Volume 1, Number 2
October 2014

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

SIGLOG news

TABLE OF CONTENTS

Association for Computing Machinery

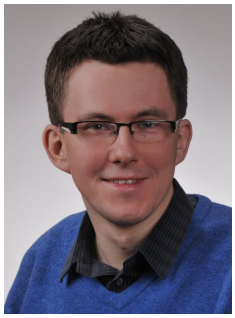
Advancing Computing as a Science & Profession

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



Welcome to the second issue! I would like to express warmest thanks to the Column Editors and the first external contributors.

— In his Automata Column, *Mikołaj Bojańczyk* takes us through a selection of open problems in automata and logic.

— *Libor Barto* surveys the Constraint Satisfaction Problem in Neil Immerman's Complexity Column.

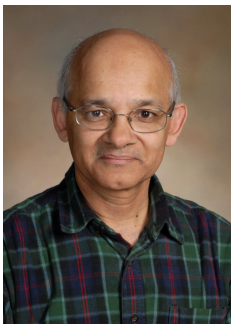
— And, in Mike Mislove's Column on Semantics, *Achim Jung* shares his perspective on teaching denotational semantics.

Do not hesitate to get in touch with the Column Editors if you would be interested in contributing to a column. Other kinds of material related to Logic and Computation are also most welcome, such as book reviews and reports from recent conferences. Please email them to `editor@siglog.org`.

Enjoy the issue and consider entering the SIGLOG Logo competition (page 45)!

Andrzej Murawski
University of Warwick
SIGLOG News Editor

Chair's Letter



Wow! Perhaps not the most dignified way to start a letter, but that was my reaction to the Federated Logic Conferences (FLoC) held this summer in Vienna as part of the Vienna Summer of Logic. This was the largest assembly of logicians and logic-related related researchers ever. There were 12 major conferences and over a 100 workshops with an overall attendance of 2000 participants. Any doubts about the size and vitality of the community are laid to rest.

SIGLOG held two launch events to announce the formation of the new SIG and to recruit members. There was a lot of interest in SIGLOG and I am pleased to report that at last count there were over 150 members. My goal is to reach 200 by the year's end. I will be visiting India, China and Japan in the coming year and plan to make a serious effort to spread the news about SIGLOG in Asia.

I am pleased to report that the education committee has been formed and consists of Iliano Cervesato, Joao Marcos, Brigitte Pientka, R. Ramanujam, Nicole Schweikardt and Richard Zach. Iliano will be the chair. There was a very interesting panel organized by ASL on the role of logic in computer science education which was organized by Richard Zach which provided a forum for some lively exchanges. Both Joao Marcos and R. Ramanujam have announced events of interest to people interested in logic education. These will be taking place in Brazil and India respectively. It is great to have such a geographically diverse group.

The main initiative that I am working on with the EC and with our European partner organizations (EATCS, EACSL, Kurt Gödel Society) is the establishment of major awards for contributions to logic and computation. I am learning that not everything can be done instantly, but it is better to do things right the first time. I am still hopeful that these awards are not too far off.

This issue of the SIGLOG Newsletter features three columns on Automata, on Semantics and on Complexity. I am very grateful to Andrzej for the great job that he is doing to keep the issues coming out on schedule.

Prakash Panangaden
McGill University
ACM SIGLOG Chair

AUTOMATA COLUMN

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

Some Open Problems in Automata and Logic

The list in this paper is, of course, a personal selection of open problems that are connected to both automata and logic. The problems are listed in no particular order.

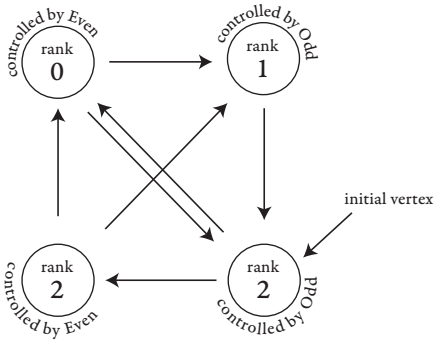
1. CAN PARITY GAMES BE SOLVED IN POLYNOMIAL TIME?

A parity game is a two player game, of infinite duration and with perfect information, which comes up naturally when studying automata and logics for infinite trees. Model checking of the μ-calculus or testing emptiness of a parity tree automaton are problems that are polynomial time equivalent to solving parity games.

A parity game is played by two players, call them Even and Odd. The goal of player Even is to see small even numbers, the goal of player Odd will is to avoid this. The game is specified by:

— a finite directed graph, called the *arena*, such that every vertex has an outgoing edge;
— a partition of vertices in the arena, into vertices controlled by players Even and Odd;
— a *ranking function* which maps vertices in the arena to natural numbers;
— a distinguished initial vertex.

Here is an example of a parity game:



The game is played as follows. The game begins in the initial vertex. The player who controls the initial vertex chooses an outgoing edge. The player who controls the target of this edge chooses an edge leaving the target, and so on ad infinitum, until an infinite path in the graph is formed. The objective of player Even is to make sure that on this

path, the minimal value of the ranking function that is seen infinitely often is even (this objective is called the parity condition).

Determinacy. From the Martin Determinacy theorem it follows that every parity game is determined, which means that either player Even has a winning strategy (no matter what player Odd does, the parity condition is satisfied), or player Odd has a winning strategy (no matter what player Even does, the parity condition is violated). The algorithmic challenge is to decide which one is the case.

