n}$.

2.1. Propositional Proof Systems

Cook and Reckhow [Cook and Reckhow 1979] defined a general concept of a propositional proof system from the perspective of computational complexity theory: a propositional proof system is a polynomial-time function f from a set of finite strings over some given alphabet *onto* the set of propositional tautologies (reasonably encoded). Thus, $f(x) = y$ means that the string x is a proof of the tautology y. Note that since f is onto, all tautologies and only tautologies have proofs (and thus the proof system is complete and sound).

The idea behind the Cook-Reckhow definition is that a purported proof x may be much longer than the tautology y it proves, but given a proof it should be possible to efficiently check (efficient with respect to the *proof length*, but not necessarily the tautology size) that it is indeed a correct proof of the tautology. We say that a propositional proof system is *polynomially bounded* if there exists a polynomial p that bounds the minimal proof size $|x|$ for every tautology y; namely, for every tautology y its minimal proof x is such that $|x| \leq \mathsf{poly}(|y|)$. Under the general Cook-Reckhow definition we have:

THEOREM 2.1 (COOK-RECKHOW [COOK AND RECKHOW 1979]). $\mathsf{NP}{=}\mathsf{coNP}$ *if and only if there is a polynomially bounded propositional proof system.*

Therefore, proving lower bounds against stronger and stronger propositional proof systems is clearly a formidable problem, as it can be considered as partial progress towards proving NP≠coNP (and thus P≠NP).

The definition of a propositional Cook-Reckhow proof system encompasses most standard proof systems for propositional tautologies, such as resolution and usual textbook proof system for propositional logic. In this survey we discuss specific propositional proof systems, that are at least as strong as the Frege or the Extended Frege system (see below). Though proving lower bounds on (Extended) Frege proof sizes for some families of tautologies would not amount to NP≠coNP, it would still constitute a breakthrough in complexity theory.

2.1.1. Frege Proof Systems. One of the most investigated and central propositional proof systems comes from the tradition of logic and is called the *Frege proof system*. A Frege proof system is any system that has a fixed number of axiom schemes and sound derivation rules, that is also implicationally complete[1], and in which proof lines are written as propositional *formulas*. It is known since Reckhow's work [Reckhow 1976] that all Frege proof systems are polynomially equivalent to each other, and hence it does not matter precisely which rules, axioms, and logical-connectives we use in the system. For concreteness, the reader can think of *the* ***Frege proof system*** as the following simple one (known as *Schoenfield's system*), consisting of only three axiom schemes (where $A \to B$ is an abbreviation of $\neg A \vee B$; and A, B, C are any propositional formulas):

$$A \to (B \to A)$$
$$(\neg A \to \neg B) \to ((\neg A \to B) \to A)$$
$$(A \to (B \to C)) \to ((A \to B) \to (A \to C)),$$

and a single inference rule (known as *modus ponens*):

$$\text{from } A \text{ and } A \to B, \text{ infer } B\,.$$

Frege systems are considered strong for several reasons. First, no super-polynomial lower bounds are known for Frege proofs, and moreover proving such lower bounds seems to be out of reach of current techniques, and believed by some to be even harder than proving explicit circuit lower bounds [Razborov 2015]. Secondly, hard candidates for Frege systems are hard to find; common tautologies such as the pigeonhole principle that are known to be hard for weaker proof systems have polynomial-size Frege proofs. (See [Bonet et al. 1995; Razborov 2015; Krajíček 2011; Li and Tzameret 2013] for further discussions on hard proof complexity candidates.)

An ***Extended Frege*** proof system is obtained by augmenting Frege with the axiom:

$$\text{Extension Axiom:} \qquad z \leftrightarrow \varphi\,,$$

where z is any *new* variable (namely, a variable that does not occur before in the proof) and φ is any formula (that does not contain z), and where the new variables z appearing in the extension axiom does not occur in the final formula in the proof. The point of the extension axiom is to allow the use of new variables to represent intermediate subformulas in a proof; with this new axiom scheme, polynomial-size Extended Frege proofs can reason about propositions computable by polynomial-size circuits (rather than just propositions computable by polynomial-size formulas, as is the case for polynomial-size Frege proofs).

For comprehensive texts on proof complexity and strong proof systems see e.g., the monograph by Krajíček [Krajíček 1995] and [Clote and Kranakis 2002, Chapter 5].

[1]Meaning that if a set of formulas Γ logically implies a formula φ, then there is a proof of φ in the system with formulas in Γ added to the axioms.

2.2. Comparing Proof Systems

To compare the relative strength of two proof systems we define the notion of a simulation. We say that a propositional proof system P **polynomially simulates** another propositional proof system Q if there is a polynomial-time computable function f that maps Q-proofs to P-proofs of the same tautologies (if P and Q use different representations for tautologies, we fix a (polynomial) translation from one representation to the other). In case f is computable in time $t(n)$ (for n the input-size), we say that P $t(n)$-*simulates* Q. We say that P and Q are *polynomially equivalent* in case P polynomially simulates Q and Q polynomially simulates P. If P polynomially simulates Q but Q does not polynomially simulate P we say that P *is strictly stronger than* Q (equivalently, that Q *is strictly weaker than* P).

2.3. Algebraic Circuits, Formulas, and Algebraic Complexity Classes

Algebraic circuits and formulas (over some fixed chosen field or ring) compute polynomials via addition and multiplication gates, starting from the input variables and constants from the field. More precisely, an *algebraic circuit* F is a finite directed acyclic graph with *input nodes* (i.e., nodes of in-degree zero) and a single *output node* (i.e., a node of out-degree zero). Input nodes are labeled with either a variable or a field element in $\mathbb{F}$. All the other nodes have in-degree two (unless otherwise stated) and are labeled by either $+$ or $\times$. An input node is said to *compute* the variable or scalar that labels itself. A $+$ (or $\times$) gate is said to compute the addition (product, resp.) of the polynomials computed by its incoming nodes. An algebraic circuit is called a *formula*, if the underlying directed acyclic graph is a tree (that is, every node has at most one outgoing edge). The *size* of a circuit is the number of nodes in it, and the *depth* of a circuit is the length of the longest directed path in it.

Algebraic Complexity Classes. We now recall some basic notions from algebraic complexity (for more details see [Shpilka and Yehudayoff 2010, Sec. 1.2]). Over a ring R, VP_R (for "Valiant's P") is the class of families $f = (f_n)_{n=1}^{\infty}$ of formal polynomials f_n such that f_n has $\mathsf{poly}(n)$ input variables, is of $\mathsf{poly}(n)$ degree, and can be computed by algebraic circuits over R of $\mathsf{poly}(n)$ size. VNP_R (for "Valiant's NP") is the class of families g of polynomials g_n such that g_n has $\mathsf{poly}(n)$ input variables and is of $\mathsf{poly}(n)$ degree, and can be written as

$$g_n(x_1, \ldots, x_{\mathsf{poly}(n)}) = \sum_{\overline{e} \in \{0,1\}^{\mathsf{poly}(n)}} f_n(\overline{e}, \overline{x})$$

for some family $(f_n) \in \mathsf{VP}_R$.

A polynomial $f(\overline{x})$ is a *projection* of a polynomial $g(\overline{y})$ if $f(\overline{x}) = g(L(\overline{x}))$ identically as polynomials in $\overline{x}$, for some map L that assigns to each y_i either a variable or a constant. A family of polynomials (f_n) is a polynomial projection or *p-projection* of another family (g_n) if there is a function $t(n) = n^{\Theta(1)}$ such that f_n is a projection of $g_{t(n)}$ for all (sufficiently large) n. The *permanent* polynomial $\sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}$ (for S_n the permutation group on n elements) is complete under p-projections for VNP. The *determinant* polynomial on the other hand is known to be in VP but is not known to be complete for VP under p-projections.

Two central questions in algebraic complexity theory are whether the permanent is a p-projection of the determinant (a stronger variant speaks about quasi-polynomial projections); and whether VP equals VNP [Valiant 1979a; Valiant 1979b; Valiant 1982]. Since the permanent is complete for VNP (under p-projections), showing $\mathsf{VP} \neq \mathsf{VNP}$ amounts to proving that the permanent cannot be computed by polynomial-size algebraic circuits.

2.4. Algebraic Proof Systems

Let us now describe several algebraic proof systems for propositional logic (i.e. for boolean tautologies). Assume we start from a set of initial polynomials (called *axioms*) $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ over some field $\mathbb{F}$, then (the weak version of) Hilbert's Nullstellensatz shows

that $f_1(\overline{x}) = \cdots = f_m(\overline{x}) = 0$ is unsatisfiable (over the algebraic closure of $\mathbb{F}$) if and only if there are polynomials $g_1, \ldots, g_m \in \mathbb{F}[\overline{x}]$ such that $\sum_j g_j(\overline{x}) f_j(\overline{x}) = 1$ (as a formal identity), or equivalently, that 1 is in the ideal generated by the $\{f_j\}_j$.

Beame, Impagliazzo, Krajíček, Pitassi, and Pudlák [Beame et al. 1996] suggested to treat these $\{g_j\}_j$ as a *proof* of the unsatisfiability of these axioms, called a ***Nullstellensatz refutation***. This is particularly relevant for complexity theory as one can restrict attention to *boolean* solutions to these axioms by adding the *boolean axioms*, that is, adding the polynomials $\{x_i^2 - x_i\}_{i=1}^n$ to the axioms. As such, one can then naturally encode NP-complete problems such as the satisfiability of 3CNF formulas as the satisfiability of a collection of constant-degree polynomials, and a Nullstellensatz refutation is then an equation of the form

$$\sum_{j=1}^{m} g_j(\overline{x}) f_j(\overline{x}) + \sum_{i=1}^{n} h_i(\overline{x})(x_i^2 - x_i) = 1$$

for $g_j, h_i \in \mathbb{F}[\overline{x}]$. This proof system is sound (only refuting unsatisfiable axioms over $\{0,1\}^n$) and complete (refuting any unsatisfiable axioms, by Hilbert's Nullstellensatz). Given that the above proof system is sound and complete, it is then natural to ask what is its power to refute unsatisfiable collections of polynomial equations over $\{0,1\}^n$. To understand this question one must define the notion of the *size* of the above refutations. Two popular notions are that of the *degree*, and the *sparsity* (number of monomials).

Strong (linear) lower bounds on Nullstellensatz degrees as well as strong (exponential) lower bounds on the sparsity of Nullstellensatz refutations are known (cf. [Beame et al. 1996; Buss et al. 1996; Razborov 1998; Grigoriev 1998; Impagliazzo et al. 1999; Buss et al. 2001; Alekhnovich and Razborov 2001] and references therein). Unfortunately, the hard examples used for these lower bounds *do* admit polynomial-size proofs in stronger proof systems like Frege.

Therefore, to correspond more accurately to Frege, strong algebraic proof systems must use a more economical representation of polynomials in proofs than sum of monomials (similarly to the way a boolean formula is a much more succinct representation of a boolean function than a mere CNF). The natural way is to measure the size of a polynomial by the size of the minimal algebraic circuit or formula that computes it.

The idea to consider algebraic circuit size of algebraic proofs was raised initially by Pitassi [Pitassi 1997] for Nullstellensatz written as algebraic circuits, and was investigated further in [Grigoriev and Hirsch 2003; Raz and Tzameret 2008b; Raz and Tzameret 2008a; Tzameret 2011] in the context of the polynomial calculus proof system.

Recently, Grochow and Pitassi [Grochow and Pitassi 2014] have suggested the following algebraic proof system that resembles the Nullstellensatz, but with a variant that proved to have important consequences. A proof in the Ideal Proof System is given as a ***single*** polynomial, lending itself quite directly to algebraic circuit complexity techniques. In what follows we follow the notation in [Forbes et al. 2016b]:

Definition 2.2 (*Ideal Proof System (IPS), Grochow-Pitassi [Grochow and Pitassi 2014]*). Let $f_1(\overline{x}), \ldots, f_m(\overline{x}) \in \mathbb{F}[x_1, \ldots, x_n]$ be a collection of polynomials. An **IPS refutation** for showing that the polynomials $\{f_j\}_j$ have no common solution in $\{0,1\}^n$ is an algebraic circuit $C(\overline{x}, \overline{y}, \overline{z}) \in \mathbb{F}[\overline{x}, y_1, \ldots, y_m, z_1, \ldots, z_n]$, such that

(1) $C(\overline{x}, \overline{0}, \overline{0}) = 0$.
(2) $C(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), x_1^2 - x_1, \ldots, x_n^2 - x_n) = 1$.

The **size** of the IPS refutation is the size of the circuit C. If C is of individual degree ≤ 1 in each y_j and z_i, then this is a **linear** IPS refutation (called basically *Hilbert* IPS by Grochow-Pitassi [Grochow and Pitassi 2014]), which is abbreviated as IPS_{LIN}. If C comes

from a restricted class of algebraic circuits $\mathcal{C}$, then this is called a $\mathcal{C}$-IPS refutation, and further called a $\mathcal{C}$-IPS$_{\text{LIN}}$ refutation if C is linear in $\overline{y}, \overline{z}$. The variables $\overline{y}, \overline{z}$ are sometimes called the *placeholder variables* since they use as a placeholder for the axioms.

Notice that the definition above adds the equations $\{x_i^2 - x_i\}_i$ to the system $\{f_j\}_j$. It is *not* necessary (for the sake of completeness) to add the equations $\overline{x}^2 - \overline{x}$ to the system in general, but this is the most interesting regime for proof complexity and thus we adopt it as part of our definition. Also, note that the first equality in the definition of IPS means that the polynomial computed by C is in the ideal generated by $\overline{y}, \overline{z}$, which in turn, following the second equality, means that C witnesses the fact that 1 is in the ideal generated by $f_1(\overline{x}), \ldots, f_m(\overline{x}), x_1^2 - x_1, \ldots, x_n^2 - x_n$ (the existence of this witness, for unsatisfiable set of polynomials, stems from the Nullstellensatz theorem as discussed above).

It is not hard to show that IPS$_{\text{LIN}}$ is polynomially equivalent to the Nullstellensatz system, when both are measured by their *circuit size*. For if we have an IPS$_{\text{LIN}}$ refutation $C(\overline{x}, \overline{y}, \overline{z})$ we can turn it into a Nullstellensatz refutation by writing it as a sum of products of the (linear) variables $\overline{y}, \overline{z}$, with only a quadratic increase in size. For instance, if we write $\overline{z}'$ to denote $\overline{z}$ without z_n, we have $C(\overline{x}, \overline{y}, \overline{z}) = C(\overline{x}, \overline{y}, \overline{z}', z_n) = C(\overline{x}, \overline{y}, \overline{z}', 0) + (C(\overline{x}, \overline{y}, \overline{z}', 1) - C(\overline{x}, \overline{y}, \overline{z}', 0)) \cdot z_n$. Now, since $C(\overline{x}, \overline{y}, \overline{z}', 0)$ does not contain the variable z_n we can continue in a similar way to "take out" the rest of the variables in $\overline{y}, \overline{z}$, one by one, reaching a Nullstellensatz refutation (when substituting the f_i's and the boolean axioms for the $\overline{y}, \overline{z}$, respectively).

Furthermore, Forbes, Shpilka, Tzameret and Wigderson [Forbes et al. 2016b] showed that IPS$_{\text{LIN}}$ refutations written as (general) algebraic circuits is *polynomially equivalent* to IPS (though for restricted classes $\mathcal{C}$, $\mathcal{C}$-IPS may differ from $\mathcal{C}$-IPS$_{\text{LIN}}$).

Considering both the Nullstellensatz and the IPS we can see that the main innovation in the IPS is the introduction of the placeholder variables $\overline{y}, \overline{z}$. This idea enables considering a refutation as a *single* polynomial instead of considering a collection of polynomials (that is, those polynomial coefficients of the initial axioms, as is the case of the Nullstellensatz).

Grochow-Pitassi [Grochow and Pitassi 2014] showed that the IPS system is very powerful and can simulate Extended Frege (this follows from the fact that IPS is a generalization of the Nullstellensatz written as algebraic circuits and already [Pitassi 1997] showed that the latter system simulates Extended Frege).

The fact that $\mathcal{C}$-IPS refutations are efficiently checkable (with randomness) follows from the fact that we only need to verify the polynomial identities stipulated by the definition. That is, it suffices to solve an instance of the ***polynomial identity testing*** *(PIT)* problem for the class $\mathcal{C}$: given a circuit from the class $\mathcal{C}$ decide whether it computes the identically zero polynomial. This problem is solvable in probabilistic polynomial time (BPP) for general algebraic circuits, and there are various restricted classes for which deterministic algorithms are known (see Section 6).

The Polynomial Calculus. The Polynomial Calculus is an algebraic proof system introduced by [Clegg et al. 1996]. It can be considered as a "dynamic" version of the Nullstellensatz; namely, instead of providing a single certificate that 1 is in the ideal of the initial (unsatisfiable) polynomials, in PC we are allowed to derive the polynomial 1 step by step, by working in the ideal generated by the initial polynomials.

Definition 2.3 (***Polynomial Calculus (PC)***). Let $\mathbb{F}$ be a field and let $F = \{f_1, \ldots, f_m\}$ be a collection of multivariate polynomials from $\mathbb{F}[x_1, \ldots, x_n]$. A *PC proof from Q of a polynomial g* is a finite sequence $\pi = (p_1, \ldots, p_\ell)$ of multivariate polynomials from $\mathbb{F}[x_1, \ldots, x_n]$, where $p_\ell = g$ and for every $1 \leq i \leq \ell$, either $p_i = f_j$ for some $j \in [m]$, or p_i is a boolean axiom $x_i \cdot (1 - x_i)$ for some $i \in [n]$, or p_i was derived from p_j, p_k, for $j, k < i$, by one of the following inference rules:

(i) *Product rule*: from p, derive $x_i \cdot p$, for $i \in [n]$;
(ii) *Addition rule*: from p, q, derive $ap + bq$, for $a, b \in \mathbb{F}$.

A *PC refutation of* F is a proof of 1 (which is interpreted as $1 = 0$, that is the unsatisfiable equation standing for false) from F.

Similar to the Nullstellensatz, the standard complexity measures for PC are the *degree* of a PC proof, which is the maximal (total) degree of a polynomial in the proof and the *size* of a PC proof which is the total number of monomials (with nonzero coefficients) in all the PC proof lines. However, it is also possible to consider the total algebraic circuit size of all the PC proofs lines as a complexity measure.

Non-commutative Algebraic Proof Systems. Motivated by the fact that the class of non-commutative formulas admits a deterministic PIT algorithm by Raz and Shpilka [Raz and Shpilka 2005a], and even more importantly admits exponential-size lower bounds by Nisan [Nisan 1991], Li, Tzameret and Wang [Li et al. 2015] considered a variant of the IPS over *non-commutative* polynomials written as non-commutative formulas. Their non-commutative IPS was shown to constitute a tighter characterization of Frege proofs than the original (commutative) IPS: first, proofs in this system are checkable in *deterministic* polynomial-time; and second, Frege can simulate (with a quasi-polynomial increase in size) non-commutative IPS refutations (over the field of two elements). But perhaps most importantly, the fact that we do have lower bounds on non-commutative formulas together with the characterization of any Frege proof as a single non-commutative formula, gives some hope to progress on the problem of Frege lower bounds. We discuss the non-commutative IPS in more details in Section 4.

3. IPS

In this section we show that lower bounds for IPS imply algebraic circuit lower bounds, namely that the permanent does not have polynomial-size algebraic circuits. This implication is interesting because it is a unique case in proof complexity where a lower bound on a *specific* proof system (on any tautology) is shown to imply explicit circuit lower bounds. We then compare the strength of IPS to Extended Frege. We show that IPS, in its full generality polynomially simulates Extended Frege, and on the other hand, show that Extended Frege polynomially simulates IPS if PIT has feasible correctness proofs in Extended Frege.[2]

3.1. Lower Bounds on IPS Imply Algebraic Circuit Lower Bounds

THEOREM 3.1 (GROCHOW-PITASSI [GROCHOW AND PITASSI 2014]). *For any ring R, a super-polynomial lower bound on IPS proofs over R of any family of tautologies implies $\mathsf{VNP}_R \neq \mathsf{VP}_R$. A super-polynomial lower bound on the number of proof-lines in polynomial calculus proofs implies that the permanent is not a p-projection of the determinant.*

We will sketch the proof for the first half of the theorem which gives the main idea. The proof of the second half can be found in [Grochow and Pitassi 2014].

LEMMA 3.2. *Every family of unsatisfiable CNF formulas (φ_n) has a family of IPS certificates (C_n) in VNP_R.*

Proof of Theorem 3.1, assuming Lemma 3.2. Our proof is taken from [Grochow and Pitassi 2014]. For a given set $\mathcal{F}$ of unsatisfiable polynomial equations $F_1 = \cdots = F_m = 0$, a lower bound on IPS refutations of $\mathcal{F}$ is equivalent to giving the same circuit lower bound on *all* IPS certificates for $\mathcal{F}$. A super-polynomial lower bound

[2]Namely, that Extended Frege has polynomial-size proofs of the statement expressing that the PIT for algebraic circuits is decidable by polynomial-size Boolean circuits.

on IPS implies that some function in VNP—namely, the VNP-IPS certificate guaranteed by Lemma 3.2—cannot be computed by polynomial-size algebraic circuits, and hence that VNP $\neq$ VP. □

Proof sketch of Lemma 3.2. We mimic one of the proofs of completeness for linear IPS [Pitassi 1997, Theorem 1] and then show that this proof can in fact be carried out in VNP. We omit any mention of the ground ring, as it will not be relevant.

Let $\varphi_n(\overline{x}) = \kappa_1(\overline{x}) \wedge \cdots \wedge \kappa_m(\overline{x})$ be an unsatisfiable CNF formula, where each κ_i is a disjunction of literals. Let $C_i(\overline{x})$ denote the (negated) polynomial translation of κ_i via $\neg x \mapsto x$, $x \mapsto 1-x$ and $f \vee g \mapsto fg$; in particular, $C_i(\overline{x}) = 0$ if and only if $\kappa_i(\overline{x}) = 1$, and thus φ_n is unsatisfiable if and only if the system of equations $C_1(\overline{x}) = \cdots = C_m(\overline{x}) = x_1^2 - x_1 = \cdots = x_n^2 - x_n = 0$ is unsatisfiable. In fact, as we will see in the course of the proof, we will not need the equations $x_i^2 - x_i = 0$. It will be convenient to introduce the function $b(e, x) = ex + (1-e)(1-x)$, i.e., $b(1, x) = x$ and $b(0, x) = 1 - x$. For example, the clause $\kappa_i(\overline{x}) = (x_1 \vee \neg x_{17} \vee x_{42})$ gets translated into $C_i(\overline{x}) = (1-x_1)x_{17}(1-x_{42}) = b(0, x_1)b(1, x_{17})b(0, x_{42})$, and therefore an assignment falsifies κ_i if and only if $(x_1, x_{17}, x_{42}) \mapsto (0, 1, 0)$.

Just as $1 = x_1x_2 + x_1(1-x_2) + (1-x_2)x_1 + (1-x_2)(1-x_1)$, an easy induction shows that

$$1 = \sum_{\overline{e} \in \{0,1\}^n} \prod_{i=1}^{n} b(e_i, x_i). \tag{1}$$

We will show how to turn this expression into a VNP certificate refuting φ_n. Let c_i be the placeholder variable corresponding to $C_i(\overline{x})$.

The idea is to partition the assignments $\{0,1\}^n$ into m parts $A_1, \ldots, A_m$, where all assignments in the i-th part A_i falsify clause i. This will then allow us to rewrite equation (1) as

$$1 = \sum_{i=1}^{m} C_i(\overline{x}) \left(\sum_{\overline{e} \in A_i} \prod_{j : x_j \notin \kappa_i} b(e_j, x_j) \right), \tag{2}$$

where "$x_j \notin \kappa_i$" means that neither x_j nor its negation appears in κ_i. Equation (2) then becomes the IPS-certificate $\sum_{i=1}^{m} c_i \cdot (\sum_{\overline{e} \in A_i} \prod_{j : x_j \notin \kappa_i} b(e_j, x_j))$. What remains is to show that the sum can indeed be rewritten this way, and that there is some partition $(A_1, \ldots, A_m)$ as above such that the resulting certificate is in fact in VNP.

First, let us see why such a partition allows us to rewrite (1) as (2). The key fact here is that the clause polynomial $C_i(\overline{x})$ divides the term $t_{\overline{e}}(\overline{x}) := \prod_{i=1}^{n} b(e_i, x_i)$ if and only if $C_i(\overline{e}) = 1$, if and only if $\overline{e}$ *falsifies* κ_i. Let $C_i(\overline{x}) = \prod_{i \in I} b(f_i, x_i)$, where $I \subseteq [n]$ is the set of indices of the variables appearing in clause i. By the properties of b discussed above, $1 = C_i(\overline{e}) = \prod_{i \in I} b(f_i, e_i)$ if and only if $b(f_i, e_i) = 1$ for all $i \in I$, if and only if $f_i = e_i$ for all $i \in I$. In other words, if $1 = C_i(\overline{e})$ then $C_i = \prod_{i \in I} b(e_i, x_i)$, which clearly divides $t_{\overline{e}}$. Conversely, suppose $C_i(\overline{x})$ divides $t_{\overline{e}}(\overline{x})$. Since $t_{\overline{e}}(\overline{e}) = 1$ and every factor of $t_{\overline{e}}$ only takes on boolean values on boolean inputs, it follows that every factor of $t_{\overline{e}}$ evaluates to 1 at $\overline{e}$, in particular $C_i(\overline{e}) = 1$.

Let $A_1, \ldots, A_m$ be a partition of $\{0,1\}^n$ such that every assignment in A_i falsifies κ_i. Since C_i divides every term $t_{\overline{e}}$ such that $\overline{e}$ falsifies clause i, C_i divides every term $t_{\overline{e}}$ with $\overline{e} \in A_i$, and thus we can indeed rewrite (1) as (2).

Next, we show how to construct a partition $A_1, \ldots, A_m$ as above so that the resulting certificate is in VNP. The partition we will use is a greedy one. A_1 will consist of *all* assignments that falsify κ_1. A_2 will consist of all *remaining* assignments that falsify κ_2. And so on. In particular, A_i consists of all assignments that falsify κ_i and *satisfy* all A_j with $j < i$. (If at some clause κ_i before we reach the end, we have used up all the assignments—

which happens if and only if the first i clauses on their own are unsatisfiable—that's okay: nothing we've done so far nor anything we do below assumes that all A_i are nonempty.)

Equivalently, $A_i = \{\overline{e} \in \{0,1\}^n | C_i(\overline{e}) = 1 \text{ and } C_j(\overline{e}) = 0 \text{ for all } j < i\}$. For any property Π, we write $[\![\Pi(\overline{e})]\!]$ for the indicator function of Π: $[\![\Pi(\overline{e})]\!] = 1$ if and only if $\Pi(\overline{e})$ holds, and 0 otherwise. We thus get the certificate:

$$\begin{aligned}
&\sum_{i=1}^{m} c_i \cdot \left(\sum_{\overline{e}\in\{0,1\}^n} [\![\overline{e} \text{ falsifies } \kappa_i \text{ and satisfies } \kappa_j \text{ for all } j < i]\!] \prod_{j:x_j \notin \kappa_i} b(e_j, x_j) \right) \\
= &\sum_{i=1}^{m} c_i \cdot \left(\sum_{\overline{e}\in\{0,1\}^n} [\![C_i(\overline{e}) = 1 \text{ and } C_j(\overline{e}) = 0 \text{ for all } j < i]\!] \prod_{j:x_j \notin \kappa_i} b(e_j, x_j) \right) \\
= &\sum_{i=1}^{m} c_i \cdot \left(\sum_{\overline{e}\in\{0,1\}^n} \left(C_i(\overline{e}) \prod_{j<i} (1 - C_j(\overline{e})) \right) \prod_{j:x_j \notin \kappa_i} b(e_j, x_j) \right) \\
= &\sum_{e\in\{0,1\}^n} \sum_{i=1}^{m} c_i C_i(\overline{e}) \left(\prod_{j<i} (1 - C_j(\overline{e})) \right) \left(\prod_{j:x_j \notin \kappa_i} b(e_j, x_j) \right)
\end{aligned}$$

Finally, it is readily visible that the polynomial function of $\overline{c}$, $\overline{e}$, and $\overline{x}$ that is the summand of the outermost sum $\sum_{\overline{e}\in\{0,1\}^n}$ is computed by a polynomial-size circuit of polynomial degree, and thus the entire certificate is in VNP. □

3.2. IPS Polynomially Simulates Extended Frege

In this section we show that IPS polynomially simulates Extended Frege. For simplicity, we exemplify this simulation by showing how IPS written as algebraic *formulas* polynomially simulates the Frege proof system, but the proof for Extended Frege is quite similar. It was further shown by [Grochow and Pitassi 2014] that restricted subsystems of IPS can polynomially simulate the corresponding restricted subsystem of Extended Frege, and specifically this holds for IPS written as constant-depth algebraic circuits and constant-depth Frege systems with modulo counting gates ($\mathsf{AC}^0[p]$-Frege).

THEOREM 3.3 (GROCHOW-PITASSI [GROCHOW AND PITASSI 2014]). *Let φ be a 3CNF formula. If there is an Extended Frege proof (Frege proof) that φ is unsatisfiable in size-s, then there is an IPS refutation of circuit (formula, resp.) size* $\mathsf{poly}(|\varphi|, s)$.

Proof sketch. One way of thinking of this simulation (and similar simulations of propositional systems by IPS-variants) is to consider a two-step conversion of propositional proofs into IPS refutations as follows. First, turn the Frege proof into a tree-like Frege proof and convert each proof-line in the tree-like Frege proof into an equivalent algebraic formula, obtaining a tree-like PC proof. Secondly, convert the tree-like PC proof into a single formula, whose underlying formula-tree is precisely the underlying tree of the PC proof.

Note that a Frege proof can be converted into a *tree-like proof* with only a polynomial increase in size, that is, a proof in which every proof-line can be used at most once in modus ponens (cf. [Krajíček 1995]). Therefore, we start from a tree-like Frege proof of a propositional formula (possibly from assumptions), and show how to obtain from this an IPS proof of the arithmetic version (see below) of the same formula. Thus, a Frege proof of false from a given CNF φ is translated into an IPS proof of 1 from the initial (arithmetic version of) φ, yielding an IPS refutation of φ.

Step 1: This step involves the arithmetization of Frege proofs, namely, converting each Frege proof-line to an algebraic formula. Note that every Frege proof-line (e.g, in the Schoen-

field's system) is a boolean *tautological* formula. The transformation converts a boolean tautology into a corresponding algebraic formula (over the rationals, or $\mathbb{F}_q$, for a prime q; the simulation uses only the fact that the field has $1, 0$ and -1) that evaluates to 0 for all 0-1 assignments. This is done in the same way as in the proof of Lemma 3.2: true becomes 0, false becomes 1, a variable x_i becomes $1-x_i$, $\neg A$ becomes $1-\mathrm{tr}(A)$, where $\mathrm{tr}(A)$ denotes the translation of A, $A \vee B$ becomes the product of the corresponding translations $\mathrm{tr}(A) \cdot \mathrm{tr}(B)$, and $A \wedge B$ becomes $1 - (1 - \mathrm{tr}(A)) \cdot (1 - \mathrm{tr}(B))$ (Schoenfield's system uses the $\rightarrow$ logical connective, but we can simply treat $A \rightarrow B$ as an abbreviation of $\neg A \vee B$). It is easy to check that indeed A is a tautology iff $\mathrm{tr}(A) = 0$ for every 0-1 assignment.

Once we converted every Frege proof-line into its corresponding algebraic formula, we get something that *resembles* a sequential algebraic proof, namely a PC proof. However, it is not precisely a legitimate PC proof because, e.g., every application of modus ponens (from A and $A \rightarrow B$ derive B) is translated into the purported rule "*from* $\mathrm{tr}(A)$ *and* $(1-\mathrm{tr}(A)) \cdot \mathrm{tr}(B)$ *derive* $\mathrm{tr}(B)$", which is not a formal rule in PC. Nevertheless, we can make this arithmetized Frege proof into a legitimate PC proof, except that our PC proof will have a *generalized* product rule: instead of being able to multiply a proof-line only by a *single* variable we will enable a product by a *polynomial*, namely, from f derive $g \cdot f$, for some polynomial $g \in \mathbb{F}[x_1, \ldots, x_n]$.

To form our (generalized) PC proof we simply *simulate* Schoenfield's system rules and axioms. Considering the example above, we need to construct a short PC proof of $\mathrm{tr}(B)$ from $\mathrm{tr}(A)$ and $(1 - \mathrm{tr}(A)) \cdot \mathrm{tr}(B)$: first derive $\mathrm{tr}(A) \cdot \mathrm{tr}(B)$ by the generalized PC product rule and then add this to $(1 - \mathrm{tr}(A)) \cdot \mathrm{tr}(B)$, to obtain $\mathrm{tr}(B)$. Similarly, Frege axioms are translated into algebraic formulas, that we then need to *derive* in PC, and this is possible to do efficiently.

Step 2: Here we transform the (generalized) PC proof from step 1, whose underlying proof-graph is a tree (since we assumed without loss of generality that our initial Frege proof is a tree-like proof), into a *single formula* whose underlying graph is essentially the same tree. This formula constitutes the IPS proof of the arithmetic translation of φ. The transformation from a PC proof to a formula is quite straightforward. For example, assume that in the PC proof we derived $g \cdot f$ from f. And suppose that we already built the IPS proof of f, namely $C(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), x_1^2 - x_1, \ldots, x_n^2 - x_n) = f$. Then, $g \cdot C(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), x_1^2 - x_1, \ldots, x_n^2 - x_n) = g \cdot f$ is the IPS proof of $g \cdot f$. Simulating the addition rule of PC is done in a similar manner. (Formally, the $f_i(\overline{x})$'s should be substituted by the placeholder variables $\overline{y}$, and the boolean axioms by the placeholder variables $\overline{z}$.)

It is easy to see that the resulted IPS is of size polynomial in the size of the PC proof, which in turn is of size polynomial in the size of the original Frege proof. □

3.3. PIT as a Bridge Between Circuit Complexity and Proof Complexity

In this section we sketch the argument that Extended Frege (EF) is polynomially equivalent to IPS if there are polynomial-size circuits for PIT whose correctness—suitably formulated—can be efficiently proved in EF. More precisely, we identify a small set of natural axioms for PIT and show that if these axioms can be proven efficiently in EF, then EF is p-equivalent to IPS.

The high-level idea is to formalize soundness of IPS as a sequence of propositional statements and then to show:

(1) if EF has efficient proofs of IPS soundness then EF can polynomially simulate IPS;
(2) Show that EF has efficient proofs of IPS soundness if a small set of natural axioms for PIT are efficiently provable in EF.

The idea behind (1) is not new and traces back to Hilbert; its counterpart for propositional proof systems was first formalized by Cook [Cook 1975]. We explain the idea for proposi-

tional refutation systems here. Soundness of a propositional proof system states that any formula that has a proof (in the system) is a tautology. Formalizing soundness propositionally involves studying *partial* soundness, where we have a different propositional formula for each proof length. In more detail, for a propositional proof system Q, $Soundness_{Q,n}, n > 0$ will be a family of propositional statements. The underlying variables of $Soundness_{Q,n}$ are $\overline{x}$, $\overline{y}$ and $\overline{z}$, where we think of $\overline{x}$ as an encoding of some Q-proof of length n, $\overline{y}$ as an encoding of a k-DNF formula with $n' \leq n$ underlying variables, and $\overline{z}$ as a boolean assignment to the n' underlying variables. $Soundness_{Q,n}(\overline{x}, \overline{y}, \overline{z})$ is of the form $Proof_{Q,n}(\overline{x}, \overline{y}) \rightarrow Truth(\overline{y}, \overline{z})$ where $Proof_{Q,n}(\overline{x}, \overline{y})$ expresses that $\overline{x}$ is an encoding of a Q-proof of the formula encoded by $\overline{y}$, and $Truth(\overline{y}, \overline{z})$ expresses that $\overline{z}$ satisfies the formula encoded by $\overline{y}$ (i.e., the formula encoded by $\overline{y}$ is a tautology).

For sufficiently strong propositional proof systems P and Q, it is well-known that P polynomially simulates Q if and only if there are polynomial-sized P-proofs of $Soundness_{Q,n}$ for all $n > 0$. The intuitive argument is as follows: Suppose that Q has a short proof of some formula g; let $\alpha(g)$ be the encoding of g, and let $\beta(g)$ be the encoding of the short Q-proof of g. Then since P has short proofs of $Soundness_{Q,n}$, we instantiate this with g to give a short P-proof of $Soundness_{Q,n}(\beta(g), \alpha(g), \overline{z})$. Since $Proof_{Q,n}(\beta(g), \alpha(g))$ is a tautology and involves no propositional variables, it has a short P-proof and thus by modus ponens, there is a short P-proof of $Truth(\alpha(g), \overline{z})$. The last step is to demonstrate short P-proofs of $Truth(\alpha(g), \overline{z}) \rightarrow g$.

We will take P to be EF and Q to be IPS. Then EF can polynomially simulate Q if and only if EF can efficiently prove the soundness tautologies for IPS. Proving the soundness tautologies for IPS amounts to stating and proving (in Extended Frege) that if C is an algebraic circuit such that: (1) $C(\overline{x}, \overline{0}, \overline{0}) = 0$ and (2) $C(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x})) = 1$, then $f_1, \ldots, f_m$ is unsatisfiable. In order to state (1) and (2) efficiently, we need polynomial-sized circuits for polynomial identity testing. Then in order to prove that (1) and (2) imply that $f_1, \ldots, f_m$ is unsatisfiable, we will need to use basic properties of our PIT circuits. We omit the proof here, but will informally state the axioms that will be required in order to carry out the above plan.

3.4. Axioms for Circuits for Polynomial Identity Testing

Fix some standard boolean encoding of algebraic circuits, so that the encoding of any size-m algebraic circuit has size $\mathsf{poly}(m)$. We use "$[C]$" to denote the encoding of the algebraic circuit C. Let $K = (K_{m,n})$ denote a family of boolean circuits for solving polynomial identity testing. That is, $K_{m,n}$ is a boolean function that takes as input the encoding of a size m algebraic circuit, C, over variables $x_1, \ldots, x_n$, and if C has polynomial degree, then K outputs 1 if and only if the polynomial computed by C is the 0 polynomial.

The first axiom states that if C is a circuit over variables $\overline{x}$ computing the identically 0 polynomial, then the circuit C where we plug in a particular boolean input $\overline{p}$, still computes the identically 0 polynomial:

$$K([C(\overline{x})]) \rightarrow K([C(\overline{p})]).$$

The second axiom states that if C is a circuit over variables $\overline{x}$ computing the zero polynomial, then the circuit $1 - C$ does not compute the zero polynomial:

$$K([C(\overline{x})]) \rightarrow \neg K([1 - C(\overline{x})]).$$

The third axiom states that if the polynomial computed by circuit G is 0, then G can be substituted for the constant 0:

$$K([G(\overline{x})]) \wedge K([C(\overline{x}, 0)]) \rightarrow K([C(\overline{x}, G(\overline{x}))]).$$

Finally, the last axiom states that PIT is closed under permutations of the variables. More specifically if $C(\overline{x})$ is identically 0, then so is $C(\pi(\overline{x}))$ for all permutations π:

Note that the issue is not the existence of small circuits for PIT since we would be happy with nonuniform polynomial-size PIT circuits, which do exist. Unfortunately the known constructions are highly nonuniform—they involve picking random points—and we do not see how to prove the above axioms for these constructions. On the other hand, it is widely conjectured that there exist uniform polynomial-sized circuits for PIT, and it is therefore a very intriguing question whether or not the proofs of correctness of such uniform algorithms (assuming that they exist) can be carried out in a feasible (polynomial-time) proof system.

4. THE NON-COMMUTATIVE IPS

In this section we discuss the non-commutative IPS, introduced by Li, Tzameret and Wang [Li et al. 2015], which is a variant of the IPS over non-commutative polynomials. The main result is that the non-commutative IPS completely captures (up to quasi-polynomial factors) the Frege proof system when the non-commutative IPS refutations are written as non-commutative formulas.

Since the class of non-commutative formulas are well understood, namely, it admits exponential-size lower bounds by Nisan [Nisan 1991], and deterministic PIT algorithm by Raz-Shpilka [Raz and Shpilka 2005a], this characterization of a Frege proof by a single non-commutative formula gives some hope for better understanding of specific Frege proofs and specifically for the eventual possibility of providing lower bounds on Frege proofs.

We need to describe first the basic setup before giving the precise definition. A ***non-commutative polynomial*** is a polynomial in which products are non-commuting, namely, x_ix_j is not the same polynomial as x_jx_i, whenever $i \neq j$. In other words, $x_ix_j - x_jx_i$ is not the zero polynomial. Thus, we can treat a non-commutative polynomial as a formal sum of non-commutative monomials. We denote by $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ the ring of non-commutative polynomials over the variables $x_1, \ldots, x_n$. A ***non-commutative formula*** is the same as a (commutative) algebraic formula only that the children of product gates have *order*, so that we can record the order of multiplication. Therefore, the polynomial that a non-commutative formula computes is the polynomial achieved by first multiplying out brackets whereby we get a sum of monomials in which the order of multiplication matters (without performing still any cancelations of monomials), and then performing monomial cancelation (and grouping) only when two monomials have the same variables with the same powers and *the same order of multiplication*.

It helps to think of non-commutative polynomials (and formulas) as a means to compute functions over non-commutative domains such as matrix algebras (in which matrix product is non-commuting in general).

Definition 4.1 (*Non-commutative IPS, Li-Tzameret-Wang [Li et al. 2015]*). Let $\mathbb{F}$ be a field. Let $f_1(\overline{x}), \ldots, f_m(\overline{x}) \in \mathbb{F}\langle\overline{x}\rangle$ be a system of non-commutative polynomials. A **non-commutative-IPS refutation** that the polynomials $\{f_j\}_j$ have no common solution in $\{0,1\}^n$,[3] is a non-commutative formula $\mathfrak{F}(\overline{x}, \overline{y}, \overline{z}, \overline{w}) \in \mathbb{F}\langle\overline{x}, y_1, \ldots, y_m, z_1, \ldots, z_n, w_1, \ldots, w_{\binom{n}{2}}\rangle$, such that

(1) $\mathfrak{F}(\overline{x}, \overline{0}, \overline{0}, \overline{0}) = 0$.
(2) $\mathfrak{F}(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), \overline{x}^2 - \overline{x}, x_1x_2 - x_2x_1, \ldots, x_{n-1}x_n - x_nx_{n-1}) = 1$.

The $\overline{x}^2 - \overline{x}$ denotes the boolean axioms $x_i^2 - x_i$, for all $i \in [n]$, and $x_ix_j - x_jx_i$, for all $i < j \in [n]$, are called the *commutator axioms*. The **size** of a non-commutative IPS refutation is the minimal size of a *non-commutative formula* computing the non-commutative-IPS refutation.

[3] One can check that the $f_i(\overline{x})$'s have no common 0-1 solutions in $\mathbb{F}$ iff they do not have a common 0-1 solution in every $\mathbb{F}$-algebra.

The novelty in the non-commutative IPS in comparison to the original (commutative) IPS is simply that a single refutation is a non-commutative polynomial instead of a commutative one.

One way of thinking about a non-commutative IPS refutation is as a *commutative* IPS formula augmented with *additional proofs* for demonstrating that all the monomials computed along the way in this formula are indeed commuting. More precisely, consider a *commutative* IPS refutation written as a formula $F(\overline{x}, \overline{y}, \overline{z})$, such that $F(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), \overline{x}^2 - \overline{x}) = 1$ as a *commutative* formula but $F(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), \overline{x}^2 - \overline{x}) \neq 1$ as a non-commutative formula. In the commutative version of $F(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}), \overline{x}^2 - \overline{x})$, two monomials computed by this refutation, say $x_1x_2x_3$ and $x_2x_1x_3$, will be considered the same monomial. However, in the non-commutative version these two monomials are considered distinct, so we need to add an explicit proof that $x_1x_2x_3$ is equal to $x_2x_1x_3$—in this case the proof added is simply $(x_1x_2 - x_2x_1) \cdot x_3$ which is a right product of a commutator axiom.

Note that to achieve the *completeness* of the system we *must* add the commutator axioms. Indeed, the non-commutative polynomial $1 + x_ix_j - x_jx_i$, for example, is unsatisfiable over 0-1 solutions, but it cannot be proven unsatisfiable without using the commutator axioms, because it *is* satisfiable over some non-commutative matrix algebra (so by soundness of the non-commutative IPS there cannot be a proof of its unsatisfiability).

The gist of Li-Tzameret-Wang's simulation of Frege by the non-commutative IPS is that even when we add the commutator axioms, and by that (informally speaking) force each refutation to explicitly witness any cancelation between (commuting) monomials, we are still not weakening the system too much, namely, we still keep the system as strong as the Frege system. And the reason for this is that in Frege we consider propositional formulas as purely *syntactic* terms. For example, if $F[z]$ is a propositional formula, then $F[(A \wedge B)/z]$ and $F[(B \wedge A)/z]$ (which are the results of substituting $A \wedge B$ and $B \wedge A$ for z in F, resp.) are two *different* formulas and the tautology $F[(A \wedge B)/z] \equiv F[(B \wedge A)/z]$ requires an explicit Frege proof.

Li et al. showed that non-commutative IPS characterizes Frege proofs: non-commutative IPS polynomially simulates Frege, and conversely, Frege quasi-polynomially simulates non-commutative IPS over $GF(2)$ (for the latter see next section):

THEOREM 4.2 (LI-TZAMERET-WANG [LI ET AL. 2015]). *Let φ be an unsatisfiable propositional formula. If Frege can prove that φ is unsatisfiable in size-s, then there is a non-commutative IPS refutation of formula size $\mathsf{poly}(|\varphi|, s)$ computing a polynomial of degree $\mathsf{poly}(|\varphi|, s)$. Further, this refutation is checkable in deterministic $\mathsf{poly}(|\varphi|, s)$ time.*

The idea to consider non-commutative formulas in algebraic proofs as well as adding the commutator axioms was considered first by Tzameret [Tzameret 2011], though in that work the proof system was built on the polynomial calculus and not the IPS, and therefore did not obtain the characterization of a Frege proof as a single non-commutative formula.

Let us sketch the proof of 4.2. We begin with the simulation of Frege by non-commutative IPS. The idea here is quite similar to the simulation of Frege by (formula) IPS (Theorem 3.3).

Non-commutative IPS polynomially simulates Frege (proof sketch). Let us consider, as in the proof of Theorem 3.3, a two-step simulation of Frege by non-commutative IPS. We start from a Frege proof, that we assume without loss of generality is a tree-like proof, of a tautology φ.

Step 1: Here we convert each proof-line into an algebraic formula in the same way we did in the proof of Theorem 3.3, using the same translation function $\mathrm{tr}(\cdot)$, only now let $tr(\cdot)$ return a *non-commutative formula*. So, for instance, assuming A and B are unequal, $\mathrm{tr}(A \vee B) = \mathrm{tr}(A) \cdot \mathrm{tr}(B) \neq \mathrm{tr}(B) \cdot \mathrm{tr}(A) = \mathrm{tr}(B \vee A)$ (note that any algebraic formula can represent either a commutative or a non-commutative polynomial; namely, a non-commutative

formula computes a non-commutative polynomial by taking into account the order in which children of product gates appear in the formula).

Now, as before, we get a purported proof of $\mathrm{tr}(\varphi)$ that only resembles a PC proof and in addition the polynomials are non-commutative. We wish to complement this purported proof into a legitimate algebraic proof operating with non-commutative polynomials; this will be in fact the *non-commutative PC* system defined by Tzameret [Tzameret 2011]: it is similar to the PC proof system, only that polynomials are considered as non-commutative polynomials, the addition rule is the same as in PC, and the generalized product rule can be applied either from the right or from the left, namely, from f derive either $g \cdot f$ or $f \cdot g$, for some g; further, in addition to the boolean axioms, we add the commutator axioms $x_i x_i - x_j x_i$, for every pair of variables, to the system. The commutator axioms are crucial for completeness (over 0-1 assignments).

We consider the case of simulating the first axiom of Schoenfield's system $A \to (B \to A)$ in this non-commutative PC system. This will exemplify why we need to use the commutator axioms. Thus, consider the translation of this axiom under $\mathrm{tr}(\cdot)$. Recall that $\to$ is just an abbreviation. Then, $\mathrm{tr}(A \to (B \to A)) = \mathrm{tr}(\neg A \vee (\neg B \vee A)) = (1-\mathrm{tr}(A)) \cdot ((1-\mathrm{tr}(B)) \cdot \mathrm{tr}(A))$. Our goal is to construct a non-commutative PC proof of the following *non-commutative* polynomial:

$$(1 - \mathrm{tr}(A)) \cdot ((1 - \mathrm{tr}(B)) \cdot \mathrm{tr}(A)) \,. \tag{3}$$

For this purpose, we first derive the polynomial $\mathrm{tr}(A) - \mathrm{tr}(A)^2 = (1 - \mathrm{tr}(A)) \cdot \mathrm{tr}(A)$. This is doable efficiently using only the boolean axioms $x_i - x_i^2$ (by induction on the size of A). Then, we wish to derive (3) from $(1-\mathrm{tr}(A)) \cdot \mathrm{tr}(A)$. We can multiply the latter by $(1-\mathrm{tr}(B))$ from the right, to get $(1 - \mathrm{tr}(A)) \cdot \mathrm{tr}(A) \cdot (1 - \mathrm{tr}(B))$. Now we *must use the commutator axioms* $x_i x_j - x_j x_i$ to commute the rightmost product in order to derive (3).

Indeed, given the product of two formulas $f \cdot g$, it is possible to show by induction on the size of f, g, that using the commutator axioms one can derive with a size $|f + g|$ non-commutative PC proof the formula $g \cdot f$.

Step 2: Here we repeat almost precisely the same idea as in Step 2 of the proof of Theorem 3.3. We have a tree-like non-commutative PC proof of $\mathrm{tr}(\varphi)$ (that possibly uses the commutator axioms) and we wish to turn it into a non-commutative formula that constitutes an IPS proof of $\mathrm{tr}(\varphi)$. We do this by constructing a non-commutative formula whose underlying graph is the same underlying proof-graph, as we did before. □

4.1. Frege Quasi-Polynomially Simulates the Non-Commutative IPS

THEOREM 4.3 (LI-TZAMERET-WANG [LI ET AL. 2015]). *Let φ be an unsatisfiable CNF formula and $f_1, \ldots, f_m$ be the non-commutative formulas corresponding to its clauses via $tr(\cdot)$. If there is a non-commutative IPS refutation of size s of $f_1, \ldots, f_m$ over $GF(2)$, then there is a Frege proof of size $s^{O(\log s)}$ of the tautology $\neg\varphi$.*

For low-degree non-commutative IPS refutations, the proof of Theorem 4.3 achieves in fact a slightly stronger simulation than stated. Specifically, when the degree of the non-commutative IPS refutation is logarithmic in s, the Frege proof is of polynomial-size in s (see [Li et al. 2015] for details).

The higher-level argument is a short Frege proof of the correctness of the Raz-Shpilka [Raz and Shpilka 2005a] deterministic PIT algorithm. This resembles the discussion in Section 3.3 about PIT for (commutative) circuits. Indeed, the argument can be viewed as a realization—for the non-commutative case—of Grochow-Pitassi [Grochow and Pitassi 2014] PIT-axioms framework (Section 3.4). The actual proof of Theorem 4.3 is rather technical and long because one needs to prove properties of the Raz-Shpilka PIT algorithm for non-commutative formulas within the restrictive framework of propositional logic (that is, Frege proofs). Let us sketch the main ideas in the proof.

Our goal is to prove $\neg\varphi$ in Frege, given a non-commutative IPS refutation π of φ. The proof uses the following main five steps. First, we *balance* the non-commutative IPS π, so that its depth is logarithmic in its size. This follows Hrubeš and Wigderson's [Hrubeš and Wigderson 2014] construction. Second, consider the balanced π, which is a non-commutative polynomial identity over $GF(2)$, as a *boolean tautology*, by replacing plus gates with XORs and product gates with ANDs. Third, we use the so-called *reflection principle* to reduce the task of efficiently proving $\neg\varphi$ in Frege to the following task: show that any non-commutative formula identity over $GF(2)$, considered as a boolean tautology, has a short Frege proof (this part was discussed—for the commutative case—in Section 3.3). Fourth, for technical reasons we turn our non-commutative polynomial identities over $GF(2)$ (considered as boolean tautological formulas) into a sum of *homogenous* identities. This is the *only* step that is responsible for the *quasi-polynomial size increase* in Theorem 4.3. For this step we use an efficient Frege simulation of a result by Raz [Raz 2013] who showed how to transform an arithmetic formula into (a sum of) homogenous formulas in an efficient manner.

The fifth and final step is to actually construct short Frege proofs for homogenous non-commutative identities. To this end we construct an efficient Frege proof of the correctness of the Raz-Shpilka PIT algorithm for non-commutative formulas [Raz and Shpilka 2005b].

In conclusion, the fact that IPS written as a non-commutative formula (with the additional commutator axioms) characterizes Frege proofs, naturally motivates studying $\mathcal{C}$-IPS for various restricted classes $\mathcal{C}$ of algebraic circuits. Lower bounds for such proofs intuitively correspond to lower bounds for restrictions of the Extended Frege proof system. This is the content of the next section, using the recent lower bounds of Forbes, Shpilka, Tzameret and Wigderson [Forbes et al. 2016b].

5. LOWER BOUNDS ON FRAGMENTS OF IPS

In Section 3.1 we have seen that proving super-polynomial lower bounds on the size of IPS certificates (written as algebraic circuits) would imply a separation of VP from VNP. On the other hand, in Section 4 we have seen that already proving lower bounds on IPS certificates when they are written as non-commutative formulas and augmented with the commutator axioms would imply Frege lower bounds. It is then natural to attempt to obtain lower bounds on IPS refutations where the certificates are written as an algebraic circuit from a restricted circuit class $\mathcal{C}$. Recall the notation $\mathcal{C}$-IPS from Definition 2.2, denoting that the IPS certificate $C(\overline{x},\overline{y},\overline{z})$ is taken from the class $\mathcal{C}$.[4] If the "placeholder" variables $\overline{y},\overline{z}$ in C are linear we call the certificate a $\mathcal{C}$-IPS$_{\text{LIN}}$ certificate. Super-polynomial lower bounds on the size of $\mathcal{C}$-IPS$_{\text{LIN}}$ refutations were recently shown by Forbes, Shpilka, Tzameret and Wigderson [Forbes et al. 2016b] when $\mathcal{C}$ is the class of read once (oblivious) algebraic branching programs (roABPs), multilinear formulas and diagonal circuits. We now survey some of these lower bounds.

Let us describe the main strategy behind the proofs, which is new, and exemplifies the potential of the algebraic complexity-based approach in proof complexity. One feature of these proof-size lower bounds is that they stem almost directly from circuit-size lower bounds.

Assume that $f(\overline{x})=0$ has no 0-1 solutions over some field $\mathbb{F}$, and let

$$g(\overline{x})\cdot f(\overline{x})+\sum_{i=1}^{n} h_i(\overline{x})\cdot(x_i^2-x_i)=1\,, \tag{4}$$

be the Nullstellensatz (equivalently, IPS$_{\text{LIN}}$) refutation of $f(\overline{x})=0$. We are going to lower bound the size of circuits computing $g(\overline{x})$. If we restrict our attention in (4) to only 0-1

[4]Note that there is a slight technical difference between requiring that $C(\overline{x},\overline{y},\overline{z})$ is taken from $\mathcal{C}$ and requiring that $C(\overline{x},\overline{F}(\overline{x}),x_1^2-x_1,\ldots,x_n^2-x_n)$ is taken from $\mathcal{C}$. In $\mathcal{C}$-IPS we require the former.

assignments, then the boolean axioms vanish, and we have

$$g(\overline{x}) \cdot f(\overline{x}) = 1\,, \qquad \text{for } \overline{x} \in \{0,1\}^n, \tag{5}$$

where (5) is now a *functional identity* (in contrast to a *formal* identity between polynomials). That is, we consider $g(\overline{x}) \cdot f(\overline{x})$ as a function from $\{0,1\}^n$ to $\mathbb{F}$, and conclude that this is the constant 1 function. Hence

$$g(\overline{x}) = \frac{1}{f(\overline{x})}\,, \qquad \text{for } \overline{x} \in \{0,1\}^n\,.$$

Therefore, to lower bound the algebraic circuit size of Nullstellensatz refutations of $f(\overline{x})$ it suffices to lower bound the algebraic circuit size of every polynomial that computes the *function* $1/f(\overline{x})$ over 0-1 assignments. Since we wish to prove a lower bound for a family of polynomials computing a certain function over 0-1, instead of a lower bound for a specific formal polynomial, this kind of lower bound is called a *functional lower bound* (see also [Grigoriev and Razborov 2000; Forbes et al. 2016a]). Forbes et al. [Forbes et al. 2016b] showed that for $f(\overline{x})$ being a variant of the subset-sum principle $\sum_{i=1}^{n} \alpha_i x_i - m$, for $\alpha_i \in \mathbb{F}$ and $m \notin \{\sum_{i=1}^{n} \alpha_i x_i \;:\; \overline{x} \in \{0,1\}^n\}$, one can obtain strong functional lower bounds on the algebraic circuit size of $1/f(\overline{x})$, for certain circuit classes, and thus conclude IPS refutations lower bounds for these circuit classes.