The open problem. Is there a polynomial time algorithm for solving parity games, i.e. determining which of the players has a winning strategy?

Memoryless determinacy. As shown in [Emerson and Jutla 1991] and [Mostowski 1991], parity games are not just determined, but even memoryless determined, i.e. if one of the players has winning strategy, then that player has a *memoryless* winning strategy, where the choice of next vertex depends only on the current vertex, and not on the entire history of the vertices visited before. For instance, in the game pictured above, player Even wins by using the following memoryless strategy: when in the southwest corner go north, and when in the the northwest corner go east (southeast would also be a good choice).

Memoryless determinacy leads to an NP algorithm for deciding if player Even has a winning strategy: guess a memoryless strategy for player Even, and then check in polynomial time if every path that is consistent with this strategy satisfies the parity condition. The same kind of algorithm works for player Odd, and therefore the problem is in NP $\cap$ coNP, and yet it is not known to be in P.

Known algorithms. As the statement of the open problem implies, there is no known algorithm for solving parity games in polynomial time. Two examples of known algorithms for solving parity games are: a divide and conquer algorithm [Zielonka 1998], and the strategy improvement algorithm [Vöge and Jurdziński 2000]. These algorithms have exponential worst case complexities, e.g. [Friedmann 2009] provides a lower bound for the strategy improvement algorithm. Interestingly, the insights obtained from analyzing the existing parity game algorithms can be used to get lower bounds for variants of the simplex algorithm in linear programming [Friedmann et al. 2011].

Fixed parameter tractability. Before solving parity games in polynomial time, one could at least try to show that the problem is fixed parameter tractable, for some choice of parameter. It is known that parity games are fixed parameter tractable for parameters of the arena such as: tree width [Obdrzálek 2003], clique width [Obdrzálek 2007], DAG width [Berwanger et al. 2012a], Kelly width [Hunter and Kreutzer 2008] or entanglement [Berwanger et al. 2012b]. Perhaps the most natural parameter is the number of ranks used by the ranking function – and fixed parameter tractability is open for this particular parameter. In other words, it is not known if there is an algorithm which solves a parity game with k ranks and n vertices in time $f(k) \cdot n^c$ for some computable function f and some exponent c which does not depend on k.

2. DO ALL REGULAR LANGUAGES HAVE GENERALISED STAR HEIGHT ONE?

A *generalised regular expression* is one that can use complementation along the more standard operations of concatenation, union and Kleene star. Since regular languages are closed under complementation, generalised regular expressions have the same expressive power as standard regular expressions, although they can be more succinct, even nonelementarily more succinct as shown in [Stockmeyer 1974].

The open problem. Is there a regular language of finite words which cannot be defined by a generalised regular expression of star height one, i.e. one that does not nest the Kleene star?

This open problem is one of the questions concerning star height, which have motivated a lot of research in automata theory. The star height of a regular expression, generalised or not, is defined to be the nesting depth of the Kleene star. For instance, the (non-generalised) regular expression $(a^*b)^*a^*$ has star height 2. The star height of a language is defined to be the smallest star height of a regular expression that defines it. There are two variants of star height, generalised and non-generalised, depending on the type of regular expressions that are considered. In other words, the open problem is: does every regular language have generalised star height zero or one?

Non-generalised star height. Non-generalised star height is by now quite well understood. There are regular languages that have arbitrarily high non-generalised star height [Eggan 1963]. Whether or not the non-generalised star height can be computed was an open problem for 35 years, until it was shown to be computable in [Hashiguchi 1988], see [Kirsten 2005] for a simpler proof.

Generalised star height. Much less is known about the hierarchy of generalised star height. The only level that is understood is level zero, which is called the *star-free languages*, i.e. the languages that can be defined from finite languages using only Boolean operations and concatenation. A famous result shown in [Schützenberger 1965] says that a languages is star-free if and only if it is recognised by a finite monoid which does not contain a nontrivial group. There is also an important logical connection, shown in [McNaughton and Papert 1971]: a language is star-free if and only if it can be defined by a formula of first-order logic, which quantifies over positions of the word, has a binary predicate $x \leq y$ for the order on positions, and has unary predicates $a(x)$. For example, the formula

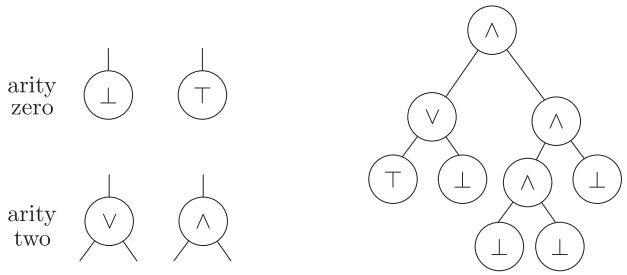
$$\forall x \exists y \; x \leq y \wedge a(y)$$

says that the every position is followed by a position with label a. The language defined by this example formula is star-free because it is the concatenation of the set of all words (the complement of the empty set) with the finite language $\{a\}$.