The lower bounds obtained in [Forbes et al. 2016b], stated below, are for IPS over multilinear formulas and read once (oblivious) algebraic branching programs (roABP). A *multilinear formula* is simply an algebraic formula (Section 2.3) in which every node computes a multilinear polynomial. For the definition or roABPs and the proof of the lower bounds we refer the reader to [Forbes et al. 2016b].

THEOREM 5.1 (FORBES-SHPILKA-TZAMERET-WIGDERSON [FORBES ET AL. 2016B]). *Let $n \geq 1$ and $\mathbb{F}$ be a field with characteristic bigger than $\binom{2n}{n}$. Suppose that $f(\overline{x},\overline{z}) = \sum_{i<j\in[2n]} z_{i,j} x_i x_j - m$ is a polynomial over $\mathbb{F}$ that has no 0-1 roots. Then, any $\mathcal{C}$-IPS$_{LIN}$ refutation of $f(\overline{x},\overline{z})$ requires:*

(1) $n^{\Omega(\log n)}$-size when $\mathcal{C}$ is the class of multilinear formulas;

(2) $2^{n^{\Omega(1)}}$-size when $\mathcal{C}$ is the class of constant-depth multilinear formulas; and

(3) $2^{\Omega(n)}$-size when $\mathcal{C}$ is the class of roABPs (in every variable order).

6. PIT AND PROOF COMPLEXITY

We already discussed the polynomial identity testing (PIT) problem in the context of both IPS and the non-commutative IPS. There, we were interested in the following question:

> *Can propositional proofs efficiently prove the correctness of a PIT algorithm for a given circuit class?*

We have seen in Section 3.4 that the PIT axioms capture the statements that express the correctness of a PIT algorithm (formally, a circuit for PIT). In other words, providing short Extended Frege proofs for the PIT axioms would de facto mean that Extended Frege efficiently proves the correctness of (some) polynomial-size circuits for PIT; from which it follows that Extended Frege polynomially simulates IPS. Subsequently, in Section 4.3, we showed that Frege *does* admit efficient (quasi-polynomial) proofs of the correctness of the Raz-Shpilka deterministic PIT algorithm for non-commutative formulas. This, in turn, implies that Frege quasi-polynomially simulates the non-commutative IPS (over $GF(2)$).

6.1. Proof Systems for Polynomial Identities

Hrubeš and Tzameret [Hrubeš and Tzameret 2009] asked the following question concerning the connection between proof complexity and the PIT problem:

Can we efficiently prove *polynomial identities? And specifically, is there a sequential proof system admitting polynomial-size proofs for all polynomial identities?*

In other words, this question asks whether the PIT problem admits short proofs, and specifically, whether there is a simple sequential proof system to witness that PIT has short proofs.

We know that an efficient *probabilistic* algorithm for PIT exists, due to Schwartz and Zippel [Schwartz 1980; Zippel 1979]: when the field is sufficiently large, with high probability, two different polynomials will differ on a randomly chosen field assignment. However, whether the PIT problem is in P, namely is solvable in *deterministic* polynomial-time, is of course a major open problem in complexity and derandomization theory. In fact, it is not even known whether there are sub-exponential-size *witnesses* that two algebraic formulas compute the same polynomial; namely, whether there is a *non-deterministic* sub-exponential-time algorithm for PIT, and in yet other words, whether PIT is in NP (note that PIT *is* known to be in $\mathsf{coRP} \subseteq \mathsf{coNP}$).

Hrubeš-Tzameret's work [Hrubeš and Tzameret 2009] investigated the PIT$\in ?\mathsf{NP}$ question from the proof-complexity perspective: assuming that PIT does posses short witnesses, is it the case that a proof system using only symbolic manipulation (resembling a Frege proof) is enough to provide these short witnesses? And if not, can we prove lower bounds on such proofs? Such lower bounds would at least delineate the methods that will work for efficiently proving polynomial identities and those that will not.

The work in [Hrubeš and Tzameret 2009], as well as the subsequent work [Hrubeš and Tzameret 2012], set out to define the analogs of Frege and Extended Frege for the PIT problem that we shall call *PI proofs* (for Polynomial Identity Proofs; originally these systems were called *arithmetic proofs*): just as Frege and Extended Frege prove boolean tautologies by deriving new tautological formulas (and circuits, resp.), PI proofs prove polynomial identities by deriving new identities between algebraic formulas (and circuits, resp.).

Let us first describe the analog of Frege for PIT, namely the PI proof operating with algebraic formulas, denoted simply $\mathbb{P}_f$ (where f stands for "formulas"). $\mathbb{P}_f$ is a sequential proof system whose axioms are the polynomial-ring axioms and whose derivation rules express the properties of the equality symbol. Each proof-line in the system is an equation between two algebraic formulas (or circuits; see below) F, G computing polynomials over a given field $\mathbb{F}$ written as $F = G$. The proof system is sound and complete for true polynomial identities:

THEOREM 6.1 ([HRUBEŠ AND TZAMERET 2009]). *Let $\mathbb{F}$ be a field. For any pair F, G of algebraic formulas, there is a PI proof in the system $\mathbb{P}_f$ of $F = G$ iff F and G compute the same polynomial.*

The specific description of the rules and axioms of the PI proof system $\mathbb{P}_f$ are quite natural. The inference rules of $\mathbb{P}_f$ are (with F, G, H formulas; where an equation below a line can be inferred from the one above the line):

$$\frac{F = G}{G = F} \qquad \frac{F = G \qquad G = H}{F = H} \qquad \frac{F_1 = G_1 \qquad F_2 = G_2}{F_1 \circ F_2 = G_1 \circ G_2} \text{ for } \circ \in \{+, \cdot\}\,.$$

And the axioms of $\mathbb{P}_c$ express reflexivity of equality ($F = F$), commutativity and associativity of addition and product ($F \circ G = G \circ F$, and $F \circ (G \circ H) = (F \circ G) \circ H$, for $\circ \in \{+, \cdot\}$), distributivity ($F \cdot (G + H) = F \cdot G + F \cdot H$), zero element ($F + 0 = F$, $F \cdot 0 = 0$), unit element ($F \cdot 1 = F$) and true identities in the field ($a \circ b = c$, for $\circ \in \{+, \cdot\}$).

A *PI proof* $\mathbb{P}_f$ is thus a sequence of equations $F_1 = G_1$, $F_2 = G_2, \ldots, F_k = G_k$, with F_i, G_i formulas, such that every equation is either an axiom or was obtained from previous equations by one of the inference rules. The *size* of a proof is the total size of all formulas

appearing in the proof. It is easy to see that, just like a Frege proof, a PI proof can be verified for correctness in polynomial-time (assuming the field has efficient representation; e.g., the field of rational numbers).

It is important to notice the distinction between PI proofs and propositional proofs: PI proofs prove polynomial identities (a language in coRP) while propositional proofs prove boolean tautologies (a language in coNP). See more on this in Section 6.1 below.

The analog of Extended Frege for PIT, denoted $\mathbb{P}_c$, is identical to $\mathbb{P}_f$, except that it operates with equations between algebraic *circuits* instead of algebraic formulas (similar to Jeřabek's Circuit Frege [Jeřábek 2004], formally, one needs to add another rule to such a system to be able to symbolically manipulate circuits, namely to merge two separate but identical sub-circuits into a single sub-circuit; see [Hrubeš and Tzameret 2012] for the details.)

It turns out that PI proofs are in fact very strong. First, [Hrubeš and Tzameret 2009] only demonstrated lower bounds on very restricted fragments of PI proofs, and apparently it is quite hard to go beyond these restricted fragments of PI proof systems (assuming any nontrivial lower bound even exists). Furthermore, PI proofs were found to admit short proofs for many non-trivial polynomial identities (like identities based on symmetric polynomials). Moreover, PI proofs are able to "simulate" PIT algorithms for restricted algebraic circuit classes; specifically, Dvir and Shpilka's PIT algorithm for restricted depth-3 algebraic circuits [Dvir and Shpilka 2006]. But more importantly, PI proofs were shown in [Hrubeš and Tzameret 2012] to efficiently simulate many of the classical structural results on algebraic circuits.

In particular, PI proofs $\mathbb{P}_c$ operating with equations between algebraic circuits, efficiently simulate the following constructions: (i) *homogenization* of algebraic circuits (implicit in [Strassen 1973]); (ii) Strassen's technique for *eliminating division gates* over large enough fields (also in [Strassen 1973]); (iii) eliminating division gates over small fields—this is done by simulating large fields in small ones; and (iv) *balancing algebraic circuits* (Valiant et al. [Valiant et al. 1983]; see also [Hyafil 1979]). Most notably, the latter result gives a strong *depth reduction* for polynomial-size $\mathbb{P}_c$ proofs to polynomial-size $O(\log^2 n)$-depth $\mathbb{P}_c$ proofs (for proving identities of a polynomial "syntactic degrees") and a quasi-polynomial simulation of $\mathbb{P}_c$ by $\mathbb{P}_f$. This is one important point where the PI proof systems differ from Frege and extended Frege, for which no non-trivial such simulation is known.

Since depth reduction is the most important of these results, let us state this more formally:

THEOREM 6.2 (DEPTH REDUCTION FOR PI PROOFS [HRUBEŠ AND TZAMERET 2012]). *Assume that F, G are circuits of (syntactic) degree $\leq d$ and depth $\leq t$. If $F = G$ has a $\mathbb{P}_c$ proof of size s then it has a $\mathbb{P}_c$ proof of size* $\mathsf{poly}(s, d)$ *and depth $O(t + \log s \cdot \log d + \log^2 d)$.*

Intuitively, one can think of this theorem as showing that VNC^2-PI proofs are equal in strength to VP-PI proofs, similar to the strong depth collapse manifested for algebraic circuits by Valiant et al. [Valiant et al. 1983] who showed that $\mathsf{VNC}^2 = \mathsf{VP}$ (where VNC^2 is defined similar to VP except that the depth of the circuits computing f_n is required to be $O(\log^2 n)$).

As we now discuss, this also had implications for understanding *propositional proofs*.

Algebraic Fragments of Propositional Proofs. Recall the Frege proof system described in Section 2.1. Each Frege proof-line is a propositional tautological formula, which is either a (substitution instance of an) axiom or was derived by the modus ponens rule. As mentioned before, Reckhow [Reckhow 1976] proved that it does not matter which derivation rules and axioms we use, nor even the specific logical connectives (gates) used: as long as

we use a finite number of rules and axioms[5] and the rules and axioms are (implicationally) complete, every two Frege systems are polynomially equivalent.

Now, consider the PI proof system $\mathbb{P}_f$, and assume the underlying field is $GF(2)$. In this case, *every PI proof-line becomes a boolean tautology*, where "+" becomes the logical gate XOR, "$\cdot$" becomes AND and "=" becomes the logical equivalence gate $\equiv$ (indeed, note that over $GF(2)$ the axioms become propositional tautologies). This then means that PI proofs over $GF(2)$ *are by themselves propositional Frege proofs*. The converse, on the other hand, is not true: not all Frege proofs are arithmetic proofs over $GF(2)$, because PI proofs over $GF(2)$ are *not* complete for the set of tautologies. For instance, $x_i^2 + x_i = 0$ is, over $GF(2)$ the boolean tautology $(x_i \wedge x_i) \oplus x_i \equiv \mathsf{false}$ over $GF(2)$, but it is not a true identity between (formal) polynomials and thus cannot be proved by a PI proof. In fact, Frege system is equivalent to the PI-system $\mathbb{P}_f$ over $GF(2)$ *augmented with the boolean axioms* $x_i^2 + x_i$; and similarly for Extended Frege and $\mathbb{P}_c$ over $GF(2)$.

Considering PI proofs as the "algebraic fragment" of propositional proofs gives us a new understanding of propositional proofs, and should hopefully shed more light on the complexity of Frege proofs. As mentioned above, it shows for instance a strong depth collapse, namely, that additional depth (beyond $O(\log^2 n)$) does not help to decrease the complexity of proofs. It is specifically useful for upper bounds questions on propositional proofs: if we can efficiently prove an algebraic identity over $GF(2)$ with a PI proof we can do the same for propositional proofs. This observation was used in [Hrubeš and Tzameret 2012] to give a polynomial-size and depth-$O(\log^2 n)$ Extended Frege proof of the determinant identities $Det(A) \cdot Det(B) = Det(AB)$ and other linear-algebraic statements such as the matrix inverse principle $AB = I_n \rightarrow BA = I_n$. By, essentially, unwinding depth-$O(\log^2 n)$ *circuits* into quasi-polynomial-size *formulas*, one can obtain quasi-polynomial-size Frege proofs of the same statements. These results give presumably tight upper bounds for the proof complexity of linear algebra, because, for example, Bonet, Buss and Pitassi conjectured that Frege does not admit polynomial-size proofs of these identities [Bonet et al. 1995].

7. CONCLUSION AND OPEN PROBLEMS

In this survey we demonstrated the emerging algebraic complexity approach to proof complexity. It is natural to expect that this close interaction between algebraic and proof complexity will continue to contribute new insights to the fundamental open problems in proof complexity. Already now very interesting and sometimes surprising new ideas came out from this interaction. In particular, we have seen that proof complexity lower bounds (for IPS restricted subsystems) are drawn almost directly from algebraic circuit lower bounds; new connections between computation and proofs, showing that some proof complexity lower bounds (IPS) imply computational lower bounds and complexity class separations ($\mathsf{VP} \neq \mathsf{VNP}$); and conversely, proving that certain lower bounds on weak computational models (non-commutative formulas) would imply strong Frege lower bounds; characterizing the "algebraic fragments" of Frege and Extended Frege systems and using structural properties of algebraic circuits yield a better understanding of the power of these systems through (apparently) tight short proofs for basic statements in linear algebra. All of these results have been achieved using methods from algebraic complexity.

The big challenge ahead is of course to find out whether the algebraic approach can eventually lead to lower bounds on Frege and Extended Frege, or conversely help to at least establish a formal (unconditional) so-called 'barrier' against proving such lower bounds (e.g., by showing that Extended Frege lower bounds imply strong explicit circuit lower bounds). But before this seemingly formidable challenge, there are many important intermediate

[5]The number of axioms and rules is finite, but they obviously induce infinite many substitution instances of axioms and rules, since the axioms and rules are closed under substitution of the variables in the axioms and rules by formulas

problems which seem relatively feasible at the moment, and whose solution will advance the frontiers of our understanding. We end by listing some of these problems.

— Can we obtain size lower bounds on constant-depth Frege with modular gates proofs ($AC^0[p]$-Frege proofs)? This problem has been open for decades, despite the known $AC^0[p]$-circuits lower bounds. It is quite conceivable that algebraic techniques may be of help on this.
— Can we establish size lower bounds on $\mathcal{C}$-IPS (linear or not) refutations for natural encodings of CNFs, for restricted circuit classes $\mathcal{C}$? The lower bounds from [Forbes et al. 2016b] hold only for a single hard axiom, and not CNFs.
— Can we extend the $\mathcal{C}$-IPS lower bounds to "dynamic" versions of $\mathcal{C}$-IPS? For instance, can we prove lower bounds on PC refutations operating with multilinear formulas or roABPs as in [Raz and Tzameret 2008b; Tzameret 2011]?
— Lower bounds on PI proofs of polynomial identities? Almost no lower bound is known for these "algebraic fragments" of Frege and Extended Frege.
— Just like PI proofs are proofs for the (algebraic) language of polynomial identities, it is very interesting to study the complexity of proof systems for other algebraic languages. Two examples of such proof systems are the proof system for matrix identities investigated in [Li and Tzameret 2013], and the proof system for non-commutative rational identities defined in [Garg et al. 2015]. Can we prove strong proof-size lower bounds on these systems? Can we connect these systems further to propositional proof complexity?

ACKNOWLEDGMENTS

We thank Stephen Cook, Kaveh Ghasemloo, Amir Shpilka and Avi Wigderson for very helpful discussions and clarifications, and Michael Forbes for very useful discussions and comments on a preliminary draft. We would also like to thank Neil Immerman for his careful reading and useful comments on this survey. Finally we are especially grateful to Joshua Grochow for many conversations and for answering our many questions that greatly improved this survey.

REFERENCES

Michael Alekhnovich and Alexander A. Razborov. 2001. Lower Bounds for Polynomial Calculus: Non-Binomial Case. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001)*. 190–199. DOI:http://dx.doi.org/10.1109/SFCS.2001.959893

Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. 1996. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. *Proc. London Math. Soc. (3)* 73, 1 (1996), 1–26. DOI:http://dx.doi.org/10.1112/plms/s3-73.1.1 Preliminary version in the *35th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1994)*.

Maria Luisa Bonet, Samuel R. Buss, and Toniann Pitassi. 1995. Are there hard examples for Frege systems? In *Feasible mathematics, II (Ithaca, NY, 1992)*. Progr. Comput. Sci. Appl. Logic, Vol. 13. Birkhäuser Boston, Boston, MA, 30–56.

Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. 2001. Linear Gaps between Degrees for the Polynomial Calculus Modulo Distinct Primes. *JCSS* 62, 2 (2001), 267–289. DOI:http://dx.doi.org/10.1006/jcss.2000.1726 Preliminary version in the *14th Annual IEEE Conference on Computational Complexity (CCC 1999)*.

Samuel R. Buss, Russell Impagliazzo, Jan Krajíček, Pavel Pudlák, Alexander A. Razborov, and Jiří Sgall. 1996. Proof Complexity in Algebraic Systems and Bounded Depth Frege Systems with Modular Counting. *Computational Complexity* 6, 3 (1996), 256–298. DOI:http://dx.doi.org/10.1007/BF01294258

Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. 1996. Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability. In *Proceedings of the*

28th Annual ACM Symposium on Theory of Computing (STOC 1996). 174–183. DOI:http://dx.doi.org/10.1145/237814.237860

Peter Clote and Evangelos Kranakis. 2002. *Boolean functions and computation models*. Springer-Verlag, Berlin. xiv+601 pages.

Stephen A. Cook. 1975. Feasibly Constructive Proofs and the Propositional Calculus (Preliminary Version). In *STOC*. 83–97.

Stephen A. Cook and Robert A. Reckhow. 1974a. On the Lengths of Proofs in the Propositional Calculus (Preliminary Version). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC 1974)*. 135–148. DOI:http://dx.doi.org/10.1145/800119.803893 For corrections see Cook-Reckhow [Cook and Reckhow 1974b].

Stephen A. Cook and Robert A. Reckhow. 1974b. Corrections for "On the Lengths of Proofs in the Propositional Calculus (Preliminary Version)". *SIGACT News* 6, 3 (July 1974), 15–22. DOI:http://dx.doi.org/10.1145/1008311.1008313

Stephen A. Cook and Robert A. Reckhow. 1979. The Relative Efficiency of Propositional Proof Systems. *J. Symb. Log.* 44, 1 (1979), 36–50. DOI:http://dx.doi.org/10.2307/2273702 This is a journal-version of Cook-Reckhow [Cook and Reckhow 1974a] and Reckhow [Reckhow 1976].

Zeev Dvir and Amir Shpilka. 2006. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. *SIAM J. on Computing* 36, 5 (2006), 1404–1434.

Michael A. Forbes, Mrinal Kumar, and Ramprasad Saptharishi. 2016a. Functional lower bounds for arithmetic circuits and boolean circuit complexity. In *31st Conference on Computational Complexity, (CCC)*. (2016).

Michael A. Forbes, Amir Shpilka, Iddo Tzameret, and Avi Wigderson. 2016b. Proof Complexity Lower Bounds from Algebraic Circuit Complexity. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*. 32:1–32:17. DOI:http://dx.doi.org/10.4230/LIPIcs.CCC.2016.32

Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. 2015. A deterministic polynomial time algorithm for non-commutative rational identity testing. *CoRR* abs/1511.03730 (2015). http://arxiv.org/abs/1511.03730

Dima Grigoriev. 1998. Tseitin's Tautologies and Lower Bounds for Nullstellensatz Proofs. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*. 648–652. DOI:http://dx.doi.org/10.1109/SFCS.1998.743515

Dima Grigoriev and Edward A. Hirsch. 2003. Algebraic proof systems over formulas. *Theoret. Comput. Sci.* 303, 1 (2003), 83–102. Logic and complexity in computer science (Créteil, 2001).

D. Grigoriev and Alexander A. Razborov. 2000. Exponential lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. *Appl. Algebra Engrg. Comm. Comput.* 10, 6 (2000), 465–487.

Joshua A. Grochow and Toniann Pitassi. 2014. Circuit Complexity, Proof Complexity, and Polynomial Identity Testing. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*. 110–119. DOI:http://dx.doi.org/10.1109/FOCS.2014.20 Full version at arXiv:abs/1404.3820.

Armin Haken. 1985. The intractability of resolution. *Theoret. Comput. Sci.* 39, 2-3 (1985), 297–308.

Pavel Hrubeš and Avi Wigderson. 2014. Non-commutative arithmetic circuits with division. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*. 49–66. DOI:http://dx.doi.org/10.1145/2554797.2554805

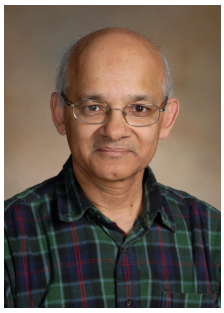
Pavel Hrubeš and Iddo Tzameret. 2009. The Proof Complexity of Polynomial Identities. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*. 41–51. DOI:http://dx.doi.org/10.1109/CCC.2009.9

Pavel Hrubeš and Iddo Tzameret. 2012. Short Proofs for the Determinant Identities. In *Proceedings of the 44th Annual ACM Symposium on the Theory of Computing (STOC)*. ACM, New York. DOI:http://dx.doi.org/10.1137/130917788
Laurent Hyafil. 1979. On the Parallel Evaluation of Multivariate Polynomials. *SIAM J. Comput.* 8, 2 (1979), 120–123.
Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. 1999. Lower Bounds for the Polynomial Calculus and the Gröbner Basis Algorithm. *Computational Complexity* 8, 2 (1999), 127–144. DOI:http://dx.doi.org/10.1007/s000370050024
Emil Jeřábek. 2004. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Logic* 129, 1-3 (2004), 1–37.
Jan Krajíček. 1995. *Bounded arithmetic, propositional logic, and complexity theory*. Encyclopedia of Mathematics and its Applications, Vol. 60. Cambridge University Press, Cambridge. xiv+343 pages. DOI:http://dx.doi.org/10.1017/CBO9780511529948
Jan Krajíček. 2011. *Forcing with random variables and proof complexity*. Cambridge University Press. 264 pages.
Fu Li and Iddo Tzameret. 2013. Generating matrix identities and proof complexity. *Electronic Colloquium on Computational Complexity, TR13-185* (2013). arXiv:1312.6242 [cs.CC] http://arxiv.org/abs/1312.6242.
Fu Li, Iddo Tzameret, and Zhengyu Wang. 2015. Non-Commutative Formulas and Frege Lower Bounds: a New Characterization of Propositional Proofs. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*. 412–432. DOI:http://dx.doi.org/10.4230/LIPIcs.CCC.2015.412 Full Version: http://arxiv.org/abs/1412.8746.
Noam Nisan. 1991. Lower Bounds for Non-Commutative Computation. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 1991)*. 410–418. DOI:http://dx.doi.org/10.1145/103418.103462
Jakob Nordström. 2015. On the Interplay Between Proof Complexity and SAT Solving. *ACM SIGLOG News* 2, 3 (Aug. 2015), 19–44. DOI:http://dx.doi.org/10.1145/2815493.2815497
Toniann Pitassi. 1997. Algebraic propositional proof systems. In *Descriptive complexity and finite models (Princeton, NJ, 1996)*. DIMACS Ser. Discrete Math. Theoret. Comput. Sci., Vol. 31. Amer. Math. Soc., Providence, RI, 215–244.
Ran Raz. 2013. Tensor-Rank and Lower Bounds for Arithmetic Formulas. *J. ACM* 60, 6 (2013), 40. DOI:http://dx.doi.org/10.1145/2535928
Ran Raz and Amir Shpilka. 2005a. Deterministic Polynomial Identity Testing in Non-commutative Models. *Comput. Complex.* 14, 1 (April 2005), 1–19. DOI:http://dx.doi.org/10.1007/s00037-005-0188-8 Preliminary version in the *19th Annual IEEE Conference on Computational Complexity (CCC 2004)*.
Ran Raz and Amir Shpilka. 2005b. Deterministic Polynomial Identity Testing in Non Commutative Models. *Computational Complexity* 14, 1 (2005), 1–19.
Ran Raz and Iddo Tzameret. 2008a. Resolution over linear equations and multilinear proofs. *Ann. Pure Appl. Logic* 155, 3 (2008), 194–224. DOI:http://dx.doi.org/10.1016/j.apal.2008.04.001
Ran Raz and Iddo Tzameret. 2008b. The Strength of Multilinear Proofs. *Computational Complexity* 17, 3 (2008), 407–457. DOI:http://dx.doi.org/10.1007/s00037-008-0246-0
Alexander A. Razborov. 1998. Lower Bounds for the Polynomial Calculus. *Computational Complexity* 7, 4 (1998), 291–324. DOI:http://dx.doi.org/10.1007/s000370050013
Alexander A. Razborov. 2015. Pseudorandom Generators Hard for k-DNF Resolution and Polynomial Calculus Resolution. *Annals of Mathematics* 181 (2015), 415–472.
Robert A. Reckhow. 1976. *On the lengths of proofs in the propositional calculus*. Ph.D. Dissertation. University of Toronto.

https://www.cs.toronto.edu/~sacook/homepage/reckhowthesis.pdf
J. T. Schwartz. 1980. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM* 27, 4 (Oct. 1980), 701–717. DOI:http://dx.doi.org/10.1145/322217.322225 Preliminary version in the *International Symposium on Symbolic and Algebraic Computation (EUROSAM 1979)*.
Amir Shpilka and Amir Yehudayoff. 2010. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science* 5, 3-4 (2010), 207–388. DOI:http://dx.doi.org/10.1561/0400000039
Volker Strassen. 1973. Vermeidung von Divisionen. *J. Reine Angew. Math.* 264 (1973), 182–202. (in German).
Iddo Tzameret. 2011. Algebraic proofs over noncommutative formulas. *Inf. Comput.* 209, 10 (2011), 1269–1292. DOI:http://dx.doi.org/10.1016/j.ic.2011.07.004
Leslie G. Valiant. 1979a. Completeness classes in algebra. In *Proceedings of the 11th Annual ACM Symposium on the Theory of Computing*. ACM, 249–261.
Leslie G. Valiant. 1979b. The Complexity of Computing the Permanent. *Theor. Comput. Sci.* 8 (1979), 189–201. DOI:http://dx.doi.org/10.1016/0304-3975(79)90044-6
Leslie G. Valiant. 1982. Reducibility by algebraic projections. *Logic and Algorithmic: International Symposium in honour of Ernst Specker* 30 (1982), 365–380.
Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. 1983. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM J. Comput.* 12, 4 (1983), 641–644.
Richard Zippel. 1979. Probabilistic Algorithms for Sparse Polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM 1979)*. Springer-Verlag, 216–226. DOI:http://dx.doi.org/10.1007/3-540-09519-573

SEMANTICS COLUMN

MICHAEL MISLOVE
Tulane University
Editor



PRAKASH PANANGADEN
McGill University
Guest Editor

In this quarter's column, Stephen Brookes and Peter O'Hearn describe how *Concurrent Separation Logic (CSL)* arose, including a little about the soundness proof and a discussion of later developments that arose from CSL. The column is especially timely, since Steve and Peter were just awarded this year's Gödel Prize for their work on CSL. The Gödel Prize was officially awarded for the papers [2] and [1] which describe CSL and its semantics. The Gödel Prize is testament to the importance of CSL as a fundamental tool for proving concurrent programs correct; the soundness proof forms a crucial component of this contribution.

It is impossible to talk about concurrent separation logic without mentioning separation logic. The name "separation logic" was coined by John Reynolds in his invited lecture at LICS'02 and the survey paper that followed [5]. There were many crucial inputs to the development of separation logic. From the point of view of logic, an important precursor was the logic of bunched implications developed by Peter and David Pym [3]. The crucial problem that separation logic addressed was reasoning about programs that manipulate pointer structures. The fact that the structure is irregular means that the simple ideas that worked in Hoare-style logic for programs using arrays fail to work here. Of course, it is not just a question of static structure. An essential problem is that data is being created and linked dynamically. As the Gödel Prize citation says: "For the last thirty years experts have regarded pointer manipulation as an unsolved challenge for program verification..."

In particular, concurrent programs that access data should avoid race conditions. How can one be sure of that without knowing whether chains of pointers end up reaching the same memory location? The sentence quoted above ends with "...and shared-memory concurrency as an even greater challenge." It is these problems *which are of central concern to practicing industrial programmers* that CSL solves.

In the 1970s and 80s there was a great deal of interest in supercomputers and parallel execution of large-scale scientific programs. Here, of course, one has to deal with similar problems of memory access conflicts. There had been quite a lot of success in dealing with programs that used regular control structures like loops traversing arrays, though much of the success was based on clever tricks involving the arithmetic of index expressions. There was no hope that these ideas would apply to heap-based structures being manipulated by recursive programs. In the late 1980s and early 1990s

there were some significant new ideas, like alias analysis for pointers and points-to analysis, which greatly extended the scope of what was possible with static analysis; but there was nothing approaching a general Hoare-style logic for the analysis of programs that manipulated pointers.

This was what the late John Reynolds and Peter O'Hearn set out to address with separation logic. Their ambition was nothing less than the ability to deal with the "dirtiest" code that one could find in the kernels of operating systems and other similar places, such as device drivers or garbage collectors. For those new to the field it is hard to imagine how impossible that seemed. Indeed it still seems miraculous to those of us who watched it happen!

The sequential version of separation logic reached maturity in the paper by O'Hearn, Reynolds and Yang [4]. Dealing with concurrency was no trivial extension. Entirely new ideas had to be brought to bear. The synergy between John Reynolds, Peter and Steve was remarkable here. The original proposals of O'Hearn turned out to be unsound, as was demonstrated by a subtle example due to Reynolds. Brookes and O'Hearn overcame this by using new assertions that precisely specified regions of memory. The evolution of these ideas is what the column they provide describes. Many others played an important role in CSL and separation logic especially Hongseok Yang and Cristiano Calcagno.

Steve's and Peter's work on CSL is one of the major milestones in the interaction between semantic theory and practice in programming languages. Programming language theory and logics of computation have the reputation of being far removed from the concerns of "real-world" programmers. The recent establishment of an SL-based verification group within Facebook, led by Peter, and the deployment of the Infer tool within Facebook shows separation logic to be a striking exception. Efforts to extend the tool to incorporate concurrency are underway. On the one hand CSL *does* use ideas from logic, category theory and denotational semantics, on the other hand it is couched in the paradigm of the conventional practicing programmer.

We are particularly thrilled to see this work recognized by the Gödel Prize. This year's prize is the 24th to be awarded and only the third to be awarded in the areas of logics, verification and programming languages. This ranks CSL as not just an outstanding contribution from two leaders of the logic and computation community, but as a contribution to computer science as a whole. For a new SIG it is a fantastic testament to the impact that logic and semantics can have on computer science as a whole.

On a personal level, it's a special privilege to introduce Steve's and Peter's column. In addition to being good friends and colleagues, they have been stalwarts of the annual conference series on the Mathematical Foundations of Programming Semantics, a conference series that we both help organize. While the major publications on separation logic have appeared at venues such as *POPL* and in journals such as *Theoretical Computer Science*, many of the early ideas were presented at MFPS. One of us (Panangaden) was (along with Steve) on Peter's PhD defence committee. It is a delight to see that young researcher now recognized as one of the leaders of the field; as we say in Canada, "Pretty good, eh?"

It's also fitting that this spring's MFPS meeting at CMU, with a special session to honor Steve on his upcoming 60th birthday,[1] provided an opportunity for the MFPS community to celebrate the awarding of this year's Gödel Prize to Steve and Peter for their pioneering work. We are particularly happy to share this with the SIGLOG community in this quarter's semantics column.

[1]The MFPS tradition of honoring colleagues on significant birthdays began in 1995 when Steve and Bob Tennent organized a session honoring John Reynolds on his 65th birthday.

REFERENCES

1. Brookes, S., A semantics for concurrent separation logic, Theoretical Computer Science **375** (2007), pp. 227270.
2. O'Hearn, P. W, Resources, concurrency, and local reasoning, Theoretical Computer Science **375** (2007), pp. 271307.
3. O'Hearn, P. and D. Pym, The logic of bunched implications, Bulletin of Symbolic Logic **5** (1999), pp. 215-244.
4. O'Hearn, P., J. Reynolds and H. Yang, Local reasoning about programs that alter data structures, in: *Proceedings* of CSL 2001, Lecture Notes in Computer Science **2142** (2001), pp. 1-19.
5. Reynolds, J., Separation Logic: A Logic for Shared Mutable Data Structures, in: *Proceedings* of LICS 2002, IEEE Press, pp. 55–74

Concurrent Separation Logic

Stephen Brookes
Carnegie Mellon University

Peter W. O'Hearn
University College London

Concurrent Separation Logic (CSL) was originally advanced in papers of the authors published in Theoretical Computer Science for John Reynolds's 70th Birthday Festschrift (2007). Preliminary versions appeared as invited papers in the CONCUR'04 conference proceedings. Foundational work leading to these papers began in 2002. Since then there have been significant developments stemming from CSL, both in theoretical and practical research. In this retrospective paper we describe the main ideas that underpin CSL, placing these ideas into historical context by summarizing the prevailing tendencies in concurrency verification and programming language semantics when the logic was being invented in 2002-2003. We end with a snapshot of the state-of-the-art as of 2016. Along the way we describe some of the main developments in the intervening period, and we attempt to classify the work that has been done, along broad lines. While we do not intend an exhaustive survey, we do hope to provide some general perspective on what has been achieved in the field, what remains to be done, and directions for future work.

1. CONTEXT, *CIRCA* 2002-2003

1.1. Verification of Concurrent Programs

> For the last thirty years experts have regarded pointer manipulation as an unsolved challenge for program verification and shared-memory concurrency as an even greater challenge. *2016 Godel Prize citation*[1]

As of 2002-2003, there had already been significant foundational work on the verification of concurrent programs, beginning with the classical work of Owicki and Gries [Owicki and Gries 1976a], who adapted ideas from Hoare's logic for sequential programs to a concurrent setting. Further important developments were made by Pnueli [Pnueli 1981], bringing temporal logic to concurrency, and by Jones [Jones 1983], who advanced the rely-guarantee proof method for compositional reasoning. There was also a large school (more accurately, separate schools) of work that had built up around process algebras such as CCS, CSP and the pi-calculus.

However, it is fair to say that at that time few realistic concurrent programs had been subjected to proof, and this was especially true for mechanized proof. Yet, around 2002-2003, the field of mechanized program verification was going through a renaissance, spurred by verification-oriented abstract interpreters such as SLAM and ASTREE, and advances in proof assistants such as Coq, Isabelle and HOL. The hitch was

[1] http://eatcs.org/index.php/component/content/article/1-news/2280-2016-godel-prize

that most of this work was for sequential programs. Model checking had been used to verify certain finite-state concurrent programs used to describe hardware, with notable success, but much less had been achieved by then for concurrent software.

Around the same time, the separation logic for sequential programs had recently been advanced by O'Hearn, Reynolds and others as a way to provide efficient reasoning about pointer manipulation [Reynolds 2000; Ishtiaq and O'Hearn 2001; O'Hearn et al. 2001; Reynolds 2002]. Separation logic was a significant advance over prior formalisms for reasoning about pointers (see [Bornat 2000] and its references for the state of the art as of 1999), providing inference rules to localize reasoning about heap chunks. Pointer mutation was also one of the key blockers in applying the foundational approaches provided by the Owicki-Gries, temporal logic and rely-guarantee methodology to real-world concurrent programs. Programs in C, Java and many current languages often make non-trivial use of pointer manipulation, to great effect, but reasoning about the correctness of such code is difficult even when the code is sequential, because of issues such as aliasing. Pressing these classic works into service for proving correctness of code using pointers requires treating the memory as a global array, and this leads to global proofs which are complicated and rather remote from programming intuition. Often, programmers think of portions of memory a little bit at a time, and employ localized reasoning that does not mention the whole memory; to force programmers to think in terms of global state flies in the face of such intuition. It therefore made sense to attempt to update one of these classical approaches to reasoning about concurrent programs by putting it together with separation logic, in which localized reasoning is very natural.

O'Hearn considered all three of these prior foundational works as possibilities for such an update, but gravitated to an even earlier work, Hoare's 1972 paper "Towards a Theory of Parallel Programming" (TTPP, [Hoare 1972]). Hoare's paper might be considered to have been superceded by its successor works, because there were many programs which could not be expressed in TTPP because of syntactic restrictions used to rule out race conditions and other forms of interference. But when it worked it allowed for delightfully simple proofs.

A useful perspective is brought by noting, and contrasting, the existence of not one but two Owicki-Gries logics. One [Owicki and Gries 1976b], which we will refer to as *race-free Owicki-Gries*, extends TTPP by slightly relaxing syntactic restrictions in a way that allows auxiliary variables to be used more freely in proofs, but still maintaining control over interference. The other [Owicki and Gries 1976a], which we will refer to as *interference-allowing Owicki-Gries*, allows interference between threads but uses a novel means of checking that a thread does not inferfere with a *proof* of another thread (note the concept of non-interference with proof, not with thread). The interference-allowing form of Owicki-Gries is often referred to (without qualification) as *the* "Owicki-Gries method", and it attracted more attention than race-free Owicki-Gries in work between the mid 1970s and 2002-2003, likely because it was applicable to more programs. But Owicki and Gries also state that "the proof process becomes much longer" in the less restrictive, interference-allowing Owicki-Gries system.

The design of Hoare's TTPP approach (and also race-free Owicki-Gries) was based almost completely, albeit implicitly, on intuitions of separation, so it just made sense to try to recast these intuitions explicitly by making use of the more recent formalism of separation logic.

1.2. Semantics of Concurrent Programs

At the same time that O'Hearn was considering ways to blend separation logic with Hoare-style concurrency logics, the state-of-the-art in programming language semantics of concurrency stood on foundations laid by David Park in the 1970s. Traditional

approaches, whether denotational or operational, relied on traces of some kind. In Park's seminal work [Park 1979] a trace is a sequence of steps, each step representing the effect of an atomic action performed on the global state by the program, assumed to be running in an environment of concurrent processes. To achieve a compositional account the trace set of a program includes traces with "gaps" allowing for the environment to make changes to the global state. Concurrent processes are assumed to be executed in a sequentially consistent manner, with the atomic actions of all processes being interleaved. In later work Brookes used traces in which the steps represent the effect of a finite sequence of atomic actions, a simple variation that led to a fully abstract semantics [Brookes 1996].

These semantic models were only applicable to shared-memory programs without data structures that could be mutated through pointer manipulation, a class of programs dubbed "simple" shared-memory by John Reynolds. For many years it had been widely regarded as a difficult task just to find a tractable denotational model for simple shared-memory, let alone develop a semantics that could deal with the combination of shared-memory concurrency and mutable data structures. These traditional semantic models were all based on global states and typically assumed that assignment to a variable was an atomic action, ignoring the potential for race conditions, even though it was known that when one process writes to a shared variable being read or written by another there is a danger of unpredictability. Instead one of the rôles of logics such as race-free Owicki-Gries [Owicki and Gries 1976b] and TTPP was to shift the burden of race-detection from semantics to logic: an assertion $\{P\}C\{Q\}$ approvable in race-free Owicki-Gries logic comes with the guarantee that when C is executed from a global state satisfying P there are no race conditions and upon termination the global state satisfies Q. This logic-based methodology only works for races involving program variables, which are statically detectable: they get dealt with in Owicki-Gries by imposing static constraints on variable usage in programs. This idea fails for programs using pointers, because aliasing of pointers is not statically determinable.

In summary, in 2002-2003 there was not yet any suitable semantic foundation for exploring the behaviour of concurrent programs that make non-trivial use of pointers. And the currently existing program logics that worked well enough for pointer-free concurrent programs relied on syntactic constraints to rule out race conditions, which prevents their use in richer settings where the existence of a potential race cannot merely be deduced from program syntax.

2. THE BIRTH OF CONCURRENT SEPARATION LOGIC

2.1. Logic

The most important proof rule of Concurrent Separation Logic is the

PARALLEL COMPOSITION RULE

$$\frac{\{P_1\}\, C_1\, \{Q_1\} \;\cdots\; \{P_n\}\, C_n\, \{Q_n\}}{\{P_1 * \cdots * P_n\}\, C_1 \parallel \cdots \parallel C_n\, \{Q_1 * \cdots * Q_n\}}$$

Here, the separating conjunction $P_1 * \cdots * P_n$ in the precondition of the parallel composition is true of a state that can be partitioned into substates making the conjuncts true, and similarly for the post-condition. So the rule says: To prove a parallel composition we give each process a separate piece of state, and separately combine the postconditions for each process.[2] The rule supports completely independent reasoning about

[2]In the earliest versions of CSL these rules were accompanied by side conditions governing variables (but not the heap), such as: no variable free in P_i or Q_i is changed in C_j when $j \neq i$. This was an historical pre-

processes. And, it is unsound if $*$ is replace by $\wedge$ because mutations in one process would invalidate the reasoning about common state in another process.

In this expository paper we will show "proof outlines" rather than logical proofs, and without even formally defining the logic in full; we hope that the spirit of the proof methodology can be gleaned from the assertions placed at program points.

An example use of the parallel composition proof rule is given by a proof of parallel mergesort. In the key step

$$\begin{array}{cc} \{\mathit{array}(a,i,m) * \mathit{array}(a,m+1,j)\} & \\ \{\mathit{array}(a,i,m)\} & \{\mathit{array}(a,m+1,j)\} \\ \mathtt{ms}(a,i,m) \quad \| & \mathtt{ms}(a,m+1,j) \\ \{\mathit{sorted}(a,i,m)\} & \{\mathit{sorted}(a,m+1,j)\} \\ \{\mathit{sorted}(a,i,m) * \mathit{sorted}(a,m+1,j)\} & \end{array}$$

we first reason independently about two independent recursive calls which operate on disjoint subarrays, and then reason about merging the sorted subarrays.

This method of reasoning about parallel mergesort is simple, straightforward, almost trivial. And that is the point. In the previous formalisms (interference-allowing Owicki-Gries, Rely-Guarantee, Temporal logic) parallel mergesort would require a complicated proof which had to explicitly describe, and then rule out, the possibility of interference. On the other hand, in TTPP and race-free Owicki-Gries parallel mergesort is not even syntactically well-formed, because assignments to components of an array in separate threads are viewed logically as assignments to the same variable; although these systems are intuitively about separation, their syntactic constraints are coarser than separation-expressed-with-logic. The CSL proof, in contrast, follows the informal reasoning directly. Although this simple reasoning pattern naturally has its limitations, it illustrates a principle which should be central in the subject of logics of programs: that is, *simple proofs for simple programs*. It is all too easy to get caught up in completeness and related issues for formal systems that turn out to be too complicated when humans try to apply them; it is more important first to get a sense for the extent to which simple reasoning is or is not supported.

Still, if CSL had only been able to reason about "disjoint concurrency", where there is no inter-process interaction, then it would have rightly been considered rather restrictive. A proof rule for conditional critical regions (a forerunner of monitors) allows reasoning about such interaction.

CRITICAL REGION RULE

$$\frac{\{(P * RI_r) \wedge B\}\, C\, \{Q * RI_r\}}{\{P\}\, \mathtt{with}\ r\ \mathtt{when}\ B\ \mathtt{do}\ C\ \{Q\}}$$

This rule assumes given an association of a "resource invariant" RI_r, to each "resource" r appearing in the program. A resource is like a monitor lock: it provides mutual exclusion for different occurrences of critical regions for r in a program. The rule supports modular reasoning by using the separating conjunction to control the visibility of the state described by the invariant, as well as the invariant itself.

An important early example done with CSL was the pointer-transferring buffer. In this example one thread allocates a pointer and puts it into a buffer, while the other thread reads it out and disposes (frees) it. The important thing about this code is that

---

sentational choice to do with keeping consistent with earlier Hoare logics; more recently, logics are given in which $*$ handles all noninterference ("variables as resource", [Parkinson et al. 2006]) or where local variable mutation is restricted to be thread-local.

not only is the pointer deemed to transfer from one process to another, but the "knowledge that it is allocated" or the "right to dereference it", or more simply "ownership" of the pointer, transfers with the proof. From the point of view of the buffer clients the proof of ownership transfer looks like this

$$
\begin{array}{ll}
\multicolumn{2}{c}{\{\texttt{emp} * \texttt{emp}\}} \\
\{\texttt{emp}\} & \{\texttt{emp}\} \\
x := \texttt{cons}(a,b); \;\; \| & \texttt{get}(y); \\
\{x \mapsto -,-\} & \{y \mapsto -,-\} \\
\texttt{put}(x); & \mathit{use}(y); \\
\{\texttt{emp}\} & \{y \mapsto -,-\} \\
 & \texttt{dispose}(y); \\
 & \{\texttt{emp}\} \\
\multicolumn{2}{c}{\{\texttt{emp} * \texttt{emp}\}} \\
\multicolumn{2}{c}{\{\texttt{emp}\}}
\end{array}
$$

where the buffer code itself is

$$
\begin{array}{lll}
\texttt{put}(x) & \triangleq & \texttt{with}\ \mathit{buf}\ \texttt{when}\ \neg\mathit{full}\ \texttt{do} \\
 & & \quad c := x;\ \mathit{full} := \texttt{true};
\end{array}
$$

$$
\begin{array}{lll}
\texttt{get}(y) & \triangleq & \texttt{with}\ \mathit{buf}\ \texttt{when}\ \mathit{full}\ \texttt{do} \\
 & & \quad y := c;\ \mathit{full} := \texttt{false};
\end{array}
$$

To reason about the buffer code using the critical region rule we supply a resource invariant

$$RI_{\mathit{buf}}: \quad (\mathit{full} \wedge c \mapsto -,-) \vee (\neg\mathit{full} \wedge \texttt{emp})$$

saying that the buffer owns the binary cons cell associated with c when *full* is true, and otherwise it owns no heap cells. The entire program begins with an initialization $\mathit{full} := \texttt{false}$ that establishes the invariant.

Then we can provide a proof outline for the body of the `with` command in $\texttt{put}(x)$.

$$
\begin{array}{l}
\{(RI_{\mathit{buf}} * x \mapsto -,-) \wedge \neg\mathit{full}\} \\
\{(\neg\mathit{full} \wedge \texttt{emp}) * x \mapsto -,-\} \\
\{x \mapsto -,-\} \\
\quad c := x; \mathit{full} := \texttt{true} \\
\{\mathit{full} \wedge c \mapsto -,-\} \\
\{RI_{\mathit{buf}}\} \\
\{RI_{\mathit{buf}} * \texttt{emp}\}
\end{array}
$$

The rule for `with` commands then gives us

$$\{x \mapsto -,-\}\texttt{put}(x)\{\texttt{emp}\}.$$

The postcondition indicates that the sending process gives up ownership of pointer x when it is placed into the buffer, even though the value of x is still held by the sender. A crucial point in the proof of the body is the penultimate step which passes from

$$\mathit{full} \wedge c \mapsto -,-$$

to RI_{buf}, reflecting the idea that the knowledge "x points to something" flows out of the user program and into the buffer resource. Some inverse manipulations give us the spec

$$\{\texttt{emp}\}\texttt{get}(y)\{y \mapsto -,-\}.$$

for getting an element out of the buffer, where the knowledge that y points to something materializes in the postcondition.

The ownership transfer idea illustrated by this buffer example made it clear that quite a few concurrent programs would have much simpler and more intuitive proofs than before, and that a wider class of programs would be susceptible to formal analysis. Modular proofs were provided of semaphore programs, of a toy memory manager, and programs with interacting resources. Generally speaking, ownership transfer allows modular reasoning about 'daring concurrency', where where a piece of state is accessed even outside of groupings of mutual exclusion. It seemed as if the logic could explain the way that synchronisation had been used in the fundamental works on concurrent programming by Dijkstra, Hoare and Brinch Hansen. For example, in the paper that essentially founded concurrent programming, Dijkstra (Co-operating Sequential Processes, [Dijkstra 1968]) had explained that the point of synchronisation was to enable programmers to avoid minute considerations of timing, in order to simplify reasoning. Brinch Hansen had emphasized the importance of speed independence and resource separation for simplifying thinking about concurrent processes when O'Hearn was his colleague at Syracuse in the 1990s, and what he said seemed to be mirrored in the proofs in this early concurrent separation logic.

In summary, the most important contributions of the logic are as follows.

— it provides a modular way of reasoning about concurrent programs by separating the state that different threads access,
— while achieving this in the presence of pointer access and mutation,
— and it supports modular reasoning about process interaction, even in the presence of 'daring concurrency' where common state is safely accessed by different processes outside of critical regions or other mutual exclusion constructs (as in a memory manager or the pointer-transferring buffer).

But, soundness was challenging.

2.2. Semantics

It was the very feature that gave rise to the unexpected power in the logic, the ownership or knowledge transfer, that made soundness non-obvious. O'Hearn was able to state principles intended to give some justification for the logic, such as the following.

> *Ownership Hypothesis*. A code fragment can access only those portions of state that it owns.
>
> *Separation Property*. At any time, the state can be partitioned into that owned by each process and each grouping of mutual exclusion.

But these principles were stated only informally. O'Hearn worked hard on the soundness problem for several months in the second half of 2001 and early in 2002, and got nowhere near a formal soundness theorem. During 2002 he asked Brookes, an expert in the semantics of concurrency, for help.

To establish soundness of CSL we needed a semantic model capable of dealing both with concurrency and pointer maipulation, suitable for formalizing and supporting accurate reasoning about ownership and separation. Although these notions seem intuitive, it was not at all obvious at the time how to design a suitably comprehensive denotational semantics, and how to make rigorous the idea behind ownership transfer and the notion that (provable) processes mind their own business. A key design choice was to take a more localized view of states, so that a global state could be seen as a combination of separate pieces of state deemed to belong to processes and resources.

While it may appear fairly straightforward to extend Park-style global traces to handle pointers and mutable state, this leads again to a semantics based on global states and it was unclear how to build in a more localized view of state. Instead we had the idea of abstracting away from state and using traces built from "actions" , initially uninterpreted and only characterized abstractly, which can later be interpreted as having an effect on state. In this two-stage approach, we would be free to instantiate the notion of state in whatever way we needed, either as global or as local. We would be able to analyze program behavior, independently of the logic, using global states; and to formalize ownership transfer and separation properties by taking a local view of traces. Another advantage, which emerged later, is that the *same* action trace semantics would turn out to be usable in validating a number of later program logics inspired by CSL, such as *logics of permissions*, merely by interpreting actions over an alternative notion of state, such as *permissive states*.

To deal with race-detection we introduced a form of interleaving operator for traces that detects race conditions and treats them as catastrophic. This is the right choice to obtain a semantics capable of establishing when a program is race-free. The new semantic model, which we refer to as *action traces*, is strongly influenced by Dijkstra's Principle [Dijkstra 1968]:

> . . . processes should be loosely connected; by this we mean that apart from the (rare) moments of explicit intercommunication, the individual processes are to be regarded as completely independent of each other.

In other words, concurrent processes do not interfere except through explicit synchronization. Action trace semantics reflects this idea through the interplay between traces, which describe interleaved behaviors of processes, and a "local enabling" relation that implements the "no interference from outside" notion. This interplay is crucial in permitting a formalization of O'Hearn's intuitive concept of "processes that mind their own business". To our knowledge, this was the first semantics in which such a formalization is possible.

The key novel features of this semantics:

— a compositional action-trace semantics built from "uninterpreted" actions, capable of incorporating both concurrency and pointers;
— a race-detecting interpretation of parallel composition;
— a global-state interpretation of actions and traces, consistent with a standard operational notion of execution;
— a local-state interpretation of actions and traces ("local enabling"), suitable for formalizing ownership transfer and the separation principle

Local enabling formalizes the notion of a process executing in an environment that respects resources (obeys the separation principle) and "minds its own business" by following the ownership transfer discipline embodied by the resource invariants. A key ingredient in the soundness proof is a Parallel Decomposition Lemma; in simplified form, this says that when $c_1 \| c_2$ is a race-free program, every interleaved computation of $c_1 \| c_2$ can be decomposed into "local" computations of the constituent processes c_1 and c_2 which are interference-free except for interactions with protected resources.

Our soundness proof assumes that each resource invariant is *precise*, so that every time a program acquires or releases a resource there is a uniquely determined portion of the heap whose ownership can be deemed to transfer. This does not seem to be a major limitation, since all of O'Hearn's examples involve precise invariants, and a methodology based on precision seems very natural [O'Hearn 2007]. Moreover this limitation is sufficient to ensure soundness, and it suffices to avoid a counterexample discovered by John Reynolds showing unsoundness when resource invariants are al-

lowed to be arbitrary separation logic formulas and the usual Hoare logic proof rule for conjoining postconditions is allowed. During the evolution of the semantic foundations John Reynolds played an important guiding rôle, and it is entirely appropriate that the two CSL papers bear dedication to him.

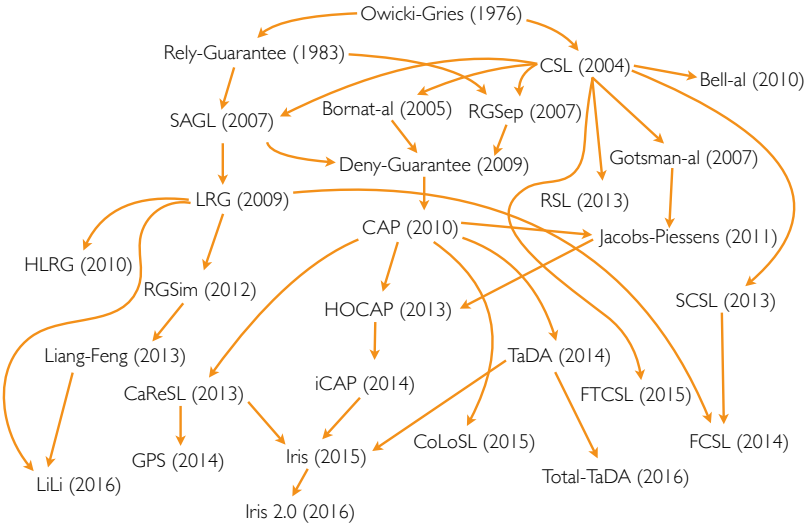


Fig. 1. CSL Family Tree (courtesy of Ilya Sergey)

3. DEVELOPMENTS IN THEORY

3.1. Logic

There have been many logics that extend or build on the ideas in CSL; Figure 1 gives an indication. The motivation for several of these logics has simply been to treat varied programming primitives that are used when writing concurrent programs. For example:

— *Storable locks.* The original CSL used statically allocated locks, whereas real programs often use dynamically-allocated locks that can themselves be stored [Gotsman et al. 2011];
— *Re-entrant locks.* CSL's critical sections cannot be nested, and similarly if you hold a semaphore and attempt to grab it again you will deadlock. On the other hand, Java's locks are such that a thread that holds a lock can acquire it again [Haack et al. 2008];
— *Fork/join.* CSL is formulated using structured parbegin/parend or parallel composition $\|$ to describe concurrent threads, which has been extended to fork/join concurrency constructs [Haack and Hurlin 2008; Dodds et al. 2009];
— *Message Passing.* [Villard et al. 2009; Bell et al. 2010; Lei et al. 2014];
— *Relaxed memory.* [Vafeiadis and Narayan 2013].

A very unexpected development has been the demonstration that the most basic principles of concurrent separation logic, particularly independent reasoning about threads using the separating conjunction, cover a much broader range of situations

than we ever expected. There have been proofs of fine-grained locking and non-blocking concurrency, and cases that involve interference and general graph structures, what might have been thought of originally as cases that don't fit well with the ideas of separation logic.

Interestingly, the unexpected power of this is based on what you might call "non-standard models" of separation logic; we mean this by analogy with the usual situation in logic, where a theory (e.g. reals, or integers) has an intended model, but also additional non-standard models of the same axioms. The proof theory may then accomplish unexpected things when applied to the non-standard models. The standard model of separation logic is the original model based on splitting portions of the heap, or heaplets. There are many other models stemming from the "resource semantics" of bunched logic invented by David Pym [Pym et al. 2004]; the preconditions and post-conditions in separation logic and an instance of bunched logic [Ishtiaq and O'Hearn 2001]. Indeed, "abstract separation logic" [Calcagno et al. 2007a] defines a version of CSL starting from an arbitrary partial commutative monoid of program states, in place of the standard partial monoid of heaplets and disjoint union. A partial commutative monoid $(M, \bullet, e)$ induces an ordered total commutative monoid on the powerset $\mathcal{P}(M)$, and this is used as the semantics of $P * Q$. In fact, even more generally we could go beyond powerset models and define an abstract separation logic where a model is an ordered total commutative monoid; this gives enough structure to formulate the proof rule for parallel composition and the rule of consequence. More structure can be added (such as meets and joins) to interpret other logical connectives, depending on what is needed in an application.