Beyond star-free languages, i.e. beyond level zero, very little is known. As stated in the open problem, as far as we know, maybe all regular languages have star height at most one, or maybe at most fifteen. As shown in [Pin et al. 1992], assuming that some languages have star height at least two, then the generalised star height of a regular language cannot be determined just by looking at the language's syntactic monoid. The paper [Pin et al. 1992] also contains several surprising languages that have generalised star height one, including the language "an even number of infixes of the form abc", which was conjectured to have bigger generalised star height in [Brzozowski 1980].

3. WHICH REGULAR LANGUAGES OF FINITE TREES ARE FIRST-ORDER DEFINABLE?

This problem concerns regular languages of finite trees. For the sake of concreteness, we use the variant of ranked trees. In this variant, the trees are over a *ranked alphabet*, where each letter comes with an arity, and trees are labelled by the alphabet so that the number of children of a node is the arity of its label, as depicted in the following picture.



Tree automata. A good notion of tree automaton for such trees is a *bottom-up deterministic automaton*. In such an automaton, if the states are Q, then every letter a of arity n induces a transition function of type $Q^n \to Q$. In particular, each leaf, which has a label of arity zero, comes with an associated state. By applying the transition functions in a bottom-up manner, the automaton evaluates a tree to a single state. A tree language is called regular if there is some automaton such that membership in the language is uniquely determined by the state to which a tree evaluates.

Monadic second-order logic on trees. Many results on automata can be generalised from finite words to finite trees without much difficulty, e.g. pumping lemmas or equivalence of deterministic and non-deterministic bottom-up automata. Another example of result that can be generalised is the following correspondence between monadic second-order logic and regular languages. A tree can be seen as a logical structure, where the universe is the set of nodes, and which has the following predicates: a unary predicate "node x has label a" for every letter a of the alphabet, a binary predicate "node x is a descendant of node y", and a binary predicate "node x is the i-th child of node y" for every number i up to the maximal arity in the alphabet. A classical result on tree automata [Thatcher and Wright 1968] says that a tree language is regular if and only if it is definable in monadic second-order logic, i.e. the logic which can quantify over nodes and sets of nodes.

First-order logic on trees. Languages that are definable in first-order logic are a strict subclass of regular tree languages. For instance the language "some leaf is at even depth" is not definable in first-order logic if the alphabet includes at least one letter of rank one. (Although, as will be later discussed, the language is definable, somehow annoyingly, when there are no letters of rank one.) Another example, which does not have any natural word counterpart, is the language of Boolean expressions, as in the picture above, that evaluate to $\top$.

The open problem. Can one decide whether a regular language of finite trees can be defined by a formula of first-order logic that uses the label predicates, the descendant predicate, and i-th child predicates?

In the case of finite words, the problem above is well understood. The results of Schützenberger, McNaughton and Papert, which were mentioned previously in the context of star-free languages, imply that one can decide if a regular language of finite words can be defined in first-order logic: one can compute the syntactic monoid and then test if it contains a nontrivial group.

For trees much less is known. The open problem dates back to [Thomas 1984]. In [Heuter 1991] it was shown that aperiodicity (in a natural tree variant) is a ne-

cessary but not sufficient condition for definability in first-order logic on finite trees. In other words, the equivalence of first-order definability and aperiodicity fails to extend from words to trees. For the equivalence of first-order definability and star-freeness, the story is a bit more complicated: there are variants of star-free expressions that are strictly more expressive than first-order logic [Thomas 1984], and there are variants of star-free expressions that are equally expressive as first-order logic [Bojańczyk 2007]. Finally, first-order logic on finite trees admits characterisations in terms of temporal logics, namely two-way CTL [Schlingloff 1992] and one-way CTL* [Hafer and Thomas 1987]. These characterisations show that first-order logic on trees is a robust concept, but they do not seem to be useful in solving the open problem.

As usual for trees, there are several variants of the problem, e.g. one can consider the logic without the i-th child predicates, or one can consider unranked trees with or without a predicate for sibling order. In all of these cases it is not known how to characterise first-order logic.

The only known result is about full first-order logic without the descendant relation, where only predicates for the labels and the child relation are allowed. In this case, one can decide if a language is definable in first-order logic [Benedikt and Segoufin 2009]. Other known results talk about fragments of first-order logic with limited quantification patters, e.g. Boolean combinations of existential formulas [Bojanczyk et al. 2012] or first-order logic with only two variables [Place and Segoufin 2010].

To give a taste of the difficulties inherent to first-order logic on trees, consider the following example, which is due to [Potthoff 1995]. Assume that the alphabet has one letter of rank zero and one letter of rank two, and the language is "some leaf is at even depth". As mentioned before, this language is definable in first-order logic, using the descendant, left child and right child predicates. Despite being invariant under swapping left and right subtrees, the language is not definable in first-order logic using the descendant predicate only.

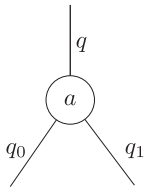
4. THE RABIN-MOSTOWSKI INDEX HIERARCHY

Tree automata are studied – arguably, better studied – also for infinite trees. Let us consider labelled binary trees, where every node has a label from a finite set and exactly two children. The famous Rabin theorem [Rabin 1969] says that a language of labelled binary trees is definable in MSO if and only if it is recognised by a nondeterministic automaton with the Rabin condition. Currently, instead of automata with the Rabin condition, one uses the parity condition, in their nondeterministic and alternating variants, as described below. Both of these variants are equally expressive as MSO, and therefore as nondeterministic automata with the Rabin condition.