The surprise is that some of these nonstandard models involve composing highly intertwined structures and interfering processes, what might have been considered bad cases for the ideas behind separation logic. Gardner coined the phrase "fiction of separation" to describe this phenomenon in the nonstandard models. Here are some representative works.

— *Permissions.* These are models where heap locations have additional information attached, which sometimes affects heap composition. An insightful paper of Boyland [Boyland 2003] influenced CSL work where fractional and counting permission models were given to account for concurrent reads and for the classic readers-and-writers problem[Bornat et al. 2005], and since then a wide variety of permission models have been invented and applied.

— *Abstraction and Fictions of Separation.* The logic of Concurrent Abstract Predicates (CAP) of [Dinsdale-Young et al. 2010a] provides a powerful means of disjoint reasoning about processes accessing an abstract module, when the concrete implementations are in fact not disjoint. This aspect, which builds on observations about the "fiction of disjointness" from work in sequential separation logic [Dinsdale-Young et al. 2010b], significantly expands the possibilities for applying CSL-style reasoning to fine-grained concurrent algorithms. CAP provides means of connecting an abstract module to a concrete implementation in a way that allows more apparent separation on the abstract level. Extensions of CAP have been used to do impressive verifications of libraries of synchronization primitives (e.g., [Svendsen et al. 2013; Dodds et al. 2016]). CAP has influenced many follow-on logics, as can be seen in Figure 1.

— *Views.* Dinsdale-Young, Parkinson and colleagues show that a simple abstract version of concurrent separation logic can embed many other techniques for reasoning about concurrency including type systems and even the classic rely-guarantee method, which was invented for the purpose of reasoning about interference [Dinsdale-Young et al. 2013]. Earlier work had sought combinations of separation logic and rely-guarantee [Feng et al. 2007; Vafeiadis and Parkinson 2007], but this work and [Dodds

et al. 2009] demonstrated that by picking a suitable monoid that expresses interference rely-guarantee could be expressed with a separation logic, constituting a valuable synthesis. Views also provides logical or fictional separation. It is a framework that can be instantiated to many logics, and so does this in a considerably more general way than CAP, while partly taking inspiration from it.
The logic of Views is similar to that of Abstract Separation Logic, but the semantics is different, and this is what opened up the possibility of a broader variety of instantiations of the framework: we say more on this in the next section.

— *Modular reasoning about history.* It is very natural to use temporal reasoning when reasoning about concurrent algorithms. For example, there are cases when wants to talk about the value of a piece of state sometime in the past, or between two timepoints. [Fu et al. 2010] and [Sergey et al. 2015] define logics that support modular ways for reasoning about both space and time, based on ways to compose histories, and they illustrate the power of this form of reasoning by proving optimistic and fine-grained locking concurrent algorithms. The previously-mentioned logics for message passing [Villard et al. 2009; Bell et al. 2010; Lei et al. 2014] also include support for reasoning about history.

Before moving on it will be instructive to consider an example of a non-standard model from the Views work [Dinsdale-Young et al. 2013]. Assume a partial commutative monoid $(M, \bullet, e)$ (e.g, the monoid where $M = L \rightharpoonup_f V$ is heaps and $\bullet$ is union of disjoint heaps). An *interference relation* is a subset $R \subseteq M \times M$ satisfying certain conditions. Then we can define a total commutative monoid $(Stab, *, \{e\})$ where the set $Stab = \{p \subseteq M \mid Rp \subseteq p\}$ consists of those sets (predicates) that are stable under the interference relation. $*$ is a lifting of $\bullet$, where $p * q = \{h_p \bullet h_q \mid h_p \in P \text{ and } h_q \in q \text{ and } h_p \bullet h_q \downarrow\}$. The (unstated, here) conditions on R ensure that $\{e\}$ and $p * q$ are stable.

The notion of stability under interference is one of the hallmarks of rely-guarantee reasoning. Models based on monoids like $(Stab, *, \{e\})$ provide one of the ways that separation logics can be used to reason modularly about interfering processes.

Even with the tremendous progress that has occurred, techniques for modular reasoning in the presence of fine-grained interference continue to evolve, with several substantial works appearing just in the past year or so; e.g. [Raad et al. 2015; Jung et al. 2015; da Rocha Pinto et al. 2016; Liang and Feng 2016].

3.2. Semantics

The action traces semantics of Brookes was denotational by choice: a denotational semantic description lends itself naturally to compositional reasoning, and this can be an advantage because most of the inference rules of CSL are syntax-directed, like the semantic clauses. (The exceptions are rules such as the Frame Rule, and the Conjunction Rule, in which the premiss and conclusion involve the same program and such rules typically express some general semantic or behavioral property applicable to all programs.)

Nevertheless, much of the notation used by Brookes in the soundness paper was deliberately chosen to emphasize intuitive similarities with operational style. For example, the notation for local enabling and global enabling resembles the labelled transition relations common in operational semantics. These remarks reflect the view that denotational and operational semantics should be seen as complementary styles of language description, not mutually exclusive.

As we have seen, CSL led to a proliferation of successor logics. Some of these can claim common philosophical links with the original logic, in that they are based on

analogous to the Separation Principle and Ownership Transfer, while departing in ways from the original, while others use proofs that do not depend on these principles.

— [Brookes 2006]: Brookes extends his semantics to cover permission-based variants of CSL, such as fractional and counting permissions [Bornat et al. 2005] and variables as resource [Parkinson et al. 2006].
— [Calcagno et al. 2007a]: the soundness result for Abstract Separation Logic is given by a generalization of the approach in Brookes's semantics, from the particular model of heaps to work for arbitrary (cancellative) partial commutative monoids, assuming that primitive operations satisfy a locality conditions ensuring that they "mind their own business".
— [Hayman and Winskel 2008]: a "true concurrency" denotational semantics based on Petri nets. The intuitions expressed in the Separation Property and in Dijkstra's principle of loose connectedness are givien formal underpinnings using the concept of independence from Petri nets. This explains the CSL phenomenon "the order of certain interleavings doesn't matter for provable programs" in terms of a well established, prior notion of independence.
— [Hobor et al. 2008]: a Concurrent Separation Logic with first-class locks and threads, with a soundness proof based on an operational semantics. The work is notable for introducing two operational semantics, one a standard concurrent semantics and the other an "oracular" semantics which is closer to the intuitions of CSL; the logic is proven sound wrt the oracular semantics, and then the semantics is separately connected to the standard concurrent model. This division mirrors Brookes's use of global and local enabling relations, but is done employing operational rather than denotational semantics.
— [Vafeiadis 2011]: a soundness result formulated in an inductive manner that matches the stepwise "small-step" operational style of semantics. In contrast to the aforementioned works, Vafeiadis does not employ an additional semantics (the oracle semantics, or local enabling relation) which is more logical than a standard semantics; he connects the logic directly to the semantics in one step, with a novel interpretation of Hoare triples. This would be no win if the connection was more complex than the composition of two semantics, but Vafeiadis's proof is notable as well for being mathematically very elementary.
  Vafeiadis is also able to show soundness of CSL without precise invariants and without the conjunction rule, as well as of the original CSL.
— [Gotsman et al. 2011]: another operational approach, but where Vafeiadis uses a structural operational semantics à la Plotkin, which involves program rewriting, Gotsman et al use a fixed-program semantics that updates a program counter without rewriting the program; this is the kind of semantics often used in the temporal logic and model checking communities. With this form of semantics they are able to give a very direct expression of the Separation Property, and also a mathematically elementary proof. And, again like Vafeiadis, they show soundness for versions of separation logic with and without precise resource invariants.

There are extensions of relatives of CSL that are not conceptually based so directly on the Ownership+Separation ideas, and then it is more difficult to employ a proof technique that directly extends Brookes's original approach. In particular, CAP and its descendants and other related logics use the rule of parallel composition to reason about interfering processes. A very general account of this kind of situation is provided by the Views work [Dinsdale-Young et al. 2013]. As we mentioned above, the logic of Views is a close relative of Abstract Separation Logic, but its semantics is entirely different.

— Views considers a model of a separation logic assertion as an *abstract interpretation [Cousot and Cousot 1977]* of a concrete semantics, not the concrete semantics itself. There is a concretization function $\lfloor \cdot \rfloor$ taking an abstract state p to a set of concrete states $\lfloor p \rfloor$. Furthermore, $*$ is required to exist in the abstract semantics, but not in the concrete.
— A non-standard interpretation of Hoare triples is given that "bakes in" the frame rule of separation logic for every step of the execution of C: $\{p\}C\{q\}$ is true of views (abstract states) p and q iff for each trace $\overline{a}$ of C, there exists an $\overline{s}$ such that the triple $\{\lfloor s_i * r \rfloor\} a_i \{\lfloor s_{i+1} * r \rfloor\}$ is true in the usual interpretation of triples in the concrete semantics for all r, and $\overline{s}_0 = p$, and if $\overline{a}$ is finite then its last state is q.
— The soundness property is stated with respect to a standard concrete operational semantics, without employing an additional, more abstract operational or denotational semantics.

Considering separation logic states as abstract states, or views of concrete states, makes great sense. Even in the original heap splitting model, a finite "heaplet" $h : L \rightharpoonup_f V$ would rightly be considered as a portion of many actual machine states, not a machine state on its own; it was convenient and simple at the time not to formalize the fact that the heaplets were abstractions of many states; we can easily draw these portions of heap, without talking about all the enclosing global heaps. But as the states in the models become more intricate, the punning of separation logic models as "like" real states starts to break down. In a fractional permission model a heaplet $[10 \mapsto 47, 1/2]$ says that location 10 holds value 47 and has "permission 1/2". The operational sensibility of permission models is much easier to grasp when such a heaplet is thought of as an abstract state rather than a concrete one on which programs execute. And beyond making intuitive sense, the Views semantics provides a validation of the rules of a version of CSL for models which allow for interference. The operative concept for Views is that of context: every step of the program must preserve all possible contexts (expressible by framing via $*r$). The Ownership Hypothesis and Separation Property together give a simple way to obtain contexts that are preserved, but Views both formalizes and exploits a more general setup in which there are others.

We remark that the proof technique of [Vafeiadis 2011], discussed above, also takes the tack of baking in the frame rule. It is similarly applicable to CAP and other logics where the Separation Property and Ownership Hypothesis would not apply. Thus, we see that the conceptual foundation of CSL-like logics has broadened over time, and with that logics that exploit the broadening have appeared.

Another trend in work on separation logic has been towards taking an axiomatic perspective, where what counts as a model of $*$ is subject to axioms rather than fixing on a single model to work with. This trend can be seen in Abstract Separation Logic and Views, as well as in other advanced logics such as Iris and HoCAP where the programmer (or human verifier) gets to "pick a monoid". The axiomatic perspective has been taken further still in recent work on Concurrent Kleene Algebra [Hoare et al. 2011; Hoare et al. 2014]: they abstract not only from the semantics of logical assertions, but even from the semantics of programs. In brief:

— A CKA is an ordered monoid (;, `skip`) and an ordered commutative monoid ($\|$, `skip`), linked by the exchange law

$$(p \parallel r); (q \parallel s) \sqsubseteq (p; q) \parallel (r; s).$$

(Variants on CKA can have more structure, such as involving meets or joins.)
— A concurrent separation logic can be derived from a CKA by making the definition $\{p\}c\{q\} \;=\; p; a \sqsubseteq q$.

— An operational semantics related to Milner's for CCS can be derived by making the definition $p \xrightarrow{a} q \;=\; p \sqsupseteq c; q$.
— The operational semantics and logic are automatically in unison; we don't need to prove a theorem connecting them.
— A number of concrete models (instances of the algebra) have been given, particularly ones based on a true concurrency view of the world where ; is weak sequential composition and $\|$ is a form of composition of pomsets.

Concurrent Kleene Algebra is more abstract than most other work on CSL, in that it does not depend on a particular semantics of programs. While abstract in this sense, the operational semantics that it derives has not been shown to cover all the cases yet that (say) the Views theory covers; see [O'Hearn et al. 2015] for further discussion. So while CKA is more general that Views or Abstract Separation Logic on some dimensions, it is not (known to be) on others. However, it is a beautifully simple theory which is remarkable for the unification of logic and operational semantics it achieves when it is applicable, and this can be taken as an inspiration or a starting point for further work.

4. DEVELOPMENTS IN MECHANIZED VERIFICATION

Mostly-Automatic Verification. Smallfoot [Berdine et al. 2005], the first separation logic verification tool, included support for CSL from the beginning. The user would input resource invariants and other annotations such as procedure pre/post specs, and then Smallfoot would attempt to construct a program proof. For the pointer-transferring buffer, given the resource invariant and pre/post specs for the `put` and `get` operations it can verify pointer safety and race freedom of the client code. Smallfoot used a decidable fragment of separation logic oriented to linked lists and simple trees, and implemented a special theorem prover for this fragment.

Smallfoot is an example of a verifier where the programmer helps the tool along by inserting (some) annotations, but then the verifier behaves automatically. There are a number of other tools for mostly-automatic verification of this kind, which extend or build on the reasoning in CSL.

— *SmallfootRG* [Calcagno et al. 2007b] is a verifier for a marriage of separation logic and the classic rely/guarantee method for concurrent programs [Vafeiadis and Parkinson 2007].
— *Heap Hop* [Villard et al. 2010] is another extension of Smallfoot. It implements the copyless message passing logic of [Villard et al. 2009] and checks their protocols using certain automata.
— *Chalice* [Müller and Summers 2016] uses an expressive permission system and also targets reasoning about deadlock. It is based on a variant or relative of separation called "implicit dynamic frames" [Parkinson and Summers 2012], and leverages an embedding into first order logic, then utilizing an SMT solver rather than a custom theorem prover like that used by Smallfoot.
— *VeriFast* [Jacobs et al. 2016] is an advanced mostly-automatic verifier. Where the previously-mentioned tools work for toy, illustrative programming langauges, VeriFast applies to C and Java programs. It has been used to produce proofs for programs ranging from object-oriented patterns to highly concurrent algorithms to systems programs. Examples of fine-grained concurrent programs proven include hand-over-hand locking on linked lists and lock-free queues.
— *Mezzo* [Balabonski et al. 2014] and *Asynchronous Liquid Separation Types* [Kloos et al. 2015] use novel type systems which incorporate ideas from CSL into a programming language. The mode of usage is similar to the other mostly-automatic verifiers

above, but the aim is to be less expressive (and less of a burden) and to weave the specifications in with the types of a programming language. Related ideas can be found earlier in Cyclone, and more recently in the ownership typing that happens in the Rust language. It seems as if there is a lot of room for experimentation and innovation in this space.

Interactive Verification. In interactive verification the human helps with the proof effort, commonly in a proof assistant such as Coq, HOL or Isabelle. With interactive verification it is possible to get closer to proofs of full functional correctness than it is with automatic verifiers, but the cost is higher. There are a number of embeddings of CSL and relatives in the Coq proof assistant, and more complex programs have been proven over time.

— [Feng et al. 2008] treat *interrupts and preemptive threads* as found in OS kernels. They use an ownership-transfer semantics where portions of state are transferred between interrupt handlers and threads. Pre-emptive thread libraries for locks, conditions variables and context switching, comprising some 300 lines of x86 assembly code, were machine certified; the proof took 82k lines of Coq, including more than 1k definitions and 1.8k lemmas and theorems
— [Sergey et al. 2015] report on the verification using the aforementioned Fine-grained CSL. They provide verifications of a variety of low-level algorithms including a CAS-lock, a Ticketed lock, a GC allocator, a non-blocking stack, and a concurrent spanning tree construction. Proofs range from hundreds to 2k lines of Coq. An emphasis is placed on reusability; for instance, the stack uses the GC allocator, which in turn uses a lock, but the stack uses the spec of the allocator and the allocator uses the spec rather than the implementation of a lock.
— [Xu et al. 2016] verify key modules of a commercial preemptive OS kernel, the μC/OS-II kernel. The authors report that the ownership transfer idea of CSL plays a key role in the specifications and proofs. Modules verified include the scheduler, interrupt handlers, message queues, and mutex locks. 1.k lines of C code is verified using 216k lines of Coq, including framework code. It took 4 person years to develop the framework, 1 person year to prove the first module (for message queues), and then the remaining modules, consisting around 900 lines of C code, were done in 6 person-months.
  This is apparently the first commercial *pre-emptive* kernel to have been verified (a number of high profile non-premptive kernel verification efforts, such as the celebrated seL4 project, have gone before).

Automatic Program Analysis.. With a verification-oriented abstract interpreter the program annotations that a human would supply to a mostly-automatic verifier – such as loop invariants, and sometimes pre/post specs – are inferred using a variety of techniques such as fixed-point iteration, widening and narrowing, interpolation, and abduction. Naturally, a verification tool will be able to prove less when the human is not supplying annotations, but the corresponding gain is that the techniques can be wholly automatic, working on bare code without asking the human for help to get started. Thus, verification-oriented abstract interpreters can be deployed with less friction than can mostly-automatic verifiers.

Abstract interpretation with sequential separation logic has seen rather a lot of attention; see [Calcagno et al. 2011] and its references. There have been comparatively fewer works on abstract interpretation with CSL (admittedly, a very difficult problem).

— [Gotsman et al. 2007] were the first to show how to infer resource invariants in CSL. They use reachability in the abstract heap to decide how to divide the state between

shared and thread-local after each critical region. Then, the invariants are discovered by convergence of a fixed point in the usual way of abstract interpretation.

— [Calcagno et al. 2009] also infers resource invariants, but differs from the work of Gotsman et al by using the concept of footprint rather than reachability to divide the state after critical regions. This allows them to prove some of the ownership transfer example, such as O'Hearn's pointer-transferring buffer, that cannot be proven with the technique of Gotsman et al. Footprints are approximated using the abductive inference technique of [Calcagno et al. 2011].

— [Botincan et al. 2012] describe an algorithm that takes as input a sequential program with a proof in separation logic and some additional annottions, and outputs a concurrent program with a proof in CSL. Abduction is used to perform the decomposiitons needed to do the CSL proof.

These works make good steps, but program analysis with CSL is underdeveloped compared to the work in mostly-automatic and interactive verification. This is an imbalance that is deserving of further attention in the research community, especially since the possibility for broad impact is, except possibly in the very long term, much greater with verification-oriented abstract interpreters than less automatic tools.

5. CONTEXT, *CIRCA* 2016

At the beginning of the paper we mentioned that, as of 2002-2003:

— few realistic concurrent programs had been subjected to proof, and

— the semantics of concurrency for "simple" shared memory programs was well developed, but we were lacking tractable models for shared memory programs with structured state accessed via pointer manipulation.

Expanding on the latter, we needed to be able to explain the connection between such concepts as race freedom, the Separation Property and Ownership Hypothesis, and the sense of independence in Dijkstra's principle of loosly connected processes. It was easy to define *some* model for a concurrent language, but not one that let us delve into these sorts of properties; and they were all intimately related to the initial Concurrent Separation Logic.

Fast-forward to 2016 and, on the verification side:

— many small but realistic and increasingly intricate concurrent programs have been verified interactively and semi-automatically, and there are tens of implemented tools to do so, and

— a significant portion of an industrial, pre-emptive OS kernel has been verified.

These advances are beyond what we would dared to imagine in 2002-2003. Shortly after CSL was introduced, we were given the friendly challenge by Doug Lea (author of java.util.concurrent, a library full of advanced tricks) to verify optimistic, non-blocking algorithms. These algorithms seemed beyond the intuitive reach of the initial CSL techniques, and we wrestled with them for quite some time (see, e.g., [Parkinson et al. 2007] for early struggles). But with perseverence and insight the field as a whole has moved to the point where mechanized verification of another of these algorithms now comes as no surprise. Concerning OS verification, as soon as CSL was available Berdine and O'Hearn became keenly interested in the problem of attacking a microkernel; microkernel designs seemed to fit CSL conceptually, but verification technology for concurrent programs was not sufficiently developed, at least as far as we knew, to make such an effort feasible as of 2002-2003.

While the distance we have come in verification is very positive, to keep some perspective on what has been achieved it is important to stress that the tools referred to

here are from the research sector. We have not yet seen concurrent software routinely verified within industry, by industry and not by researchers from either academia or industry; that is, verification of concurrent software is not yet *deployed*.

On the semantics side, there has been a flurry of development. We have models supporting increasingly simple and powerful proof methods. Soundness has been proven for concurrency logics not only with higher-order predicates but even with higher-order store obtained from storing resource invariants. The semantic models incorporate separation and interference at the same time. And there are general frameworks like in the Views work which give general conditions that allow to formulate concurrent separation logics for a given situation.

So there has been significant progress in logic and semantics. But there is much more work to do.

One important direction in pure theory is unification. The diagram in Figure 1 showing the CSL family tree shows a lot of variation, and it would be valuable to bring to a form of conclusion all of the developments on non-standard models and variations of CSL. The Views and Concurrent Kleene Algebra work both provide interesting starting points. There are also a lot of related ideas swirling around in models for weak memory and for distributed databases that have not been fully worked out. It seems that there is good and possibly deep theoretical work to be done.

As we have seen there has been a lot of progress in mechanized verification of concurrent programs; but there has been less in automatic program analysis. With program analysis we would like to give programmers feedback without requiring annotations, say by trying to prove specific integrity properties (such as memory safety or race freedom); annotations can help the analysis along, but are not needed to get started, and this greatly eases broad deployment. A number of prototype concurrency analyses based on CSL have been developed, as we mentioned in the previous section, but there has been much more work applying sequential separation logic to program analysis. For example, the Infer program analyser [Calcagno et al. 2015], which is in production at Facebook, is applied automatically on a daily basis to hundreds and sometimes thousands of modifications made to the Facebook code bases; it uses sequential but not concurrent separation logic. To make advanced program analysis for concurrency which brings value to programmers in the real world is in the main an open problem, and not an easy one.

Finally, we would like to say that CSL is but one part of work that goes on in the broader field of concurrency verification and analysis. There is much interesting work happening in dynamic concurrency analysis (e.g., the T-SAN tool), in symbolic model checking for concurrent software (e.g. CBMC), in modelling weak memory (e.g., the Herd tool), in logics and model checkers for models of distributed systems (e.g., TLA), and on more closely related logics and tools (e.g., VCC). In this retrospective paper we have not discussed or even referenced work other than that related to CSL, because to do justice to the other work would take too much more time and space. But, if we were to look at the state of the broader field in 2016 versus 2002 we would similarly see quite a significant level of advance, particularly in tools. While that is certainly positive and a cause for optimism, making concurrency theory+logic+tools simple and tractable enough to be deployed is still a great challenge, and a most worthwhile one because concurrent programs remain amongst the most difficult to understand.

ACKNOWLEDGEMENTS. Thanks to Mike Mislove and Prakash Panangaden for inviting us to write this article, and to Philippa Gardner, Matthew Parkinson and Ilya Sergey for discussion on some of the recent developments.

REFERENCES

Thibaut Balabonski, François Pottier, and Jonathan Protzenko. 2014. Type Soundness and Race Freedom for Mezzo. In *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*. 253–269. DOI:http://dx.doi.org/10.1007/978-3-319-07151-0_16

Christian J. Bell, Andrew W. Appel, and David Walker. 2010. Concurrent Separation Logic for Pipelined Parallelization. In *SAS (LNCS)*, Radhia Cousot and Matthieu Martel (Eds.), Vol. 6337. Springer, 151–166.

Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2005. Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In *FMCO (LNCS)*, Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever (Eds.), Vol. 4111. Springer, 115–137.

Richard Bornat. 2000. Proving Pointer Programs in Hoare Logic. In *MPC (LNCS)*, Roland Carl Backhouse and José Nuno Oliveira (Eds.), Vol. 1837. Springer, 102–126.

Richard Bornat, Cristiano Calcagno, Peter W. O'Hearn, and Matthew J. Parkinson. 2005. Permission accounting in separation logic. In *POPL*, Jens Palsberg and Martín Abadi (Eds.). ACM, 259–270.

Matko Botincan, Mike Dodds, and Suresh Jagannathan. 2012. Resource-sensitive synchronization inference by abduction. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*. 309–322. DOI:http://dx.doi.org/10.1145/2103656.2103694

John Boyland. 2003. Checking Interference with Fractional Permissions. In *SAS (LNCS)*, Radhia Cousot (Ed.), Vol. 2694. Springer, 55–72.

Stephen D. Brookes. 1996. Full Abstraction for a Shared-Variable Parallel Language. *Inf. Comput.* 127, 2 (1996), 145–163. DOI:http://dx.doi.org/10.1006/inco.1996.0056

Stephen D. Brookes. 2006. Variables as Resource for Shared-Memory Programs: Semantics and Soundness. *Electr. Notes Theor. Comput. Sci.* 158 (2006), 123–150. DOI:http://dx.doi.org/10.1016/j.entcs.2006.04.008

Cristiano Calcagno, Dino Distefano, and Peter O'Hearn. 2015. Open-sourcing Facebook Infer: Identify bugs before you ship. (2015). https://code.facebook.com/posts/1648953042007882/open-sourcing-facebook-infer-identify-bugs-before-you-ship/.

Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. 2011. Compositional Shape Analysis by Means of Bi-Abduction. *J. ACM* 58, 6 (2011), 26. DOI:http://dx.doi.org/10.1145/2049697.2049700 Preliminary version appeared in POPL'09.

Cristiano Calcagno, Dino Distefano, and Viktor Vafeiadis. 2009. Bi-abductive Resource Invariant Synthesis, See Hu [2009], 259–274. DOI:http://dx.doi.org/10.1007/978-3-642-10672-9_19

Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. 2007a. Local Action and Abstract Separation Logic. In *LICS*. IEEE Computer Society, 366–378.

Cristiano Calcagno, Matthew J. Parkinson, and Viktor Vafeiadis. 2007b. Modular Safety Checking for Fine-Grained Concurrency. In *SAS (LNCS)*, Hanne Riis Nielson and Gilberto Filé (Eds.), Vol. 4634. Springer, 233–248.

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*. 238–252.

Pedro da Rocha Pinto, Thomas Dinsdale-Young, Philippa Gardner, and Julian Sutherland. 2016. Modular Termination Verification for Non-blocking Concurrency. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. 176–201. DOI:http://dx.doi.org/10.1007/978-3-662-49498-1_8

E. W. Dijkstra. 1968. Cooperating Sequential Processes. In *Programming Languages*, F. Genuys (Ed.). Academic Press, 43–112. Reprinted in [**?**].

Thomas Dinsdale-Young, Lars Birkedal, Philippa Gardner, Matthew J. Parkinson, and Hongseok Yang. 2013. Views: compositional reasoning for concurrent programs. In *40th POPL*. 287–300.

Thomas Dinsdale-Young, Mike Dodds, Philippa Gardner, Matthew J. Parkinson, and Viktor Vafeiadis. 2010a. Concurrent Abstract Predicates. In *24th ECOOP*. 504–528.

Thomas Dinsdale-Young, Philippa Gardner, and Mark J. Wheelhouse. 2010b. Abstraction and Refinement for Local Reasoning. In *3rd VSTTE*. 199–215.

Mike Dodds, Xinyu Feng, Matthew J. Parkinson, and Viktor Vafeiadis. 2009. Deny-Guarantee Reasoning. In *18th ESOP*. 363–377.

Mike Dodds, Suresh Jagannathan, Matthew J. Parkinson, Kasper Svendsen, and Lars Birkedal. 2016. Verifying Custom Synchronization Constructs Using Higher-Order Separation Logic. *ACM Trans. Program. Lang. Syst.* 38, 2 (2016), 4. DOI:http://dx.doi.org/10.1145/2818638

Xinyu Feng, Rodrigo Ferreira, and Zhong Shao. 2007. On the Relationship Between Concurrent Separation Logic and Assume-Guarantee Reasoning. In *16th ESOP*. 173–188.

Xinyu Feng, Zhong Shao, Yuan Dong, and Yu Guo. 2008. Certifying low-level programs with hardware interrupts and preemptive threads. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*. 170–182. DOI:http://dx.doi.org/10.1145/1375581.1375603

Ming Fu, Yong Li, Xinyu Feng, Zhong Shao, and Yu Zhang. 2010. Reasoning about Optimistic Concurrency Using a Program Logic for History. In *CONCUR (Lecture Notes in Computer Science)*, Paul Gastin and François Laroussinie (Eds.), Vol. 6269. Springer, 388–402.

Alexey Gotsman, Josh Berdine, and Byron Cook. 2011. Precision and the Conjunction Rule in Concurrent Separation Logic. *Electr. Notes Theor. Comput. Sci.* 276 (2011), 171–190.

Alexey Gotsman, Josh Berdine, Byron Cook, and Mooly Sagiv. 2007. Thread-modular shape analysis. In *PLDI*, Jeanne Ferrante and Kathryn S. McKinley (Eds.). ACM, 266–277.

Christian Haack, Marieke Huisman, and Clément Hurlin. 2008. Reasoning about Java's Reentrant Locks. In *Programming Languages and Systems, 6th Asian Symposium, APLAS 2008, Bangalore, India, December 9-11, 2008. Proceedings*. 171–187. DOI:http://dx.doi.org/10.1007/978-3-540-89330-1_13

Christian Haack and Clément Hurlin. 2008. Separation Logic Contracts for a Java-Like Language with Fork/Join. In *Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008, Urbana, IL, USA, July 28-31, 2008, Proceedings*. 199–215. DOI:http://dx.doi.org/10.1007/978-3-540-79980-1_16

Jonathan Hayman and Glynn Winskel. 2008. Independence and Concurrent Separation Logic. *Logical Methods in Computer Science* 4, 1 (2008).

C. A. R. Hoare. 1972. Towards a theory of parallel programming. In *Operating Systems Techniques*, Hoare and Perrot (Eds.). Academic Press.

Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. 2011. Concurrent Kleene Algebra and its Foundations. *J. Log. Algebr. Program.* 80, 6 (2011), 266–296.

Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, Jules Villard, Huibiao Zhu, and Peter W. O'Hearn. 2014. Developments in Concurrent Kleene Algebra. In *Relational and Algebraic Methods in Computer Science - 14th International Conference, RAMiCS 2014, Marienstatt, Germany, April 28-May 1, 2014. Proceedings*. 1–18. DOI:http://dx.doi.org/10.1007/978-3-319-06251-8_1

Aquinas Hobor, Andrew W. Appel, and Francesco Zappa Nardelli. 2008. Oracle Semantics for Concurrent Separation Logic. In *ESOP (LNCS)*, Sophia Drossopoulou (Ed.), Vol. 4960. Springer, 353–367.

Zhenjiang Hu (Ed.). 2009. *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings*. LNCS, Vol. 5904. Springer.

Samin S. Ishtiaq and Peter W. O'Hearn. 2001. BI as an Assertion Language for Mutable Data Structures. In *POPL*. 14–26.

Bart Jacobs, Jan Smans, and Frank Piessens. 2016. VeriFast project website. (2016). https://people.cs.kuleuven.be/ bart.jacobs/verifast/.

C. B. Jones. 1983. Specification and design of (parallel) programs. (1983). *IFIP Conference*.

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 637–650. DOI:http://dx.doi.org/10.1145/2676726.2676980

Johannes Kloos, Rupak Majumdar, and Viktor Vafeiadis. 2015. Asynchronous Liquid Separation Types. In *29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic*. 396–420. DOI:http://dx.doi.org/10.4230/LIPIcs.ECOOP.2015.396

Jinjiang Lei, Zongyan Qiu, and Zhong Shao. 2014. Trace-Based Temporal Verification for Message-Passing Programs. In *2014 Theoretical Aspects of Software Engineering Conference, TASE 2014, Changsha, China, September 1-3, 2014*. 10–17. DOI:http://dx.doi.org/10.1109/TASE.2014.14

Hongjin Liang and Xinyu Feng. 2016. A program logic for concurrent objects under fair scheduling. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. 385–399. DOI:http://dx.doi.org/10.1145/2837614.2837635

Peter Müller and Alex Summers. 2016. Chalice project website. (2016). http://www.pm.inf.ethz.ch/research/chalice.html.

Peter W. O'Hearn. 2007. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.* 375, 1-3 (2007), 271–307. Prelim version appeared in the proceedings of CONCUR'04.

Peter W. O'Hearn, Rasmus Lerchedahl Petersen, Jules Villard, and Akbar Hussain. 2015. On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra. *J. Log. Algebr. Meth. Program.* 84, 3 (2015), 285–302. DOI:http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local Reasoning about Programs that Alter Data Structures. In *CSL (LNCS)*, Laurent Fribourg (Ed.), Vol. 2142. Springer, 1–19.

S. Owicki and D. Gries. 1976a. An axiomatic proof technique for parallel programs. *Acta Informatica* 19 (1976), 319–340.

S. Owicki and D. Gries. 1976b. Verifying properties of parallel programs: An axiomatic approach. *Comm. ACM* 19, 5 (1976), 279–285.

David Michael Ritchie Park. 1979. On the Semantics of Fair Parallelism. In *Abstract Software Specifications (Lecture Notes in Computer Science)*, Dines Bjørner (Ed.), Vol. 86. Springer, 504–526.

Matthew J. Parkinson, Richard Bornat, and Cristiano Calcagno. 2006. Variables as Resource in Hoare Logics. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*. 137–146. DOI:http://dx.doi.org/10.1109/LICS.2006.52

Matthew J. Parkinson, Richard Bornat, and Peter W. O'Hearn. 2007. Modular verification of a non-blocking stack. In *POPL*, Martin Hofmann and Matthias Felleisen (Eds.). ACM, 297–302.

Matthew J. Parkinson and Alexander J. Summers. 2012. The Relationship Between Separation Logic and Implicit Dynamic Frames. *Logical Methods in Computer Science* 8, 3 (2012). DOI:http://dx.doi.org/10.2168/LMCS-8(3:1)2012

A. Pnueli. 1981. The temporal semantics of concurrent programs. (1981). *Theoretical Computer Science*, 13(1), 45–60.

D. Pym, P. O'Hearn, and H. Yang. 2004. Possible worlds and resources: the semantics of BI. *Theoretical Computer Science* 315, 1 (2004), 257–305.

Azalea Raad, Jules Villard, and Philippa Gardner. 2015. CoLoSL: Concurrent Local Subjective Logic. In *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. 710–735. DOI:http://dx.doi.org/10.1007/978-3-662-46669-8_29

John C. Reynolds. 2000. Intuitionistic Reasoning about Shared Mutable Data Structure. In *Millennial Perspectives in Computer Science (Cornerstones of Computing)*, Jim Davies, Bill Roscoe, and Jim Woodcock (Eds.). Palgrave Macmillan.

John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *LICS*. IEEE Computer Society, 55–74.

Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. 2015. Mechanized verification of fine-grained concurrent programs. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*. 77–87. DOI:http://dx.doi.org/10.1145/2737924.2737964

Kasper Svendsen, Lars Birkedal, and Matthew J. Parkinson. 2013. Joins: A Case Study in Modular Specification of a Concurrent Reentrant Higher-Order Library. In *ECOOP 2013 - Object-Oriented Programming - 27th European Conference, Montpellier, France, July 1-5, 2013. Proceedings*. 327–351. DOI:http://dx.doi.org/10.1007/978-3-642-39038-8_14

Viktor Vafeiadis. 2011. Concurrent Separation Logic and Operational Semantics. *Electr. Notes Theor. Comput. Sci.* 276 (2011), 335–351.

Viktor Vafeiadis and Chinmay Narayan. 2013. Relaxed separation logic: a program logic for C11 concurrency. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*. 867–884. DOI:http://dx.doi.org/10.1145/2509136.2509532

Viktor Vafeiadis and Matthew J. Parkinson. 2007. A Marriage of Rely/Guarantee and Separation Logic. In *CONCUR (LNCS)*, Luís Caires and Vasco Thudichum Vasconcelos (Eds.), Vol. 4703. Springer, 256–271.

Jules Villard, Étienne Lozes, and Cristiano Calcagno. 2009. Proving Copyless Message Passing, See Hu [2009], 194–209.

Jules Villard, Étienne Lozes, and Cristiano Calcagno. 2010. Tracking Heaps That Hop with Heap-Hop. In *TACAS (LNCS)*, Javier Esparza and Rupak Majumdar (Eds.), Vol. 6015. Springer, 275–279.

Fengwei Xu, Ming Fu, Xinyu Feng, Xiaoran Zhang, Hui Zhang, and Zhaohui Li. 2016. A Practical Verification Framework for Preemptive OS Kernels. (2016). CAV.

VERIFICATION COLUMN

NEHA RUNGTA, SGT Inc./NASA Ames Research Center
neha.s.rungta@nasa.gov

The technical column on verification presents an invited contribution by César A. Muñoz, Aaron Dutle, Anthony Narkawicz, and Jason Upchurch at NASA Langley Research Center titled *Unmanned Aircraft Systems in the National Airspace System: A Formal Methods Perspective* in the July issue of the SIGLOG newsletter. In the past few years we have seen a dramatic increase in the number of hobbyist unmanned aircraft systems (UAS). In addition we have seen a push by several companies including Amazon to use UAS as a package delivery mechanism. The increasing numbers of UAS in our current airspace have introduced a new safety challenge. There have been some high-profile incidents that were reported widely in the media. One example was a drone being 100 feet within an commercial airliner at the JFK airport in New York in Aug 2015, while another was a near miss at the Los Angeles airport between a Lufthansa passenger jumbo jet and a UAS in March 2016. These are however not isolated incidents. The Federal Aviation Administration (FAA) records shows that there are hundreds of close calls between airplanes and UAS in the US airspace. Similar reports have been documented in the UK and Europe. This safety concern has prompted the development of a detect and avoid capability for the UAS that is designed to avoid aircraft in its vicinity. The article presents the work being done at the NASA Langley Research Center Formal Methods group in Hampton Virigina. I am very excited for this article to appear in the SIGLOG newsletter. This article I believe will be of interest to people not only in the logic, formal methods, and verification community but a broader and more general audience as well.

Unmanned Aircraft Systems in the National Airspace System: A Formal Methods Perspective

César A. Muñoz, NASA
Aaron Dutle, NASA
Anthony Narkawicz, NASA
Jason Upchurch, NASA

As the technological and operational capabilities of unmanned aircraft systems (UAS) have grown, so too have international efforts to integrate UAS into civil airspace. However, one of the major concerns that must be addressed in realizing this integration is that of safety. For example, UAS lack an on-board pilot to comply with the legal requirement that pilots see and avoid other aircraft. This requirement has motivated the development of a detect and avoid (DAA) capability for UAS that provides situational awareness and maneuver guidance to UAS operators to aid them in avoiding and remaining well clear of other aircraft in the airspace. The NASA Langley Research Center Formal Methods group has played a fundamental role in the development of this capability. This article gives a selected survey of the formal methods work conducted in support of the development of a DAA concept for UAS. This work includes specification of low-level and high-level functional requirements, formal verification of algorithms, and rigorous validation of software implementations.

1. INTRODUCTION

In their 2013 economic report, the Association for Unmanned Vehicle Systems International (AUVSI) [Jenkins and Vasigh 2013] projected the cumulative impact of the Unmanned Aircraft Systems (UAS) industry on the US economy between 2015 and 2025 to be more than US $80 billion and the generation of more than one hundred thousand jobs. As the availability of and applications for UAS grow in the US and worldwide, there have been concerted efforts by stakeholders throughout the international community aimed at addressing the problem of safely integrating UAS into standard airspace operations. NASA's Unmanned Aircraft Systems Integration in the National Airspace System (UAS in the NAS) project aims to developing key capabilities to enable routine and safe access for public and civil use of UAS in non-segregated airspace operations.

One of the major challenges to the safe integration of UAS into the NAS is the lack of an on-board pilot to comply with particular US and international legal requirements. In manned aircraft operations on-board pilots have, in part, the responsibility for not "operating an aircraft so close to another aircraft as to create a collision hazard" [International Civil Aviation Organization (ICAO) 2005a; US Code of Federal Regulations 1967a], "to see and avoid other aircraft" [International Civil Aviation Organization (ICAO) 2005b; US Code of Federal Regulations 1967b], and when complying with the particular rules addressing right-of-way, on-board pilots "may not pass over, under, or ahead [of the right-of-way aircraft] unless well clear" [International Civil Aviation Organization (ICAO) 2005b; US Code of Federal Regulations 1967b]. To address the safety challenge and establish parallel requirements for UAS, the final report of the Federal Aviation Administration (FAA) Sense and Avoid (SAA) Workshop [FAA Sponsored Sense and Avoid Workshop 2009] defined the concept of *sense and avoid* as "the capability of a UAS to remain well clear from and avoid collisions with other airborne traffic." This definition has been proposed as a means of compliance with the preceding legal requirements. In the case of manned aircraft operations, the ability to remain well clear and see and avoid other aircraft depends upon the perception and judgement of the human pilot. In absence of an on-board pilot, there is a need for a formal understanding of the notion of *well-clear*, which resolves this ambiguity and is appropriate

for integrated UAS operations [Crück and Lygeros 2007; Coulter 2009; Tomasello and Haddon 2011; Weibel et al. 2011; Adaska 2012; Theunissen et al. 2014; Consiglio et al. 2012]. In recent years, efforts to provide such a definition have been underway.

In 2011, the Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics established the UAS Sense and Avoid Science and Research Panel (SARP) and charged it with making a recommendation for a quantitative definition of well clear. In 2013, the RTCA organization established Special Committee 228 (SC-228) to provide technical guidance to the FAA for defining minimum operational performance standards for a UAS sense and avoid concept, which is now called *detect and avoid*, based on the quantitative definition of well-clear recommended by the UAS SARP. The NASA Langley Research Center (LaRC) Formal Methods group closely collaborated with the UAS SARP to both identify a mathematical definition of well clear that was appropriate for UAS operations and verify that it satisfied operational requirements. The LaRC Formal Methods group is currently participating in the RTCA SC-228, and has responsibility for the specification, development, and verification of a reference implementation of the algorithms that support the overall UAS DAA concept. This article presents an overview of the research undertaken by the LaRC Formal Methods group while collaborating with these groups, and addresses some of the key technical challenges in Formal Methods research in the context of the development and safety analysis of advanced air traffic management concepts.

The mathematical formulas and theorems presented in the subsequent sections have been formally specified and verified in the Prototype Verification System (PVS) [Owre et al. 1992].[1] However, to make the article accessible to a broader audience, the formulas and theorems herein are expressed in mathematical notation instead of concrete PVS syntax. Furthermore, the states of the aircraft pair of interest, denoted *ownship* and *intruder*, are represented by position and velocity vectors in a local East, North, Up (ENU) Cartesian coordinate system. This coordinate system is based on the orthogonal projection of the ownship and intruder geodesic coordinates onto a plane tangent to the projected ownship position on the surface of the earth. For notational convenience, horizontal and vertical components of a three-dimensional vector are represented by a two-dimensional vector and a scalar, respectively, and these components are given in a relative coordinate system, where the intruder is at the origin, and the ownship moves relative to the intruder. Letters in bold-face, e.g., $\mathbf{v}$, $\mathbf{s}$, denote two-dimensional vectors. Finally, vector operations such as addition, subtraction, scalar multiplication, dot product, i.e., $\mathbf{s} \cdot \mathbf{v} \equiv s_x v_x + s_y v_y$, the square of a vector, i.e., $\mathbf{s}^2 \equiv \mathbf{s} \cdot \mathbf{s}$, and the norm of a vector, i.e., $\|\mathbf{s}\| \equiv \sqrt{\mathbf{s}^2}$, are defined in a two-dimensional Euclidean geometry.

2. WELL-CLEAR VOLUME

Consiglio et al. proposed a UAS detect and avoid concept where the well-clear notion is defined by a protected volume around the UAS [Consiglio et al. 2012]. If no traffic aircraft is located inside this volume, the UAS is considered to be well clear. A key idea in this proposal is that the protected volume is assumed to be large enough to avoid resolution advisories from a collision avoidance system, but small enough to avoid disruption to traffic flow. Subsequently, several candidate definitions of the well-clear volume were proposed to the UAS SARP, and they were analyzed with respect to these conceptual requirements as well as other operational requirements [Cook et al. 2015].

The definition of the well-clear volume ultimately recommended by the UAS SARP, and adopted by RTCA SC-228, is a boolean formula based on the second generation of the Traffic Alerting and Collision Avoidance System (TCAS II) Resolution Advisory

[1]For further information on this formal development, the reader is referred to the directories `TCASII`, `WellClear`, and `DAIDALUS` available in the NASA PVS Library (https://github.com/nasa/pvslib).

(RA) detection logic. In particular, the well-clear volume is defined by a predicate on the relative position and velocity vectors of the ownship and intruder at the current time. This predicate determines that two aircraft are *well clear* of each other when computed distance and time functions fall outside a set of predefined threshold values. The particular distance and time functions used in the definition of the well-clear volume are based on those used in the TCAS II RA detection logic [Muñoz et al. 2013] and are discussed in the subsequent presentation.

A well-clear violation is defined as a situation when there is both a horizontal and a vertical violation. That is, the predicate defining the well-clear volume is a conjunction of two predicates, one representing the horizontal dimension and the other representing the vertical dimension, as given in Formula (1).

$$\mathit{WCV}(\mathbf{s}, s_z, \mathbf{v}, v_z) \equiv \mathit{HWCV}(\mathbf{s}, \mathbf{v}) \wedge \mathit{VWCV}(s_z, v_z), \tag{1}$$

Here $\mathbf{s}, \mathbf{v} \in \mathbb{R}^2$ are the respective relative horizontal position and velocity vectors of the aircraft pair, and $s_z, v_z \in \mathbb{R}$ are the respective relative vertical position and velocity of the aircraft pair. The horizontal and vertical violation predicates are defined in Formula (2) and Formula (3), respectively.

$$\mathit{HWCV}(\mathbf{s}, \mathbf{v}) \equiv \|\mathbf{s}\| \leq \texttt{DMOD} \vee (\mathit{HMDF}(\mathbf{s}, \mathbf{v}) \wedge 0 \leq \tau_{\text{mod}}(\mathbf{s}, \mathbf{v}) \leq \texttt{TAUMOD}), \tag{2}$$

$$\mathit{VWCV}(s_z, v_z) \equiv |s_z| \leq \texttt{ZTHR} \vee 0 \leq t_{\text{coa}}(s_z, v_z) \leq \texttt{TCOA}, \tag{3}$$

where TAUMOD and DMOD are horizontal time and distance thresholds, respectively, and TCOA and ZTHR are vertical time and distance thresholds, respectively. The predicate *HMDF* refers to the *horizontal miss-distance filter* and is defined as in Formula (4).

$$\mathit{HMDF}(\mathbf{s}, \mathbf{v}) \equiv d_{\text{cpa}}(\mathbf{s}, \mathbf{v}) \leq \texttt{HMD}, \tag{4}$$

where HMD is the horizontal miss-distance threshold and is typically set to the same value as DMOD. The distance function d_{cpa} computes the projected horizontal distance between the aircraft at their closest point of approach in the horizontal dimension, assuming constant relative horizontal velocity $\mathbf{v}$, and is formally defined in Formula (5).

$$d_{\text{cpa}}(\mathbf{s}, \mathbf{v}) \equiv \|\mathbf{s} + t_{\text{cpa}}(\mathbf{s}, \mathbf{v})\mathbf{v}\|. \tag{5}$$

The time function t_{cpa} in Formula (5) is the time to closest point of approach, and is defined as

$$t_{\text{cpa}}(\mathbf{s}, \mathbf{v}) \equiv \begin{cases} -\frac{\mathbf{s} \cdot \mathbf{v}}{\mathbf{v}^2} & \text{if } \|\mathbf{v}\| \neq 0, \\ 0 & \text{otherwise}, \end{cases} \tag{6}$$

where the inequality $\mathbf{s} \cdot \mathbf{v} < 0$ holds when the aircraft are horizontally converging, $\mathbf{s} \cdot \mathbf{v} > 0$ holds when the aircraft are horizontally diverging, and $\mathbf{s} \cdot \mathbf{v}$ equals 0 when the aircraft are at their horizontal closest point of approach.

The time function τ_{mod} (*modified tau*) was introduced in the TCAS II RA logic [Hammer 1996]. In the vector notation used in this article, modified tau is defined in Formula (7).

$$\tau_{\text{mod}}(\mathbf{s}, \mathbf{v}) \equiv \begin{cases} \frac{\texttt{DMOD}^2 - \mathbf{s}^2}{\mathbf{s} \cdot \mathbf{v}} & \text{if } \mathbf{s} \cdot \mathbf{v} < 0, \\ -1 & \text{otherwise}. \end{cases} \tag{7}$$

The time function t_{coa} computes the time to co-altitude, assuming constant relative vertical speed v_z, and is defined in Formula (8).

$$t_{\text{coa}}(s_z, v_z) \equiv \begin{cases} -\frac{s_z}{v_z} & \text{if } s_z v_z < 0, \\ -1 & \text{otherwise}. \end{cases} \tag{8}$$

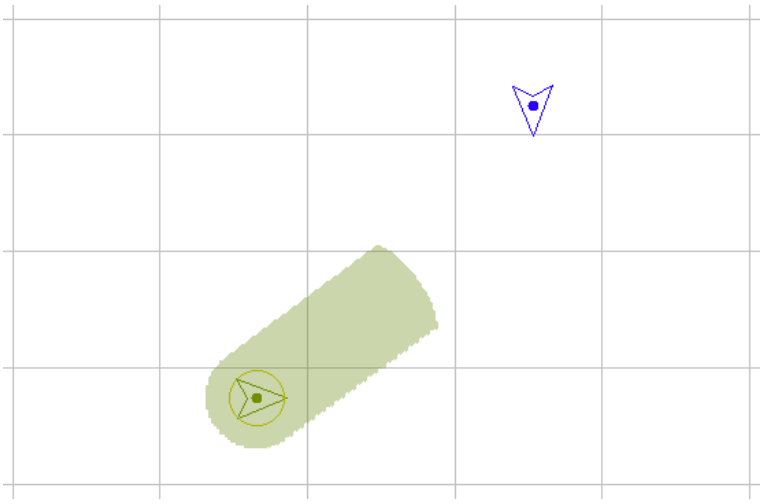


Fig. 1. Top View of Well-Clear Volume

In a way similar to the horizontal case, the product $s_z v_z$ characterizes whether the aircraft are vertically diverging, i.e., $s_z v_z > 0$, or vertically converging, i.e., $s_z v_z < 0$. For completeness, time to co-altitude is defined as -1 when the aircraft are not vertically converging.

The values of the distance and time thresholds recommended by the UAS SARP [Cook et al. 2015] and adopted by the RTCA SC-228 are $\texttt{DMOD} = \texttt{HMD} = 4000$ ft, $\texttt{ZTHR} = 450$ ft, $\texttt{TAUMOD} = 35$ s, and $\texttt{TCOA} = 0$ s. The shaded area in Figure 1 illustrates the resultant top view of the ownship's well-clear volume with respect to an intruder aircraft for an example encounter. Any traffic aircraft located inside this area with the same velocity vector as the intruder aircraft will be in well-clear violation with the UAS. The diameter of the circular base of the well-clear volume is DMOD. The length of the elongated shape depends on TAUMOD. The shape is elongated in the direction of the relative velocity vector.

For arbitrary values of DMOD, ZTHR, TAUMOD, and TCOA, with $\texttt{HMD} = \texttt{DMOD}$, Formula (1) satisfies several operational requirements [Muñoz et al. 2014]. For instance, in a pairwise encounter, the ownship and intruder aircraft make the same determination of their well-clear status. That, both aircraft are simultaneously aware of an in agreement on a well-clear violation. This property is called *symmetry*, and it is formally stated in Theorem 2.1.

THEOREM 2.1 (SYMMETRY). *For all relative states* $\mathbf{s}, s_z, \mathbf{v}, v_z$, *WCV*$(\mathbf{s}, s_z, \mathbf{v}, v_z) =$ *WCV*$(-\mathbf{s}, -s_z, -\mathbf{v}, -v_z)$.

Another important property states that for straight line trajectories there is at most one time interval where the aircraft are not well clear. This property is called *local convexity*, and it enables the definition of an alerting algorithm that, in a non-maneuvering encounter, continuously alerts a predicted well-clear violation until the violation disappears. Once the violation disappears, it does not reappear unless the aircraft maneuver. This property is formally stated in Theorem 2.2.

THEOREM 2.2 (LOCAL CONVEXITY). *For all relative states* $\mathbf{s}, s_z, \mathbf{v}, v_z$, *if there are times* $0 \leq t_1 \leq t_2$ *such that* *WCV*$(\mathbf{s}+t_1\mathbf{v}, s_z+t_1 v_z, \mathbf{v}, v_z)$ *and* *WCV*$(\mathbf{s}+t_2\mathbf{v}, s_z+t_2 v_z, \mathbf{v}, v_z)$, *then for all times* $t1 \leq t \leq t2$, *WCV*$(\mathbf{s}+t\mathbf{v}, s_z+tv_z, \mathbf{v}, v_z)$.

The well-clear volume defined by Formula (1) assumes perfect aircraft state information. To accomodate for uncertainty in the position and velocity information, the RTCA SC-228 requirements for the well-clear alerting logic allows for the use of a larger set of threshold values within some specified ranges, giving an *extended well-clear volume*. This extended well-clear volume is characterized by a predicate WCV^* defined by Formula (1), using parameters $\texttt{DMOD}^* \geq \texttt{DMOD}$, $\texttt{HMD}^* \geq \texttt{HMD}$, $\texttt{ZTHR}^* \geq \texttt{ZTHR}$, $\texttt{TAUMOD}^* \geq \texttt{TAUMOD}$, and $\texttt{TCOA}^* \geq \texttt{TCOA}$. Theorem 2.3 formally states that the well-clear volume instantiated with the RTCA SC-228 standard threshold values is safely included in any of its extensions.

THEOREM 2.3 (EXTENSION). *For all relative states* $\mathbf{s}, s_z, \mathbf{v}, v_z$, $WCV(\mathbf{s}, s_z, \mathbf{v}, v_z) \implies WCV^*(\mathbf{s}, s_z, \mathbf{v}, v_z)$.

The extension property enables the definition of formally verified alerting and maneuvering algorithms that *completely* protect against the violation of the standard well-clear volume by using a more conservative definition.

3. WELL-CLEAR AND TCAS II INTEROPERABILITY

TCAS is a family of airborne devices that are designed to reduce the risk of mid-air collisions between aircraft equipped with operating transponders. TCAS II [RTCA SC-147 2009], the current generation of TCAS devices, is mandated in the US for aircraft with greater than 30 seats or a maximum takeoff weight greater than 33,000 pounds. Although not required, TCAS II is also installed on many turbine-powered general aviation aircraft. TCAS II provides *resolution advisories* (RAs), which are visual and vocalized alerts that direct pilots to maintain or increase vertical separation with intruders that are considered collision threats.

Formula (1), which defines the UAS well-clear volume, closely follows the formula that defines the logic of the TCAS II RA detection algorithm. The primary difference between the well-clear volume, as defined by RTCA SC-228, and the TCAS II RA volume is in the particular choice of time and distance threshold values. The TCAS II RA detection logic uses a lookup table indexed by the altitude of the ownship to determine which set of threshold values to use. For some entries, e.g., for aircraft flying above 10,000 ft, the TCAS II RA threshold values are higher than the well-clear threshold values. Due to these particular instances of larger threshold values for TCAS II, the well-clear volume is not a proper extension of the TCAS II RA volume. Hence, Theorem 2.3 does not apply to these cases. Thus, in practice, it is possible that TCAS II issues an RA before the ownship declares a well-clear violation. The vertical threshold values in the well-clear definition are also problematic. Indeed, ZTHR and TCOA for the well-clear volume are set to 450 ft and 0 s, respectively, and these values are always smaller than the corresponding threshold values in the TCAS II RA table. This choice of values affects the interoperability of the UAS DAA concept with TCAS II in encounters with high vertical closure rate [Upchurch et al. 2015].

There are operational justifications for this mismatch between the well-clear definition and the TCAS II RA threshold values. For instance, the detect and avoid concept considered by RTCA SC-228 only applies to certain types of UAS and in classes of airspace that are usually below 10,000 ft, that is, Class D, Class E, and perhaps Class G airspace. Additionally, the negative effects of this mismatch, i.e., TCAS II RAs preceding well-clear violations, may be mitigated by the use of extended well-clear volumes in the alerting and maneuvering algorithms, which are also part of the RTCA SC-228 UAS detect and avoid concept. Theorem 3.1 states that there are extended well clear volumes that properly include the TCAS II RA volume.

THEOREM 3.1 (INCLUSION). *Let* $\mathtt{HMD}^* = \mathtt{DMOD}^*$, $\mathtt{TAUMOD}^* = \mathtt{TCOA}^*$, *and* $\mathtt{ZTHR}^*$ *be threshold values larger than the corresponding TCAS II RA threshold values. For all relative states* $\mathbf{s}, s_z, \mathbf{v}, v_z$, $\mathit{TCASII\_RA}(\mathbf{s}, s_z, \mathbf{v}, v_z) \implies \mathit{WCV}^*(\mathbf{s}, s_z, \mathbf{v}, v_z)$.