Nondeterministic parity tree automata. A nondeterministic parity tree automaton consists of an input alphabet A, a set of states Q, an initial state $q_0 \in Q$, a parity ranking function $\Omega : Q \to \mathbb{N}$, and a transition relation

$$\delta \subseteq Q \times A \times Q \times Q.$$

A transition can be visualised as a node in a binary tree, with the node labelled by the input alphabet, and its surrounding edges labelled by states:



When given an input tree, a run of the automaton is a labelling of the edges of the tree by states, including a special dummy edge that enters the root from above, such that the neighbourhood of every node is consistent with some transition. A run is accepting if the dummy edge gets the initial state, and on every infinite path, the parity condition is satisfied by the ranks assigned to the states.

Alternating parity tree automata. An alternating parity tree automaton differs from a nondeterministic one in two ways: a) there is a partition of the states into states owned by players Even and Odd, b) the transition relation is a subset

$$\delta \subseteq Q \times A \times \{\text{same, left, right}\} \times Q.$$

To determine whether or not such an automaton accepts an input tree, a parity game is played between players Even and Odd. Positions of the game are pairs (q, v), where q is a state of the automaton and v is a node of the input tree. The initial position is the pair consisting of the initial state and the root of the tree. When the game is in a position (q, v), then the player who controls state q chooses a transition (q, a, i, p), and the game continues from position (p, w), where w is either the same node as v, the left child of v, or the right child of v, depending on the value of i. The tree is accepted if player Even has a winning strategy in this game, according to the parity condition as determined by the parity ranking function on the states.

The index hierarchies. For numbers $i \leq j$, consider parity automata, either nondeterministic or alternating, where the parity ranking function has image included in $\{i, \ldots, j\}$. It is easy to see that the class of recognised languages depends only on the parity of the smaller rank i, and on the number of available ranks $j - i + 1$. In other words, one can assume without loss of generality that i is either 0 or 1.

Adding more ranks gives a strict hierarchy, in the following sense, see [Bradfield 1998; Arnold 1999]. For every i and every $j \geq i$, there exists a language L_{ij} which is recognised by a nondeterministic parity automaton with ranks $\{i, \ldots, j\}$, but which is not recognised by an alternating parity automaton with ranks $\{i+1, \ldots, j+1\}$, i.e. the same number of ranks but shifted by one. The open problem is to decide the position of a regular language of infinite trees in this hierarchy:

The open problem. Are the following decision problems decidable? In each of them, the input is a regular language of infinite trees and natural numbers $i \leq j$.

— **Nondeterministic index problem.** Is the language recognised by some nondeterministic parity automaton that uses ranks between i and j?
— **Alternating index problem.** Is the language recognised by some alternating parity automaton that uses ranks between i and j?

Known results. The nondeterministic and alternating index problems are known to be decidable assuming that the input regular language is recognised by a top-down deterministic parity automaton [Niwiński and Walukiewicz 2003], or even assuming that the input regular language is recognised by a generalisation of deterministic top-down

automata called *game automata* [Duparc et al. 2009]. In [Colcombet and Löding 2008], the nondeterministic index problem is reduced to a problem about counter automata on infinite trees; the latter problem (which is interesting in its own right) remains open. A partial result on these counter automata is given in [Colcombet et al. 2013], this partial result implies that one can decide, given a language recognised by a nondeterministic Büchi automaton (i.e. using ranks 0 and 1), if the complement of the language is also recognised by a nondeterministic Büchi automaton.

Weak MSO. An important special case of the index problems is deciding if a regular tree language can be defined in weak MSO, i.e. the variant of MSO where set quantification is restricted to finite sets. Why is this a special case? As shown in [Rabin 1970], a language of infinite trees is definable in weak MSO if and only if both the language and its complement are recognised by nondeterministic Büchi tree automata. Furthermore, with the Büchi acceptance condition, nondeterministic and alternating automata have the same expressive power [Muller and Schupp 1995]. Therefore, if one could decide which regular languages of infinite trees are recognised by nondeterministic (equivalently, alternating) Büchi tree automata, then one could decide which ones are definable in weak MSO. A related conjecture, see [Skurczyński 1993], is that for regular languages of infinite trees, being Borel is equivalent to being definable in weak MSO.

5. MONADIC SECOND-ORDER LOGIC ON THE CANTOR SPACE.

Consider the sets 2^* and 2^ω as logical structures: the former is equipped with both descendant and lexicographic orders, and the latter is equipped with only the lexicographic order. The Rabin theorem says that the first structure has decidable MSO theory. Shelah showed that the second structure, i.e. the Cantor space, has undecidable MSO theory. A corollary is that the real numbers have undecidable MSO theory. Shelah's original proof in [Shelah 1975] uses the Continuum Hypothesis, while a later proof [Gurevich and Shelah 1982] uses only the axioms of ZFC. In the undecidability proof, it is important that formulas can quantify over arbitrary subsets of 2^ω and not just simple sets, such as Borel sets. Therefore, Shelah stated the following problem, which remains open.

The open problem. Can one decide the MSO theory of the Cantor space 2^ω with lexicographic ordering, assuming that set qantifiers range only over Borel sets?