Despite Theorem 3.1, it has been observed in flight tests that it is still possible for TCAS II to issue an RA before a violation of an extended well-clear volume. The reason for this apparent inconsistency is that Theorem 3.1 assumes that the same state information, in vector form, is available to both systems, yet this is not the case. In the case of the well-clear concept, state information in vector form is readily available through modern global positioning systems. On the other hand, TCAS II assumes that aircraft are equipped with active transponders, and the lack of reliable vector information is compensated for by a sophisticated tracking system. This tracking system is a key component of the TCAS II RA system, and depending on the quality of the range rate estimate computed by the tracker and other conditions, the TCAS II RA system may disable the use of the horizontal miss distance filter in Formula (2), i.e., $\mathit{HMDF}(\mathbf{s}, \mathbf{v})$ is considered to be true, leading to the case of TCAS II RAs preceding violation of the extended well-clear volume.

4. DAIDALUS

The RTCA SC-228 Minimum Operational Performance Standards (MOPS) for Unmanned Aircraft Systems includes a reference implementation of DAA algorithms that assist remote pilots by providing situational awareness of proximity to other aircraft in the airspace. This reference implementation, called DAIDALUS (Detect & Avoid Alerting Logic for Unmanned Systems [Muñoz et al. 2015], is being developed by the LaRC Formal Methods group in support of NASA's Safe Autonomous System Operations (SASO) project. The DAIDALUS source code is available in both C++ and Java under NASA's Open Source Agreement[2].

The algorithms contained in DAIDALUS compute: (1) predictions of well-clear violations between the ownship and a given intruder aircraft, (2) maneuver guidance for the ownship to maintain or regain well-clear status with respect to all traffic aircraft, and (3) an alert level representing the severity of a potential well-clear violation between the ownship and a given intruder aircraft. A fundamental element of DAIDALUS is the algorithm that computes the time interval of violation for a non-maneuvering encounter. This algorithm, called `wc_interval`, has as inputs a relative aircraft state and a non-empty lookahead time interval $[B, T]$, with $0 \leq B < T$. It returns the time interval $[t_{\text{in}}, t_{\text{out}}]$, which corresponding to the interval of a predicted well-clear violation, assuming constant-velocity trajectories. The returned interval is empty if no such violation is predicted. Theorem 4.1 states that `wc_interval` is correct and complete.

THEOREM 4.1 (CORRECT AND COMPLETE). *For all relative states* $\mathbf{s}, s_z, \mathbf{v}, v_z$ *and times* B, T, *with* $0 \leq B < T$, *let* $[t_{in}, t_{out}]$ *be the time interval returned by* $\mathtt{wc\_interval}(\mathbf{s}, s_z, \mathbf{v}, v_z, B, T)$ *and* t *be a time in* $[B, T]$, *then* $\mathit{WCV}(\mathbf{s} + t\mathbf{v}, s_z + tv_z, \mathbf{v}, v_z)$ *if and only if* $t \in [t_{in}, t_{out}]$.

The local convexity property given by Theorem 2.2 is a necessary condition for the existence of an algorithm, such as `wc_interval`, that satisfies Theorem 4.1. Without local convexity, either one of the implications in the theorem must be removed.

The algorithms that comprise DAIDALUS are specified in PVS, and a large number of their functional requirements, such as Theorem 4.1, are proven to hold in these formal models of the algorithms. The DAIDALUS algorithms are implemented in Java and C++, and the formal verification of these implementations, while possible in some

[2]http://github.com/nasa/wellclear.

cases, is exceedingly difficult. The difficulty arises due to the use of floating point arithmetic in the implementations, while the formal verification of these algorithms is done over the real numbers. This difficulty is also partially due to the object-oriented nature of these languages versus the functional PVS notation. The difficulty is further compounded when, as in the case of DAIDALUS, the properties to be verified are themselves complex.

The validation of the DAIDALUS software implementation is performed using a more pragmatic approach named *model animation* [Dutle et al. 2015]. While model animation does not offer the same high level of safety assurance as a complete formal verification, it addresses the numerical issues in practical way, while still offering strong assurance that the implemented software performs to its specification. Three elements are required in order to implement the technique: (1) there must be an executable formal model of the software, which has been verified to possess the desired properties of the actual software, (2) the software to be assessed needs to be a close translation from the formal specification into the desired language, and (3) a collection of representative test cases must be selected in some manner decided by the user. Each test case is then evaluated using the formal model and the software implementation, and the results are compared. If the results are sufficiently close for an acceptable number of the test cases, then the software is considered to faithfully implement the formal model. However, there are details which present technical challenges in the model animation process. The algorithms in DAIDALUS use mathematical functions such as roots, trigonometric, and inverse trigonometric functions. Due to the presence of irrational and transcendental numbers, the outputs computed by these algorithms are not exact. Nonetheless, other than the presence some of these non-computable functions, the formal specifications of the DAIDALUS algorithms are executable using the PVS ground evaluator. In order to fully execute these algorithms, PVS includes an animation tool called PVSio [Muñoz 2003], which extends the ground evaluator, notably by providing support for *semantic attachments*; a semantic attachment is a way to replace a PVS function call with a call to a trusted oracle. In its default mode, PVSio replaces non-computable mathematical functions with internal LISP floating point functions. When more precision is needed, or when floating point evaluation must be avoided, these standard functions are replaced with a collection of attachments that are formally verified in PVS to be correct up to a given, but arbitrary, precision.

Other fundamental model animation design decisions relate to the selection of a proper collection of test cases, how to determine when outputs are "sufficiently close," and how many disagreements between the software and the model are acceptable. The selection of a set of test cases for this method can be done in any number of ways. Test cases might be selected to ensure some code coverage criteria are met, or a large number of random cases may be chosen to allow for a wide variety of inputs to be generated, or other desired criteria may go into the selection process. In the case of assessing the DAA functionality of DAIDALUS, the test set chosen began as a collection of 95 aircraft encounter scenarios, which were developed jointly by the USAF, MIT Lincoln Laboratory, and NASA during the UAS SARP activity. These scenarios represent a suite of stressing cases which were designed to be difficult cases for the well-clear logic. Throughout the DAIDALUS development and assessment process, a number of additional stressing scenarios have been provided by the FAA, and others, and added to the test set.

For some of the functionality provided by DAIDALUS, testing whether the formal models and their reference implementations agree or not is straightforward. For example, the well-clear violation logic computes a boolean value indicating the well-clear status between the ownship and an intruder aircraft. Testing if this logic is correctly implemented in software amounts to checking if the same boolean value is computed

in both the formal model and the software implementation. However, for algorithms that compute a numerical value, the verification is less straightforward. In such cases, even very close outputs might be different due to numerical approximations. For these numerical differences, an allowable tolerance for each particular output is determined. If the difference between the formal model and the implementation is within this tolerance, then the two are declared to be in agreement. These tolerances are determined on a case-by-case basis, based on the needed accuracy of the output. For example, if the output is a time value intended to be displayed to an operator in whole number seconds then a tolerance for such a value might be set to 0.5 seconds. Furthermore, if the output is a distance based on ADS-B, then a tolerance of one meter might be sufficient, since the guaranteed accuracy of ADS-B is within approximately five meters. The choices made for these tolerances also directly affect the number of cases in which the software and formal model disagree, which is the final criterion for determining whether or not they agree, overall.

In the case of DAIDALUS, this validation procedure is being performed in conjunction with the development of the software. As such, the procedure not only validates that the software is in agreement with its formal model, but also reveals places where the software must be modified to be in closer agreement with its formal model. At each step in this iterative process, the DAIDALUS software implementation is brought closer to matching its formal model in all of its functionality, thus providing a high level of assurance that the implemented software retains the safety properties proven in the formal model.

5. CONCLUSION

Motivated by the safety-critical nature of a detect and avoid concept and the corresponding need for strong assurances and mathematical guarantees, Formal Methods research has contributed to the formal development of a DAA concept for UAS. The use of Formal Methods in the DAA problem includes a formal definition of the well-clear violation volume, formal proofs of its key properties, formal specification and verification of DAA algorithms, and the rigorous validation of the software implementation of these algorithms against their formal specifications. All formal specifications and proofs supporting this work are written and mechanically verified in the interactive theorem prover, PVS.

The application of Formal Methods to the safety analysis of air traffic management systems faces technical challenges common to complex cyber-physical systems (CPS). Chief among these challenges is the interaction of these systems with the physical environment that yields mathematical models with both continuous and discrete behaviors. Formally proving properties involving continuous mathematics, such non-linear arithmetic, in particular, is a well-known problem in automated deduction. As part of this research effort, several automated decision and semi-decision procedures for dealing with different kinds of non-linear real arithmetic problems have been developed [Narkawicz et al. 2015; Moscato et al. 2015; Denman and Muñoz 2014; Narkawicz and Muñoz 2014; Muñoz and Narkawicz 2013]. Most of these procedures are formally verified and are available as proof-producing automated strategies in the PVS theorem prover.

The formal verification of software implementations of a CPS is a major endeavor, even when the algorithms that are implemented have been formally verified. The main difficulty arises from the fact that modern programming languages use floating point arithmetic, while formal verification is usually performed over the real numbers. Furthermore, there is a large semantic gap between modern programming languages and the functional notation used in formal tools such as PVS. In the research discussed in this paper, model animation is used as a practical approach to the validation of

numerical software. Model animation compares computations performed in the software implementations against those symbolically evaluated to an arbitrary precision on the corresponding formal models. While this approach does not provide an absolute guarantee that the software is correct, it increases the confidence that the formal models are faithfully implemented in code. Finally, air traffic management systems are unique in many aspects. For instance, these systems involve human and automated elements and these elements are often subject to strict operational and legal requirements. These requirements restrict the design space of operational concepts, such as DAA for UAS. More importantly, new concepts and algorithms have to support an incremental evolution of the airspace system on a global scale. Thus, solutions may result which are non-optimal from a theoretical point of view, or which may have complex verification issues due to legacy systems such as TCAS. This article has presented a survey of Formal Methods research and applications to the practical problem of safely integrating UAS into the NAS, a design space thus constrained by requirements for safety, interoperability with legacy systems, regulatory compliance, and public trust.

ACKNOWLEDGMENTS

The work presented in this paper was conducted in support of the Unmanned Aircraft Systems Integration in the National Airspace System project and the Safe Autonomous System Operations project at NASA.

REFERENCES

Jason Adaska. 2012. Computing risk for Unmanned Aircraft self separation with maneuvering intruders. In *Proceedings of the 31st IEEE/AIAA Digital Avionics Systems Conference (DASC)*. 8A4–1–8A4–10.

María Consiglio, James Chamberlain, César Muñoz, and Keith Hoffler. 2012. Concept of integration for UAS operations in the NAS. In *Proceedings of 28th International Congress of the Aeronautical Sciences, ICAS 2012*. Brisbane, Australia.

Stephen P. Cook, Dallas Brooks, Rodney Cole, Davis Hackenberg, and Vincent Raska. 2015. Defining Well Clear for Unmanned Aircraft Systems. In *Proceedings of the 2015 AIAA Infotech @ Aerospace Conference*. Kissimmee, Florida.

Dennis Coulter. 2009. UAS integration into the national airspace system: modeling the sense and avoid challenge. In *Proceedings of the 2009 AIAA Infotech at Aerospace Conference*. Seattle, Washington.

Eva Crück and John Lygeros. 2007. Sense and avoid system for a MALE UAV. In *Proceedings of the 2007 AIAA Conference on Guidance, Navigation and Control*. Hilton Head, SC.

William Denman and César Muñoz. 2014. Automated Real Proving in PVS via MetiTarski. In *Proceedings of the 19th International Symposium on Formal Methods (FM 2014) (Lecture Notes in Computer Science)*, Cliff Jones, Pekka Pihlajasaari, and Jun Sun (Eds.), Vol. 8442. Springer, Singapore, 194–199.

Aaron Dutle, César Muñoz, Anthony Narkawicz, and Ricky Butler. 2015. Software Validation via Model Animation. In *Proceedings of the 9th International Conference on Tests & Proofs (TAP 2015) (Lecture Notes in Computer Science)*, Jasmin Blanchette and Nikolai Kosmatov (Eds.), Vol. 9154. Springer, L'Aquila, Italy, 92–108. DOI:http://dx.doi.org/10.1007/978-3-319-21215-9_6

FAA Sponsored Sense and Avoid Workshop. 2009. Sense and avoid (SAA) for Unmanned Aircraft Systems (UAS). (October 2009).

Jonathan Hammer. 1996. Horizontal miss distance filter system for suppressing false resolution alerts. (October 1996). U.S. Patent 5,566,074.

International Civil Aviation Organization (ICAO). 2005a. Annex 2 to the Convention on International Civil Aviation. (July 2005).

International Civil Aviation Organization (ICAO). 2005b. Annex 2 to the Convention on International Civil Aviation. (July 2005).

Darryl Jenkins and Bijan Vasigh. 2013. The economic impact of Unmanned Aircraft Systems integration in the United States. Economic report of the Association For Unmanned Vehicle Systems International (AUVSI). (March 2013).

Mariano Moscato, César Muñoz, and Andrew Smith. 2015. Affine Arithmetic and Applications to Real-Number Proving. In *Proceedings of the 6th International Conference on Interactive Theorem Proving (ITP 2015) (Lecture Notes in Computer Science)*, Christian Urban and Xingyuan Zhang (Eds.), Vol. 9236. Springer, Nanjing, China.

César Muñoz. 2003. *Rapid prototyping in PVS*. Contractor Report NASA/CR-2003-212418. NASA, Langley Research Center, Hampton VA 23681-2199, USA.

César Muñoz and Anthony Narkawicz. 2013. Formalization of a Representation of Bernstein Polynomials and Applications to Global Optimization. *Journal of Automated Reasoning* 51, 2 (August 2013), 151–196. DOI:http://dx.doi.org/10.1007/s10817-012-9256-3

César Muñoz, Anthony Narkawicz, and James Chamberlain. 2013. A TCAS-II Resolution Advisory Detection Algorithm. In *Proceedings of the AIAA Guidance Navigation, and Control Conference and Exhibit 2013*. Boston, Massachusetts.

César Muñoz, Anthony Narkawicz, James Chamberlain, María Consiglio, and Jason Upchurch. 2014. A Family of Well-Clear Boundary Models for the Integration of UAS in the NAS. In *Proceedings of the 14th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*. Atlanta, Georgia, USA.

César Muñoz, Anthony Narkawicz, George Hagen, Jason Upchurch, Aaron Dutle, and María Consiglio. 2015. DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems. In *Proceedings of the 34th Digital Avionics Systems Conference (DASC 2015)*. Prague, Czech Republic.

Anthony Narkawicz and César Muñoz. 2014. A Formally Verified Generic Branching Algorithm for Global Optimization. In *Proceedings of the 5th International Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2013) (Lecture Notes in Computer Science)*, Ernie Cohen and Andrey Rybalchenko (Eds.), Vol. 8164. Springer, Menlo Park, CA, US, 326–343.

Anthony Narkawicz, César Muñoz, and Aaron Dutle. 2015. Formally-Verified Decision Procedures for Univariate Polynomial Computation Based on Sturm's and Tarski's Theorems. *Journal of Automated Reasoning* 54, 4 (2015), 285–326. DOI:http://dx.doi.org/10.1007/s10817-015-9320-x

Sam Owre, John Rushby, and Natarajan Shankar. 1992. PVS: A Prototype Verification System. In *Proceedings of the 11th International Conference on Automated Deduction (Lecture Notes in Artificial Intelligence)*, Deepak Kapur (Ed.), Vol. 607. Springer-Verlag, 748–752.

RTCA SC-147. 2009. RTCA-DO-185B, Minimum operational performance standards for traffic alert and collision avoidance system II (TCAS II). (July 2009).

Erik Theunissen, Brandon Suarez, and Maarten Uijt de Haag. 2014. The impact of a quantitative specification of a Well Clear Boundary on pilot displays for self separation. In *Proceedings of the 2014 Integrated Communications, Navigation and Surveillance Conference (ICNS)*. IEEE, F2–1 – F2–11.

Filippo Tomasello and David Haddon. 2011. Detect and avoid for Unmanned Aircraft Systems in the total system approach. In *Proceedings of the 2011 Tyrrhenian International Workshop on Digital Communications-Enhanced Surveillance of Aircraft and Vehicles (TIWDC/ESAV)*. IEEE, Capri, Italy, 47–52.

Jason Upchurch, César Muñoz, Anthony Narkawicz, María Consiglio, and James Chamberlain. 2015. Characterizing the Effects of a Vertical Time Threshold for a Class of Well-Clear Definitions. In *Proceedings of the 11th USA/Europe Air Traffic Management R&D Seminar, ATM 2015*. Lisbon, Portugal.

US Code of Federal Regulations. 1967a. Title 14 Aeronautics and Space; Part 91 General operating and fight rules. (1967).

US Code of Federal Regulations. 1967b. Title 14 Aeronautics and Space; Part 91 General operating and fight rules. (1967).

Roland E Weibel, Matthew WM Edwards, and Caroline S Fernandes. 2011. Establishing a risk-based separation standard for unmanned aircraft self separation. In *Proceedings of the Ninth USA/Europe Air Traffic Management Research & Development Seminar*. Berlin, Germany.

Conference Reports

JORGE A PÉREZ, University of Groningen, The Netherlands
j.a.perez@rug.nl

This installment of the conference report column includes a report by Hugo Torres Vieira (IMT School for Advanced Studies Lucca, Italy) on DisCoTec 2016, a which took place in Heraklion, Greece on June 6-9, 2016.[1]

As Hugo describes in his report, DisCoTec 2016 consisted of three conferences (COORDINATION, DAIS, and FORTE) and two co-located workshops (ICE and RTPBD), all of them concerning recent research on distributed computing at large. The keynote lectures and contributed talks in these scientific events covered a wide range of topics of broad interest to the SIGLOG community, including: coordination models and languages, distributed applications and interoperable systems, and formal techniques for distributed objects, components and systems.

I am most grateful to Hugo for his availability and detailed report.

I look forward to receive your personal impressions and/or reports on conferences and meetings broadly related to SIGLOG. I will also be pleased to hearing your ideas and suggestions for future installments of the column.

[1] See http://2016.discotec.org.

DisCoTec 2016 Conference Report

Hugo Torres Vieira, IMT School for Advanced Studies Lucca, Italy

The 11th International Federated Conference on Distributed Computing Techniques (DisCoTec 2016) took place in Heraklion, Crete (Greece) from June 6 to June 9, 2016. DisCoTec has gathered for the past 11 years three main conferences, that previously where held separately, and hosts co-located workshops with an unifying theme on distributed computing, encompassing research originating in different communities. The main conferences are:

— The International Conference on Coordination Models and Languages (COORDINATION, 18th edition);
— The International Conference on Distributed Applications and Interoperable Systems (DAIS, 16th edition); and
— The International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE, 36th edition).

Co-located workshops were Interaction and Concurrency Experience (ICE, 9th edition) and the Final Public Workshop from LeanBigData and CoherentPaaS (RTPBD).

COORDINATION focuses on techniques that address interaction patterns and mechanisms at the base of collaboration in distributed systems. The 2016 edition included presentations targeting issues such as models for the Internet of Things, bridging models and implementations, techniques for ensuring security concerns and communication reliability, among others. The keynote speaker Vijay Saraswat (IBM TJ Watson Research Lab, USA) talked about *Logical and Imperative Calculi for Distributed Coordination*, namely Concurrent Constraint Programming (CCP) and X10. Vijay presented CCP as a foundation for concurrent, compositional computing, and X10 as a foundation for resilient, distributed stateful programming.

DAIS topics revolved around issues related to the Cloud, such as Data Processing, Computing and Resource management, among others. Tim Harris (Oracle Labs, UK) presented the keynote *What does the operating system ever do for me? System challenges in graph analytics*. Tim addressed some research efforts that aim at the exploitation of hardware specific information at the level of the distribution of the work load among threads, as well as the allocation of memory and threads, and how such fine-tuning can lead to the development of OS principles.

FORTE included sessions dedicated to verification and testing, formal methods in practice, session types, and expressiveness, among others. The keynote speaker Catuscia Palamidessi (INRIA, France) presented *Verifying Generalized Differential Privacy in Concurrent Systems*, addressing privacy protection in the context of shared data, namely regarding anonymity. Catuscia showed how k-anonymity falls short in ensur-

ing anonymity, and addressed other techniques such as randomization and noise insertion, together with issues specific to concurrent systems.

The DisCoTec conference series includes an award for best paper. The 2016 edition award went to *Enhanced Energy Efficiency with the Actor Model on Heterogeneous Architectures* by Yaroslav Hayduk, Anita Sobe and Pascal Felber. Other nominees were *On Sessions and Infinite Data* by Paula Severi, Luca Padovani, Emilio Tuosto and Mariangiola Dezani-Ciancaglini, and *Fencing Programs with Self-Invalidation and Self-Downgrade* by Parosh Aziz Abdulla, Mohamed Faouzi Atig, Stefanos Kaxiras, Carl Leonardsson, Alberto Ros and Yunyun Zhu.

The workshops took place after the main conferences, in the case of ICE with the duration of one and a half days while RTPBD lasted one day. The ICE workshop has been co-located with DisCoTec since 2010, featuring a distinguishing selection procedure that comprises interaction between authors and reviewers via a forum. The 2016 edition included a novelty regarding the disclosure of reviews of accepted papers. Invited talks at ICE where given by Uwe Nestmann (Technische Universität Berlin, Germany), who talked about *Dynamic Causality*, and by Alexandra Silva (University College London, UK), who talked about *Probabilistic NetKAT*.

The conference venue was the Aquila Atlantis Hotel, where also the reception on the first day took place, located in the terrace offering a great panoramic view of Heraklion's seaside. In the second day the social dinner was held in Plaka, a small village outside of Heraklion in an excellent location and with a very friendly atmosphere, which allowed DisCoTec participants to enjoy a great evening.

CFP

SIGLOG Monthly 184

DANIELA PETRIŞAN, Université Paris Diderot

SIGLOG Monthly 184
July 1, 2016

****
* Past issues of the newsletter are available at
http://lii.rwth-aachen.de/lics/newsletters/
* Instructions for submitting an announcement to the newsletter
can be found at
http://lii.rwth-aachen.de/lics/newsletters/inst.html
****

TABLE OF CONTENTS

THIRTY-FIRST ANNUAL ACM/IEEE SYMPOSIUM ON LOGIC IN COMPUTER SCIENCE (LICS 2016)

Call for Participation
5-8 July 2016, New York City, USA
http://lics.rwth-aachen.de/lics16/
https://regmaster4.com/2016conf/LICS16/register.php

* EVENT
LICS 2016 will be hosted in New York City during July 5-8, 2016.
This event also marks the thirtieth anniversary of LICS.

* AFFILIATED WORKSHOPS
Logic Mentoring Workshop
LSB: 6th Workshop on Logic and Systems Biology
NLCS: 4th Workshop on Natural Language and Computer Science.
SR: 4th International Workshop on Strategic Reasoning.
LOLA: Syntax and Semantics of Low-Level Languages.

* ACCEPTED PAPERS
http://lics.rwth-aachen.de/lics16/accepted.html

* IMPORTANT DATES
July 5-8, 2016 - Conference
July 9-10, 2016 - Workshops

THE 2016 ALONZO CHURCH AWARD
FOR OUTSTANDING CONTRIBUTIONS TO LOGIC AND COMPUTATION

http://siglog.hosting.acm.org/wp-content/uploads/2016/05/church16.pdf

* The 2016 Alonzo Church Award for Outstanding Contributions to Logic and Computation is given to Rajeev Alur and David Dill for their invention of timed automata, a decidable model of real-time systems, which combines a novel, elegant, deep theory with widespread practical impact.
Rajeev Alur and David Dill: A theory of timed automata.
Theoretical Computer Science 126(2):183-235, 1994.

* Alur and Dill will receive the award at the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), which will be held on July 5-8, 2016, at Columbia University, New York City, USA.

* The Award was established in 2015 by the ACM Special Interest Group for Logic and Computation (SIGLOG), the European Association for Theoretical Computer Science (EATCS), the European Association for Computer Science Logic (EACSL), and the Kurt Goedel Society (KGS). It recognises an outstanding contribution represented by a paper or small group of papers within the past 25 years; this is the first such award. The Award Committee consisted of Catuscia Palamidessi, Gordon Plotkin, Wolfgang Thomas, and Moshe Vardi (chair).

2016 GOEDEL PRIZE

http://eatcs.org/index.php/component/content/article/1-news/2280-2016-godel-prize-

* The 2016 Goedel Prize is awarded to Stephen Brookes and Peter W. O'Hearn for their invention of Concurrent Separation Logic, as described in the following two papers:

- S. Brookes, A Semantics for Concurrent Separation Logic.
  Theoretical Computer Science 375(1-3): 227-270 (2007)
- P. W. O'Hearn, Resources, Concurrency, and Local Reasoning.
  Theoretical Computer Science 375(1-3): 271-307 (2007)

Stephen Brookes and Peter O'Hearn will receive the 2016 Goedel Prize at the 43rd International Colloquium on Automata, Languages and Programming (ICALP 2016), 12-15 July 2016, in Rome, Italy.

ACM SIGLOG ANNOUNCEMENT

http://siglog.acm.org

* The ACM has recently chartered a Special Interest Group on Logic and Computation (ACM SIGLOG).
* We are pleased to announce the 2016 ACM SIGLOG election results for the term of 1 July 2016 - 30 June 2019. The SIGLOG Chair is Prakash Panangaden and the other officers are Luke Ong (vice-Chair), Amy Felty (Treasurer) and Alexandra Silva (Secretary).
* The ACM-IEEE Symposium on Logic in Computer Science is the flagship conference of SIGLOG. SIGLOG will also actively seek association agreements with other conferences in the field. A SIGLOG newsletter (SIGLOG News) is also published quarterly in an electronic format with community news, technical columns, members' feedback, conference reports, book reviews and other items of interest to the community.
* One can join SIGLOG by visiting
  https://campus.acm.org/public/qj/gensigqj/siglist/gensigqj˙siglist.cfm
  It is possible to join SIGLOG without joining ACM (the SIGLOG membership fee is $25 and $15 for students).