To illustrate the problem, consider a logic which uses a smaller prefix of the Borel hierarchy, namely level Π_2, i.e. the countable intersections of open sets. Consider MSO logic over 2^ω where set quantification is restricted to Π_2 sets. We claim that this logic is decidable, as it reduces to MSO on 2^* with left and right child predicates, i.e. to the logic from Rabin's theorem. The idea is to reduce both first-order quantification and set quantification to set quantification in 2^*. For first-order quantification, one simply encodes an element of 2^ω as the set of its finite prefixes, i.e. a path in a tree. To encode a set $X \subseteq 2^\omega$ that is in Π_2, one uses the following observation, which follows easily from the definition of level Π_2 in the Borel hierarchy: a set $X \subseteq 2^\omega$ belongs to Π_2 if and only if there exists a subset $[X] \subseteq 2^*$ such that

$$\pi \in X \qquad \text{iff} \qquad \pi \text{ has infinitely many prefixes in } [X].$$

This reduction seems to use almost all of the power of the Rabin theorem. It seems that giving a positive answer to the open problem would require a significant extension of the techniques in the Rabin theorem.

6. IS REACHABILITY DECIDABLE FOR BRANCHING VECTOR ADDITION SYSTEMS?

One of the most famous decidability results [Mayr 1984] is that reachability is decidable in vector addition systems with states. A vector addition system with states, VASS for short, is a type of counter machine that is equivalent to Petri nets, and is defined as follows. The syntax of a VASS consists of a finite set of states Q with a distinguished initial state, a dimension $d \in \mathbb{N}$, and a finite subset of $\delta \subseteq Q \times \mathbb{Z}^d \times Q$, which is called the set of transitions. The semantics of a VASS is the set of *reachable configurations*, which is the least set of pairs in $Q \times \mathbb{N}^d$ such that:

— If q is the initial state, then $(q, \bar{0})$ is a reachable configuration.
— If a configuration $(q, \bar{a})$ is reachable, and there is a transition $(q, \bar{b}, p)$ such that the vector $\bar{a} + \bar{b}$ has only nonnegative numbers, then also $(p, \bar{a} + \bar{b})$ is reachable.

The key thing is that transitions can have negative numbers, while configurations cannot. As shown in [Mayr 1984], the set of reachable configurations is decidable. One notorious open problem is the computational complexity of reachability – the best lower bound is EXPSPACE [Cardoza et al. 1976], but it is not even known if reachability has elementary complexity. A recent decidability proof can be found in [Leroux 2012].

Branching vector addition systems with states. A branching vector addition system with states, BVASS for short, generalises a VASS by allowing a new kind of transition, call it a *branching transition*, which is a triple of states. The set of reachable configurations is defined as in a VASS, with the additional rule that if $(q, \bar{a})$ and $(p, \bar{b})$ are reachable configurations, and the BVASS contains a branching transition (p, q, r), then also $(r, \bar{a} + \bar{b})$ is a reachable configuration.

The open problem. Is reachability decidable for branching vector addition systems?

It is known that reachability for BVASS is Ackermann hard [Lazić and Schmitz 2014], i.e. if it is decidable then the running time of the algorithm must be bigger than the Ackermann function. Control state reachability – i.e. the problem of determining which states can appear in reachable configurations – is known to be decidable [Verma and Goubault-Larrecq 2005].

Connections with logic. Here are two examples of how the problem is connected with logic. The first connection comes from linear logic – the decidability of the reachability problem for BVASS is equivalent to the decidability of multiplicative exponential linear logic [de Groote et al. 2004]. The second connection comes from the theory of XML – the reachability problem for BVASS reduces to the satisfiability problem of a certain logic on data trees, namely two variable first-order logic with predicates for descendant, successor, next sibling, and equal data value [Bojańczyk et al. 2009].

References

A. Arnold. 1999. The μ-calculus alternation-depth hierarchy is strict on binary trees. *Informatique Théorique et Applications* 33, 4/5 (1999), 329–340.

Michael Benedikt and Luc Segoufin. 2009. Regular tree languages definable in FO and in FO_{mod}. *ACM Trans. Comput. Log.* 11, 1 (2009).

Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdrzálek. 2012a. The dag-width of directed graphs. *J. Comb. Theory, Ser. B* 102, 4 (2012), 900–923.

Dietmar Berwanger, Erich Grädel, Łukasz Kaiser, and Roman Rabinovich. 2012b. Entanglement and the Complexity of Directed Graphs. *Theoretical Computer Science* 463, 0 (2012), 2–25. http://logic.rwth-aachen.de/~rabinovich/entangle.pdf

Mikołaj Bojańczyk. 2007. Forest Expressions. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*. 146–160.

Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. 2009. Two-variable logic on data trees and XML reasoning. *J. ACM* 56, 3 (2009).

Mikolaj Bojanczyk, Luc Segoufin, and Howard Straubing. 2012. Piecewise testable tree languages. *Logical Methods in Computer Science* 8, 3 (2012).

J.C. Bradfield. 1998. The Modal μ-Calculus Alternation Hierarchy is Strict. *Theor. Comput. Sci.* 195, 2 (1998), 133–153.

J.A. Brzozowski. 1980. *Open Problems about Regular Languages*. Department of Computer Science, University of Waterloo.

E. Cardoza, Richard J. Lipton, and Albert R. Meyer. 1976. Exponential Space Complete Problems for Petri Nets and Commutative Semigroups: Preliminary Report. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*. 50–54.

Thomas Colcombet, Denis Kuperberg, Christof Löding, and Michael Vanden Boom. 2013. Deciding the weak definability of Büchi definable tree languages. (2013). submitted.

Thomas Colcombet and Christof Löding. 2008. The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata. In *ICALP (2)*. 398–409.

Philippe de Groote, Bruno Guillaume, and Sylvain Salvati. 2004. Vector Addition Tree Automata. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*. 64–73.

Jacques Duparc, Alessandro Facchini, and Filip Murlak. 2009. Linear Game Automata: Decidable Hierarchy Problems for Stripped-Down Alternating Tree Automata. In *CSL*. 225–239.

Lawrence C. Eggan. 1963. Transition graphs and the star-height of regular events. *Michigan Math. J.* 10, 4 (1963), 385–397.

E. Allen Emerson and Charanjit S. Jutla. 1991. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*. 368–377. DOI:http://dx.doi.org/10.1109/SFCS.1991.185392

Oliver Friedmann. 2009. An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*. 145–156.

Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. 2011. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*. 283–292.

Yuri Gurevich and Saharon Shelah. 1982. Monadic theory of order and topology in ZFC. *Annals of Mathematical Logic* 23, 2-3 (1982), 179–198.

Thilo Hafer and Wolfgang Thomas. 1987. Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree. In *Automata, Languages and Programming, 14th International Colloquium, ICALP87, Karlsruhe, Germany, July 13-17, 1987, Proceedings*. 269–279.

Kosaburo Hashiguchi. 1988. Relative star height, star height and finite automata with distance functions. In *Formal Properties of Finite Automata and Applications*. 74–88.

Uschi Heuter. 1991. First-order properties of trees, star-free expressions, and aperiodicity. *ITA* 25 (1991), 125–146.

Paul Hunter and Stephan Kreutzer. 2008. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.* 399, 3 (2008), 206–219.

Daniel Kirsten. 2005. Distance desert automata and the star height problem. *Informatique Théorique et Applications* 39, 3 (2005), 455–509.

Ranko Lazić and Sylvain Schmitz. 2014. Non-elementary complexities for branching VASS, MELL, and extensions. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. 61.

Jérôme Leroux. 2012. Vector Addition Systems Reachability Problem (A Simpler Solution). In *Turing-100 - The Alan Turing Centenary, Manchester, UK, June 22-25, 2012*. 214–228.

Ernst W. Mayr. 1984. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comp.* 13, 3 (1984), 441–459.

R. McNaughton and S. Papert. 1971. *Counter-Free Automata*. MIT Press.

Andrzej W. Mostowski. 1991. *Games with forbidden positions*. Technical Report. University of Gdańsk.

D. E. Muller and P. E. Schupp. 1995. Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science* 141 (1995), 69–107.

Damian Niwiński and Igor Walukiewicz. 2003. A gap property of deterministic tree languages. *Theor. Comput. Sci.* 1, 303 (2003), 215–231.

Jan Obdrzálek. 2003. Fast Mu-Calculus Model Checking when Tree-Width Is Bounded. In *CAV*. 80–92.

Jan Obdrzálek. 2007. Clique-Width and Parity Games. In *Computer Science Logic (CSL)*. 54–68.

Jean-Éric Pin, Howard Straubing, and Denis Thérien. 1992. Some Results on the Generalized Star-Height Problem. *Inf. Comput.* 101, 2 (1992), 219–250.

Thomas Place and Luc Segoufin. 2010. Deciding Definability in $FO_2(<)$ (or XPath) on Trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*. 253–262.

A. Potthoff. 1995. First-Order Logic on Finite Trees. In *Theory and Practice of Software Development (Lecture Notes in Computer Science)*, Vol. 915. 125–139.

M. O. Rabin. 1969. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the AMS* 141 (1969), 1–23.

M. O. Rabin. 1970. Weakly Definable Relations and Special Automata. *Mathematical Logic and Foundations of Set Theory* (1970), 1–23.

Bernd-Holger Schlingloff. 1992. Expressive completeness of temporal logic of trees. *Journal of Applied Non-Classical Logics* 2, 2 (1992), 157–180.

M. P. Schützenberger. 1965. On Finite Monoids Having Only Trivial Subgroups. *Information and Control* 8 (1965), 190–194.

Saharon Shelah. 1975. The Monadic Theory of Order. *The Annals of Mathematics* 102, 3 (1975), 379–419.

Jerzy Skurczyński. 1993. The Borel hierarchy is infinite in the class of regular sets of trees. *Theoretical Computer Science* 112, 2 (1993), 413–418.

L.J. Stockmeyer. 1974. *The Complexity of Decision Problems in Automata Theory and Logic*. Ph.D. Dissertation.

J. W. Thatcher and J. B. Wright. 1968. Generalized Finite Automata Theory With an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory* 2, 1 (1968), 57–81.

Wolfgang Thomas. 1984. Logical Aspects in the Study of Tree Languages. In *CAAP*. 31–50.