DATES

* RULEML AND DECISIONCAMP 2016
  Call for Participation, Call for Papers
  Stony Brook, NY, 6-9 July, 2016
  http://2016.ruleml.org
* SSBSS 2016
  Call for Participation
  8-14 July 2016, Volterra (Pisa), Tuscany, Italy
* FSTTCS 2016
  Call for Papers
  December 13--15, 2016, Chennai, India
  http://www.fsttcs.org/
  Paper submission: July 15, 2016
* ESSLLI 2017
  Call for Course and Workshop Proposals
  17-28 July, 2017, Toulouse, France
  https://www.irit.fr/esslli2017
  15 July 2016: Proposal submission deadline
* DEDUCTION MEETING/DEDUKTIONSTREFFEN 2016
  Call for Papers
  September 26, 2016, Alpen Adria University, Klagenfurt, Austria
  Hosted by KI 2016.

https://fg-dedsys.gi.de/dt2016.html
Early bird submission: July 15
Standard submission: August 15

* CAV 2016
Call for participation
July 17-23 2016, Hyatt Regency Toronto, Ontario, Canada
http://i-cav.org/2016/

* ILP2016 - Call For Papers
Call For Papers
September 4th - 6th, 2016, London, UK
http://ilp16.doc.ic.ac.uk
Short Paper submission: 24 July 2016

* CRV 2016
Call for Participation
https://rv2016.imag.fr/?page˙id=188
September 2016, Madrid, Spain

* DARe-16 at ECAI
Call for Participation
Date: 29 August 2016
The Hague, Netherlands
http://dare2016.yolasite.com

* ARVI COST 2016
Call for Participation
September 23-25, Madrid, Spain
https://rv2016.imag.fr/?page˙id=128

* SSS 2016
Call for Papers
Lyon, France, November 7-10, 2016
http://graal.ens-lyon.fr/ SSS16/
Abstract Submission: July 17
Paper Submission: July 24

* ICDT 2017
Call for papers
27-31 March, 2017, Venice, Italy
http://www.cs.ox.ac.uk/people/michael.benedikt/icdt17.html
Paper deadine (2nd submission cycle): September 18, 2016

RULEML AND DECISIONCAMP 2016
Call for Participation
Dates: 6-9 July, 2016
Location: New Computer Science Department Building, room 120,
Stony Brook University, New York, USA
http://2016.ruleml.org

* Invited presentations by: Richard Waldinger (2016 Herbrand Award winner), Charles Forgy (inventor of the RETE algorithm, Bruce Silver (book author on DMN Method and Style), Michael Kifer, Theresa Swift and Benjamin Grosof.
* Submission Deadline (Papers/Demos/Posters) for all Open Calls: June 5th, 2016
* The RuleML 2016 Challenge is one of the highlights of the RuleML 2016 Symposium, providing a friendly competition environment for

innovative rule-oriented systems and applications, aimed at both research and industry participants.
http://2016.ruleml.org/challenge
* Posters are called for about late-breaking rule topics, e.g. domain-specific applications of rule systems. Posters should be accompanied by high-quality, original 1-4 page extended abstracts, which should be submitted at:
https://easychair.org/conferences/?conf=ruleml2016 ("10th International Rule Challenge" track).
* Both PhD students and mentors are called for:
http://2016.ruleml.org/doctoral-consortium
* Industry Track will include a panel about use of rules and rule technologies in business cases:
http://2016.ruleml.org/industry-track
* Accepted papers
http://2016.ruleml.org/list-of-accepted-papers
* Accomodation
http://2016.ruleml.org/venue-and-accommodation

3RD INT. SYNTHETIC AND SYSTEMS BIOLOGY SUMMER SCHOOL (SSBSS 2016)
Call for Participation
8-14 July 2016, Volterra (Pisa), Tuscany, Italy
* The 3rd International Synthetic and Systems Biology Summer School is a great opportunity to exchange ideas and information with colleagues and peers from around the world and discover the latest trends and new exciting results in Synthetic and Systems Biology.
* Further details: ssbss.school@gmail.com
http://www.taosciences.it/ssbss/
* Previous Editions:
SSBSS 2015 http://www.taosciences.it/ssbss2015/
SSBSS 2014 http://www.taosciences.it/ssbss2014/

36TH FOUNDATIONS OF SOFTWARE TECHNOLOGY AND THEORETICAL COMPUTER SCIENCE (FSTTCS 2016)
Call for Papers
December 13--15, 2016, Chennai, India
http://www.fsttcs.org/
* IARCS, the Indian Association for Research in Computing Science, announces the 36th Foundations of Software Technology and Theoretical Computer Science (FSTTCS) conference at Chennai Mathematical Institute, Chennai, India. The FSTTCS conference is a forum for presenting original results in foundational aspects of Computer Science and Software Technology.
* Important Dates
Paper submission: July 15, 2016
Notification to authors: September 15, 2016
Camera-ready version due: October 15, 2016
Conference: December 13-15, 2016
* Scope
- Algorithms & data structures, Automata & formal languages

- Algorithmic graph theory & combinatorics
- Models of concurrent, distributed mobile systems
- Approximation algorithms, Cryptography & security
- Combinatorial optimization, Game theory & mechanism design
- Communication complexity, Logic in computer science
- Computational complexity, Model theory & temporal logics
- Computational geometry, Computational biology
- Computational learning theory
- Models of timed, reactive, hybrid & stochastics systems
- Parallel, distributed & online algorithms
- Principles & semantics of programming languages
- Parameterized complexity, Program analysis & transformation
- Proof complexity, Specification, verification & synthesis
- Quantum computing, Theorem proving & model checking
- Randomness in computing, Type systems, type theory & calculi

* Further details:
http://www.fsttcs.org/

29TH EUROPEAN SUMMER SCHOOL IN LOGIC, LANGUAGE AND INFORMATION (ESSLLI 2017)

Call for Course and Workshop Proposals
17-28 July, 2017, Toulouse, France
https://www.irit.fr/esslli2017
Call for Course and Workshop Proposals
17-28 July, 2017, Toulouse, France
https://www.irit.fr/esslli2017

* IMPORTANT DATES (with extended deadline)
15 July 2016: Proposal submission deadline
30 September 2016: Notification

* SUBMISSION PORTAL
Please submit your proposals here:
https://easychair.org/conferences/?conf=3Desslli2017

* TOPICS AND FORMAT Proposals for courses and workshops at ESSLLI 2017 are invited in all areas of Logic, Linguistics and Computing Sciences. Cross-disciplinary and innovative topics are particularly encouraged. Each course and workshop will consist of five 90 minute sessions, offered daily (Monday-Friday) in a single week. The EACSL will sponsor for one course or workshop in the areas of Logic and Computation.

* PROGRAMME COMMITTEE
Chair:
Shravan Vasishth, Universitaet Potsdam
Local co-chair:
Philippe Balbiani, IRIT, Toulouse
Language and Computation:
Sebastian Pado, Universitaet Stuttgart
Mehrnoosh Sadrzadeh, University of London
Language and Logic:
Denis Bonnay, l'Universite Paris Ouest
Jessica Rett, UCLA
Logic and Computation:

Tomer Kotek, Technische Universitaet Wien
Anna Zamansky, University of Haifa
* FURTHER INFORMATION:
Please send any queries you may have to vasishth dot shravan at gmail dot com.

30th DEDUCTION MEETING/DEDUKTIONSTREFFEN 2016
Call for Papers
September 26, 2016, Alpen Adria University, Klagenfurt, Austria
Hosted by KI 2016.
https://fg-dedsys.gi.de/dt2016.html
* The annual meeting Deduktionstreffen is the prime activity of the Interest Group for Deduction Systems (FGDedSys) of the German Informatics Society.
It is a meeting with a familiar, friendly atmosphere, where all members and friends of the German deduction community are invited to present, discuss and share their latest research results and ideas in an informal setting.
* A particular focus of the Deduktionstreffen is on young researchers and students, who are particularly encouraged to present their ongoing research projects to a wider audience. Another goal of the meeting is stimulate networking effects and to foster collaborative research projects.
* Invited Speakers:
- Armin Biere
- Cezary Kaliszyk
* We welcome contributions on all theoretical, experimental and application aspects of deduction. Accepted abstracts will be presented as 5min teaser talks followed by a poster presentation.
* Important dates:
- Early bird submission: July 15
- Standard submission: August 15

28TH INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION (CAV 2016)
Call for Participation
July 17-23, 2016, Hyatt Regency Toronto, Ontario, Canada
http://i-cav.org/2016/
* Highlights
- 46 regular papers, 12 tool papers
- Four invited talks, four invited tutorials
- Talk by a winner of the 2016 CAV award
* Registration deadline
- Early registration deadline: June 10, 2016
- Register at https://regmaster4.com/2016conf/CAV16/register.php
* Conference program
Available at http://i-cav.org/2016/program/
* Invited talks
- Gilles Barthe (IMDEA Software Institute)
Computer-aided Cryptography
- Gerwin Klein (NICTA and University of New South Wales)

Scaling Up - From Trustworthy seL4 to Trustworthy Systems
- Moshe Vardi (Rice University)
Constrained Sampling and Counting
- A winner of the 2016 CAV Award
(To be announced at the conference)
* Invited tutorials
- Parosh Abdulla (Uppsala University)
Small Models in Parameterized Verification
- Vitaly Chipounov (EPFL)
The S2E Platform: Design, Implementation, and Applications
- Paulo Tabuada (UCLA)
Synthesizing Robust Cyber-Physical Systems
- Martin Vechev and Pavol Bielek (ETH)
Machine Learning for Programs
* Associated workshops
- NSV: 9th International Workshop on Numerical Software Verification
http://nsv2016.pages.ist.ac.at/
- VSTTE: 8th Working Conference on Verified Software: Theories,
Tools, and Experiments
http://www.cs.toronto.edu/~chechik/vstte16/
- SYNT: 5th Workshop on Synthesis
http://formal.epfl.ch/synt/2016/
- (EC)2: 9th International Workshop on Exploiting Concurrency
Efficiently and Correctly
http://ecee.colorado.edu/pavol/ec2-2016/
- HCCV: Workshop on High-Consequence Control Verification
http://www.sandia.gov/hccv/
- VMW: Verification Mentoring Workshop
http://i-cav.org/2016/vmw/
* Conference chairs
Swarat Chaudhuri (Rice University)
Azadeh Farzan (University of Toronto)

THE 26TH INTERNATIONAL CONFERENCE ON INDUCTIVE LOGIC PROGRAMMING (ILP2016)
Call For Papers
September 4th - 6th, 2016, London, UK
http://ilp16.doc.ic.ac.uk
* AIMS: The ILP conference series is the premier international forum for learning from structured relational data. Originally focusing on the induction of logic programs, over the years it has expanded its research horizon significantly and welcomes contributions to all aspects of learning in logic, multi-relational data mining, statistical relational learning, graph and tree mining, learning in other (non-propositional) logic-based knowledge representation frameworks, exploring intersections to statistical learning and other probabilistic approaches.
* TOPICS OF INTEREST include:
- Theoretical aspects: logical-foundations of learning;
- computational/statistical learning theory; specialisation and
- generalisation; probabilistic logic-based learning; graph and tree

- mining. Representation and languages for learning: logic
- programming; Datalog;first-order logic; description logic and
- ontologies; higher-order logic; Answer Set Programming;
- probabilistic logic languages; constraint logic programming;
- knowledge graphs. Algorithms and systems: learning with
- (semi-)structured data; (semi-)supervised and unsupervised
- relational learning; relational reinforcement learning; predicate
- invention; propositionalisation approaches; multi-instance learning;
- learning in the presence of uncertainty; meta-level learning.
- Applications of learning in: art; bioinformatics; systems biology;
- games; medical informatics; robotics; natural language processing;
- web-mining; software engineering; modelling and adaptation of
- control systems; socio-technical systems.

In addition to the above topics, ILP 2016 is also encouraging contributions in the areas of cognitive technologies, knowledge acquisition from big data, the cloud and crowd sourced data, deep relational learning, as well as contributions on the application of any of these solutions to real world problems. The conference will host keynote talks from both industry and academia and will run the first International ILP Competition.

* Submission guidlines: please see the conference website
* IMPORTANT DATES:
  - Abstract registration: 7 May 2016
  - Long paper submission: 13 May 2016
  - Long Paper notification: 26 June 2016
  - Short Paper submission: 24 July 2016
  - Short Paper notification: 28 July 2016
* We expect there will be a special issue of the Machine Learning Journal following the conference, which will be open for everyone. This special issue will welcome conference submissions from all three categories, which should be significantly revised and extended, to meet the MLJ criteria, and will be re-reviewed by PC members.
* CONFERENCE AND PROGRAM CO-CHAIRS:
  - Alessandra Russo, Imperial College London UK
  - James Cussens, University of York, UK
* ILP COMPETITION CHAIR;
  Mark Law, Imperial College London, UK
* PUBLICITY CHAIR:
  Krysia Broda, Imperial College London, UK
* ASSOCIATED EVENT:
  3rd International Workshop on Probabilistic Logic Programming

THE 3RD INTERNATIONAL COMPETITION ON RUNTIME VERIFICATION (CRV 2016)
Call for Participation
https://rv2016.imag.fr/?page˙id=188
September 2016, Madrid, Spain

* The 3rd International Competition on Runtime Verification In Association with COST Action "Runtime Verification beyond Monitoring" held with RV 2016, September 23-30 2016, Madrid, Spain

* The main goal of CRV 2016 is to compare tools for runtime verification. We invite and encourage the participation with benchmarks and tools for the competition. The competition will consist of three main tracks based on what is being monitored:
 - Track on monitoring Java programs (online monitoring)
 - Track on monitoring C programs (online monitoring)
 - Subtrack on Generic Specifications (e.g. in LTL)
 - Subtrack on Implicit Specifications (e.g. memory safety)
 - Track on monitoring of traces (offline monitoring)
* The general organisation of the competition is described in the rules document found at http://crv.liflab.ca/CRV2016.pdf.
* To register please fill in the form at http://goo.gl/forms/kWxFFfFCvZ.
* Please direct any enquiries to the competition co-organizers (crv2016@crv.liflab.ca):
 - Ylies Falcone (Univ. Grenoble Alpes, Inria, France),
 - Sylvain Halle (Universite du Quebec a Chicoutimi, Canada),
 - Giles Reger (University of Manchester, Manchester, UK).
* Expected Important Dates
 May 9th Registration Opens
 May 29th Benchmark Submission Deadline
 June 5th Registration Closes
 June 5-12th Clarifications Phase
 June 19th Benchmarks Announced
 July 10th Monitor Submission Deadline
 August 1st Notifications
 At RV 2016 Presentation of Results

THE THIRD INTERNATIONAL WORKSHOP ON ”DEFEASIBLE AND AMPLIATIVE REASONING” (DARE 2016)
 Call for Participation
 29 August 2016, The Hague, Netherlands
 http://dare2016.yolasite.com
* The workshop is held at the European Conference on Artificial Intelligence (ECAI 2016)
* DARe welcomes contributions on all aspects of defeasible and ampliative reasoning such as (but not limited to):
 - Abductive and inductive reasoning
 - Explanation finding, diagnosis and causal reasoning
 - Inconsistency handling and exception-tolerant reasoning
 - Decision-making under uncertainty and incomplete information
 - Default reasoning, non-monotonic reasoning, non-monotonic logics, conditional logics
 - Specific instances and variations of ampliative and defeasible reasoning
 - Probabilistic and statistical approaches to reasoning
 - Vagueness, rough sets, granularity and fuzzy-logics
 - Philosophical foundations of defeasibility
 - Empirical studies of reasoning
 - Relationship with cognition and language

- Contextual reasoning
- Preference-based reasoning
- Analogical reasoning
- Similarity-based reasoning
- Belief dynamics and merging
- Argumentation theory, negotiation and conflict resolution
- Heuristic and approximate reasoning
- Defeasible normative systems
- Reasoning about actions and change
- Reasoning about knowledge and belief, epistemic and doxastic logics
- Ampliative and defeasible temporal and spatial reasoning
- Computational aspects of reasoning with uncertainty
- Implementations and systems
- Applications of uncertainty in reasoning

* IMPORTANT DATES
- Submission deadline: 16 June 2016
- Notification: 28 June 2016
- Camera ready: 17 July 2016
- Early registration: 5 July 2016
- Late registration: [TBA]
- Workshop date: 30 August 2016

* WORKSHOP CO-CHAIRS
- Richard Booth (Cardiff University, United Kingdom)
- Giovanni Casini (Universite du Luxembourg)
- Szymon Klarman (Brunel University London, United Kingdom)
- Gilles Richard (Universite Paul Sabatier, France)
- Ivan Varzinczak (CRIL, Universite d'Artois, France)

* FURTHER INFORMATION. Please visit the workshop website (http://dare2016.yolasite.com) for further information and regular updates.

FOURTH CONFRENCE ON HIGHLIGHTS OF LOGIC, GAMES AND AUTOMATA (HIGHLIGHTS 2016)

Call for Presentations
September 6-9, 2016, Brussels, Belgium
http://highlights-conference.org

* HIGHLIGHTS 2016 is the fourth conference on Highlights of Logic, Games and Automata which aims at integrating the community working in these fields. Papers from these areas are dispersed across many conferences, which makes them difficult to follow. A visit to Highlights conference should offer a wide picture of the latest research in the field and a chance to meet everybody in the community, not just those who happen to publish in one particular proceedings volume. We encourage you to attend and present your best work, be it already published or not, at the Highlights conference.
* Representative areas include, but are not restricted to: logic and finite model theory, automata theory, games for logic and verification.
* You submit a proposal for a presentation, not a paper. Hence, submissions should have a single author, who is the speaker. Since we expect you to present your favorite result of the year, there

should be at most one submission per speaker. The abstract, of 1-2 pages, may include a list of coauthors. There are no formal proceedings and we encourage submission of work presented elsewhere. Submissions are possible through https://easychair.org/conferences/?conf=3Dhighlights2016.
* The program will further offer three keynotes by Meena Mahajan (Chennai), Andreas Maletti (Stuttgart), and Marc Zeitoun (Bordeaux), two invited sessions, organised by Slawomir Lasota (Warsaw) and Anca Muscholl (Bordeaux), and two tutorials by Benedikt Bollig (Cachan) and Antonin Kucera (Prague).
* Important dates:
Submission deadline: June 3, 2016
Registration possible until August 7, 2016

36TH ACM SIGMOD-SIGACT-SIGAI SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS (PODS 2017)
Call for Papers (1st submission cycle)
May 14-19, 2017, Raleigh, North Carolina, USA
http://www.sigmod2017.org
* PODS has two rounds of submissions (see dates below). Note that the 1st round is moved earlier compared with the previous editions of PODS. This is to better synchronize with the two deadlines of the ICDT conference (see http://www.cs.ox.ac.uk/people/michael.benedikt/icdt17.html)
* TOPICS that fit the interests of the symposium include the following:
- design, semantics, query languages
- data models, data structures, algorithms for data management
- concurrency and recovery, distributed and parallel databases, cloud computing
- model theory, logics, algebras, computational complexity
- graph databases and (semantic) Web data
- data mining, information extraction, search
- data streams
- data-centric (business) process management, workflows, web services
- incompleteness, inconsistency, uncertainty in data management
- data and knowledge integration and exchange, data provenance, views and data warehouses, metadata management
- domain-specific databases (multi-media, scientific, spatial, temporal, text)
- deductive databases
- data privacy and security
* PROGRAM CHAIR
Floris Geerts (University of Antwerp, BE)
* IMPORTANT DATES
- Dates for first submission cycle:
June 12, 2016, 11:59pm PST: Abstract submission
June 19, 2016, 11:59pm PST: Paper submission
August 28, 2016, 11:59pm PST: Accept/Reject/Revise notification
September 25, 2016, 11:59pm PST: Revision deadline
October 30, 2016, 11:59pm PST: Accept/Reject notification (Revisions)

- Dates for second submission cycle:
  December 11, 2016, 11:59pm PST: Abstract submission
  December 18, 2016, 11:59pm PST: Paper submission
  February 26, 2017, 11:59pm PST: Accept/Reject notification
  March 19, 2017, 11:59pm PST: Camera-ready deadline

* AWARDS
- Best Paper Award: An award will be given to the best submission, as judged by the program committee.
- Best Student Paper Award: There will also be an award for the best submission, as judged by the program committee, written by a student or exclusively by students.

ARVI COST SUMMER SCHOOL ON RUNTIME VERIFICATION (ARVI COST 2016)
Call for Participation
September 23-25, Madrid, Spain
https://rv2016.imag.fr/?page˙id=128

* The first edition of the ARVI COST Summer School on Runtime Verification: Branches of Practical Topics Rooted in Theory will be co-located with the 16th International Conference on Runtime Verification in Madrid and organised over three days with a series of lectures from international experts in the field.

* TOPICS:
- The basics of runtime verification
- Instrumentation techniques
- Specification languages
- Monitor parametrisation
- Monitoring concurrency errors
- Performance issues of monitors
- Combination of static and dynamic analysis
- Monitoring of distributed systems
- Time-triggered monitoring

* Confirmed Speakers:
- Prof. Wolfgang Ahrendt - Chalmers University of Technology and University of Gothenburg (Sweden)
- Prof. Ezio Bartocci - University of Vienna (Austria).
- Prof. Borzoo Bonakdarpour - University of MacMaster (Canada).
- Dr. Marius Bozga - CNRS - Verimag (France).
- Dr. Christian Colombo - University of Malta (Malta).
- Dr. Ylies Falcone - University of Grenoble (France).
- Dr. Adrian Francalanza - University of Malta (Malta).
- Dr. Klaus Havelund - NASA Jet Propulsion Laboratory (USA).
- Prof. Martin Leucker - University of Lubeck (Germany).
- Prof. Joao Lourenco - Universidade Nova de Lisboa (Portugal).
- Prof. Dejan Nikovic Technical University of Vienna (Austria).
- Prof. Gordon Pace - University of Malta (Malta).
- Dr. Giles Reger - University of Manchester (UK).

* Application Procedure and Important Dates:
Deadline for Applications: July 15, 2016.
Response to Applicants: July 20, 2016.
Online Registration and Fee payment: July 29, 2016.

* More details can be found at: https://rv2016.imag.fr/?page˙id=128.

Enquiries can be sent to Ylies.Falcone@imag.fr or
Christian.Colombo@um.edu.mt.

18TH INTERNATIONAL SYMPOSIUM ON STABILIZATION, SAFETY, AND SECURITY OF DISTRIBUTED SYSTEMS (SSS 2016)

Call for Papers
Lyon, France, November 7-10, 2016
http://graal.ens-lyon.fr/ SSS16/

* The Symposium on Stabilization, Safety, and Security of Distributed Systems is an international forum for researchers and practitioners working on the design and development of distributed systems that guarantee specific desired properties despite adversity, or that are able to restore the desired properties following adversarial perturbations in the computing medium building on the principles of self-stabilization. The symposium encourages the submission of original contributions spanning fundamental research and practical applications within its scope, covered by the three symposium tracks.

* TOPICS AND TRACKS

* Track 1: Self-* and Autonomic Computing
- Self-stabilizing systems
- Self-organizing, self-managing, and self-configuring systems
- Self-optimizing and self-healing systems
- Self-protecting and self-repairing systems
- Autonomic cloud computing
- Autonomous vehicles

* Track 2: Foundations
- Theory of self-stabilization
- Distributed algorithms
- Fault-Tolerant distributed systems
- Formal methods, validation, verification, and synthesis
- Safety and security

* Track 3: Networks, Multi-Agent Systems, and Mobility
- Self-Stabilizing networks
- Peer-to-peer networks
- Sensor networks, MANETs, and wireless mesh networks
- Large and extreme scale systems
- Distributed robot systems
- Cooperating multi-agent distributed systems
- Dynamic systems and networks
- Overlay networks
- Social networks
- High-Speed networks

* IMPORTANT DATES
- Abstract Submission: July 17
- Paper Submission: July 24
- Notification: September 12
- Camera Ready Submission: September 23
- Authors Registration: September 23
- Early Registration: October 7
- Conference: November 7-10

* Keynote Speakers
Hagit Attiya (Technion, Israel)
Joseph Halpern (Cornell University, USA)
TBA
* General Chair:
Franck Petit (Universite Pierre et Marie Curie, LIP6, France)
* Program Committee Chair:
Borzoo Bonakdarpour (McMaster University, Canada)
* Program Chairs
- Track 1: Self-* and Autonomic Computing
Stephane Devismes, co-chair (University of Grenoble, France)
Manish Parashar, co-chair (Rutgers University, USA)
- Track 2: Foundations
Vijay Garg, co-chair (University of Texas - Austin, USA)
Sergio Rajsbaum, co-chair (UNAM, Mexico)
- Track 3: Networks, Multi-Agent Systems, and Mobility
Yvonne-Anne Pignolet, co-chair (ABB Corporate Research, Switzerland)
Roger Wattenhofer, co-chair (ETH-Zurich, Switzerland)
* The program committee will select two papers for the best paper and best student paper awards.

THE 20TH INTERNATIONAL CONFERENCE ON DATABASE THEORY (ICDT 2017)
Call for papers
27-31 March, 2017, Venice, Italy
http://www.cs.ox.ac.uk/people/michael.benedikt/icdt17.html
* The series of ICDT conferences (http://icdt.tu-dortmund.de/) provides an international forum for the communication of research advances on the theoretical foundations of database systems.
* ICDT has made significant changes to its submission dates, in co-ordination with its sibling conference PODS. There are now two submission cycles, with the first providing the possibility of revision.
* First submission cycle:
Abstract deadline: March 18, 2016
Full paper submission deadline: March 25, 2016
Accept/Reject/Revise Notification: May 29, 2016
* Second submission cycle:
Abstract deadline: September 11, 2016
Full paper submission deadline: September 18, 2016
Notification: November 27, 2016
* Examples of relevant topics are: concurrency and recovery, distributed and parallel databases, cloud computing, connections between databases and knowledge representation, graph databases and (semantic) Web data, data mining, information extraction, search, data streams, data-centric (business) process management, workflows, web services, incompleteness, inconsistency, uncertainty in databases, data and knowledge integration and exchange, data provenance, views and data warehouses, metadata management, domain-specific databases (multi-media, scientific, spatial, temporal, text), deductive databases, data privacy and security, database aspects of machine learning, model theory, logics,

algebras, computational complexity, design, semantics, query languages, data models, data structures, algorithms for data management.

PHD STUDENTSHIP IN SEMANTICS AND VERIFICATION OF HETEROGENEOUS PROGRAMS

* Applications are invited for a fully-funded PhD studentship within the Theory Group at Queen Mary University of London, as part of a project which aims to develop a unified semantic framework for heterogeneous software systems and apply it to compositional software compilation and verification. Cloud computing and heterogeneous computing are widely acknowledged to dominate the software landscape in the foreseeable future. The recent work on System-Level Games provides a semantic framework for modelling low-level code interactions involving resources shared between a program and its environment. This project will apply the framework for deriving compositional analysis techniques for the compilation and verification of heterogeneous programs.
* All nationalities are eligible to apply for this studentship, which will start in October 2016. The studentship is for three years, and covers student fees as well as a tax-free stipend of 16,057 per annum. Candidates must have a 2:1 degree or equivalent, and/or a good MSc Degree, in Computer Science or a related discipline. The ideal candidate should be creative and motivated in the studying of semantics and verification of programming languages. Good coding skills will be an advantage, and applicants will have at least good knowledge of programming languages such as C/C++, Java, Python, OCaml. Analytical and good communication skills are also welcome.
* The PhD supervisor will be Dr Nikos Tzevelekos. The project will be based in the School of Electronic Engineering and Computer Science (EECS), and the student will join a world-leading centre for research on logical methods for reasoning about computer systems in the Theory Group (http://theory.eecs.qmul.ac.uk/). The position will be integrated in the EPSRC project ”System-Level Game Semantics: A unifying framework for composing systems”, which is in collaboration with the University of Birmingham. Informal enquiries about the studentship can be made by email to Dr Tzevelekos (nikos.tzevelekos@qmul.ac.uk).
* To apply, please follow the on-line process at www.qmul.ac.uk/postgraduate/applyresearchdegrees/ click on the list of Research Degree Subjects, select ”Computer Science”, and follow the instructions on the right-hand side of the web page. Please note that instead of the Research Proposal we request a Statement of Research Interests. Your statement should answer two questions: (i) Why are you interested in the topic described above? (ii) What relevant experience do you have? Your statement should be brief: no more than 500 words or one side of A4 paper. In addition we would also like you to send a sample of your written work (e.g. excerpt of final year dissertation or published academic paper). More details can be found at: http://www.eecs.qmul.ac.uk/phd/how-to-apply
* The closing date for the applications is 24/07/2016. Interviews are expected to take place the week of 25 July 2016.

3-YEAR POSTDOC POSITION AT HASSELT UNIVERSITY

* We have a vacancy for a 3-year postdoc position at Hasselt University. The salary is very good and it comes with social security, health insurance, what have you. The topic is very flexible as long as it has to do with finite model theory, expressive power of database query languages, in particular query languages for novel data models such as JSON or graph data, tractable fragments of higher-order logic is also a theme that fits.
* The position needs to be filled by 1 January 2017 at the latest.
* The research group on Databases and Theoretical Computer Science at Hasselt University is a leading group in the theoretical foundations of data management. Professors are Marc Gyssens, Bart Kuijpers, Frank Neven, and Jan Van den Bussche
* Please email Jan Van den Bussche (jan.vandenbussche@uhasselt.be) if you are interested.
* http://alpha.uhasselt.be/jan.vandenbussche

join today!

SIGLOG & ACM

siglog.acm.org www.acm.org

The **Special Interest Group on Logic and Computation** is the premier international community for the advancement of logic and computation, and formal methods in computer science, broadly defined.

The **Association for Computing Machinery** (ACM) is an educational and scientific computing society which works to advance computing as a science and a profession. Benefits include subscriptions to *Communications of the ACM*, *MemberNet*, *TechNews* and *CareerNews*, full and unlimited access to online courses and books, discounts on conferences and the option to subscribe to the ACM Digital Library.

- ❑ SIGLOG (ACM Member) ..... $ 25
- ❑ SIGLOG (ACM Student Member & Non-ACM Student Member) ..... $ 15
- ❑ SIGLOG (Non-ACM Member) ..... $ 25
- ❑ ACM Professional Membership ($99) & SIGLOG ($25) $124
- ❑ ACM Professional Membership ($99) & SIGLOG ($25) & ACM Digital Library ($99) $223
- ❑ ACM Student Membership ($19) & SIGLOG ($15) ..... $ 34

payment information

Name _____

ACM Member # _____

Mailing Address _____

_____

City/State/Province _____

ZIP/Postal Code/Country _____

Email _____

Mobile Phone _____

Fax _____

Mailing List Restriction
ACM occasionally makes its mailing list available to computer-related organizations, educational institutions and sister societies. All email addresses remain strictly confidential. Check one of the following if you wish to restrict the use of your name:

- ❑ ACM announcements only
- ❑ ACM and other sister society announcements
- ❑ ACM subscription and renewal notices only

Credit Card Type: ❑ AMEX ❑ VISA ❑ MC

Credit Card # _____

Exp. Date _____

Signature _____

Make check or money order payable to ACM, Inc

ACM accepts U.S. dollars or equivalent in foreign currency. Prices include surface delivery charge. Expedited Air Service, which is a partial air freight delivery service, is available outside North America. Contact ACM for more information.

Questions? Contact:
ACM Headquarters
2 Penn Plaza, Suite 701
New York, NY 10121-0701
voice: 212-626-0500
fax: 212-944-1318
email: acmhelp@acm.org

Remit to:
ACM
General Post Office
P.O. Box 30777
New York, NY 10087-0777

SIGAPP

Association for Computing Machinery

www.acm.org/joinsigs

Advancing Computing as a Science & Profession