Kumar Neeraj Verma and Jean Goubault-Larrecq. 2005. Karp-Miller Trees for a Branching Extension of VASS. *Discrete Mathematics & Theoretical Computer Science* 7, 1 (2005), 217–230.

Jens Vöge and Marcin Jurdziński. 2000. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*. 202–215. DOI:http://dx.doi.org/10.1007/10722167_18

Wiesław Zielonka. 1998. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theor. Comput. Sci.* 200, 1-2 (1998), 135–183.

COMPLEXITY COLUMN

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

For years when I taught algorithms or complexity, I would tell my students that while there is an infinite range of intermediate complexity classes, problems occurring in practice tend to be complete for one of a handful of important complexity classes.

I used to say that no one knows why this happens. Now however I can say that it happens for an algebraic reason which Libor Barto and his colleagues are close to figuring out. I am thrilled to have as our first Complexity Column in the *SIGLOG Newsletter,* the following survey on CSP.

I am also happy to announce that upcoming complexity columns will include the following:

— Anuj Dawar, "The Nature and Power of Fixed-Point Logic with Counting," [recent progress in the search for a logical language capturing order-independent Polynomial Time], Jan. 1, 2015.
— Jakob Nordstrom, [using proof complexity to gain insights into the complexity of SAT].

Constraint Satisfaction Problem and Universal Algebra

Libor Barto,
Department of Algebra,
Faculty of Mathematics and Physics,
Charles University in Prague

This column gives a brief survey of current research on the complexity of the constraint satisfaction problem (CSP) over fixed constraint languages.

1. INTRODUCTION

The Constraint Satisfaction Problem (CSP) provides a common framework for expressing a wide range of both theoretical and real-life combinatorial problems [Rossi et al. 2006]. One solves an instance of CSP by assigning values to the variables so that the constraints are satisfied.

In this column I describe some dramatic recent progress in our understanding of the computational complexity of CSP over a **fixed constraint language**. This restricted framework is still broad enough to include many NP-complete problems, yet it is narrow enough to potentially allow a complete classification of all such CSP problems.

One particularly important achievement is the understanding of what makes a problem in this class computationally easy or hard. It is not surprising that hardness comes from lack of symmetry. However, usual objects capturing symmetry, automorphisms (or endomorphisms) and their groups (or semigroups), are not sufficient in this context. It turned out that the complexity of CSP is determined by more general symmetries: polymorphisms and their clones.

My aim in this column is to introduce the basics of this exciting area and highlight selected deeper results, in a way that is understandable to readers with a basic knowledge of computational complexity (see [Papadimitriou 1994; Arora and Barak 2009]). The presentation of the material is based on my talk "Universal algebra and the constraint satisfaction problem" delivered at the Association of Symbolic Logic North American Annual Meeting held in Boulder, Colorado, in 2014. A more detailed version of this column is being prepared for the *Bulletin of Symbolic Logic*.

2. CSP OVER A FIXED CONSTRAINT LANGUAGE

A constraint – such as $R(x_3, x_1, x_4)$ – restricts the allowed values for a tuple of variables – in this case (x_3, x_1, x_4) – to be an element of a particular relation on the domain – in this case $R \subseteq D^3$.[1] By an n-ary *relation* R on a domain D we mean a subset of the n-th cartesian power D^n. It is sometimes convenient to work with the corresponding *predicate* which is a mapping from D^n to $\{\mathrm{true}, \mathrm{false}\}$ specifying which tuples are in R. We will use both formalism, so e.g. $(a, b, c) \in R$ and $R(a, b, c)$ both mean that the triple $(a, b, c) \in D^3$ is from the relation R.

An *instance* of CSP is a list of constraints, e.g.,

$$R(x),\ S(y, y, z),\ T(y, w),$$

where R, S, T are relations of appropriate arity on a common domain D and x, y, z, w are variables. A mapping f assigning values from the domain to variables is a *solution*

[1] There are also different types of constraints considered in the literature, see e.g. Chapter 7 in [Rossi et al. 2006].

if it satisfies all the constraints, that is, in our example,

$$R(f(x)) \text{ and } S(f(y), f(y), f(z)) \text{ and } T(f(y), f(w)) \ .$$

Three basic computational problems associated with an instance are the following:

— **Satisfiability.** Does the given instance have a solution? (A related problem, the *search problem*, is to find some solution if at least one solution exists.)
— **Optimization.** Even if the instance has no solution, find an optimal assignment, i.e., one that satisfies the maximum possible number of constraints. (*Approximation algorithms* are extensively studied, where the aim is, for example, to find an assignment that satisfies at least 80% of the number of constraints satisfied by an optimal assignment.)
— **Counting.** How many solutions does the given instance have? (This problem also has an approximation version: *approximate counting*.)

2.1. Satisfiability over a fixed constraint language

Even the easiest of the problems, satisfiability, is computationally hard: It contains many NP-complete problems including, e.g., 3-SAT (see Example 2.2). However, certain natural restrictions to CSP satisfiability ensure tractability. The main types of restrictions that have been studied are *structural restrictions*, which limit how constraints interact, and *language restrictions*, which limit the choice of constraint relations.

In this column, I focus just on satisfiability problems with language restrictions. Please see [Živný 2012] for optimization problems and a generalization to valued CSPs, [Håstad 2007] for approximation, [Cai and Chen 2012] for counting, and [Bodirsky 2008] for a generalization to infinite domains.

Definition 2.1. A *constraint language*, $\mathcal{D}$, is a set of relations on a common finite domain, D. We use $\mathrm{CSP}(\mathcal{D})$ to denote the set of CSP satisfiability problems whose relations are drawn from $\mathcal{D}$.

2.2. Examples

Example 2.2. An instance of the standard NP-complete problem, 3-SAT, is a Boolean formula in conjunctive normal form with exactly three literals per clause. For example, the formula,

$$\varphi \quad = \quad (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_4 \vee x_5 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_4 \vee \neg x_3)$$

is a satisfiable instance of 3-SAT. (Any assignment making x_1 and x_2 false, satisfies φ.) 3-SAT is equivalent to $\mathrm{CSP}(\mathcal{D}_{\mathrm{3SAT}})$, where $D_{\mathrm{3SAT}} = \{0, 1\}$ and

$$\mathcal{D}_{\mathrm{3SAT}} = \{S_{ijk} : i, j, k \in \{0, 1\}\}, \text{ where } S_{ijk} = \{0, 1\}^3 \setminus \{(i, j, k)\} \ .$$

For example, the above formula φ corresponds to the following instance of $\mathrm{CSP}(\mathcal{D}_{\mathrm{3SAT}})$

$$S_{010}(x_1, x_2, x_3),\ S_{101}(x_4, x_5, x_1),\ S_{111}(x_1, x_4, x_3) \ .$$

More generally, for a natural number k, k-SAT denotes a similar problem where each clause is a disjunction of k literals.

Since 3-SAT is NP-complete, it follows that k-SAT is NP-complete for each $k \geq 3$. On the other hand, 2-SAT is solvable in polynomial time, and is in fact complete for the complexity class NL (non-deterministic logarithmic space).

Example 2.3. HORN-3-SAT is a restricted version of 3-SAT, where each clause may have at most one positive literal. This problem is equivalent to $\mathrm{CSP}(\mathcal{D}_{\mathrm{HornSAT}})$ for $\mathcal{D}_{\mathrm{HornSAT}} = \{S_{011}, S_{101}, S_{110}, S_{111}\}$ (or just $\mathcal{D}_{\mathrm{HornSAT}} = \{S_{011}, S_{111}\}$). HORN-3-SAT is solvable in polynomial time, in fact, it is a P-complete problem.

Example 2.4. For a fixed natural number k, the k-COLORING problem is to decide whether it is possible to assign colors $\{1, 2, \ldots, k\}$ to vertices of an input graph in such a way that adjacent vertices receive different colors. This problem is equivalent to $\mathrm{CSP}(\mathcal{D}_{\mathrm{kCOLOR}})$, where $D_k = \{1, 2, \ldots, k\}$ and $\mathcal{D}_{\mathrm{kCOLOR}} = \{\neq_k\}$ consists of a single relation – the binary inequality relation $\neq_k = \{(a, b) \in D_k^2 : a \neq b\}$.

Indeed, given an instance of $\mathrm{CSP}(\mathcal{D})$, we can form a graph whose vertices are the variables and edges correspond to the binary constraints (that is, x has an edge to y iff the instance contains the constraint $x \neq_k y$). It is easily seen that the original instance has a solution if and only if the obtained graph is k-colorable. The translation in the other direction is similar.

The k-COLORING problem is NP-complete for $k \geq 3$. 2-COLORING is equivalent to deciding whether an input graph is bipartite. It is solvable in polynomial time, in fact, it is an L-complete problem (where L stands for logarithmic space) by a celebrated result of [Reingold 2008].

Example 2.5. Let p be a prime number. An input of 3-LIN(p) is a system of linear equations over the p-element field $\mathrm{GF}(p)$, where each equation contains 3 variables, and the question is whether the system has a solution. This problem is equivalent to $\mathrm{CSP}(\mathcal{D})$, where $D_{\mathrm{3LINp}} = \mathrm{GF}(p)$ and $\mathcal{D}_{\mathrm{3LINp}}$ consists of all affine subspaces of $\mathrm{GF}(p)^3$ of dimension 2 or 3:

$$\mathcal{D}_{\mathrm{3LINp}} = \{R_{abcd} : a, b, c, d \in \mathrm{GF}(p)\}, \text{, where } R_{abcd} = \{(x, y, z) \in \mathrm{GF}(p)^3 : ax{+}by{+}cz = d\} \ .$$

This problem is solvable in polynomial time, e.g. by Gaussian elimination. It is complete for a somewhat less familiar class $\mathrm{Mod}_p\mathrm{L}$.

Example 2.6. An instance of the s, t-connectivity problem, STCON, is a directed graph and two vertices s, t. The question is whether there exists a directed path from s to t.

A closely related (but not identical) problem is $\mathrm{CSP}(\mathcal{D}_{\mathrm{STCON}})$, where $D_{\mathrm{STCON}} = \{0, 1\}$ and $\mathcal{D}_{\mathrm{STCON}} = \{C_0, C_1, \leq\}$, $C_0 = \{0\}$, $C_1 = \{1\}$, $\leq = \{(0, 0), (0, 1), (1, 1)\}$. Indeed, given an instance of $\mathrm{CSP}(\mathcal{D}_{\mathrm{STCON}})$ we form a directed graph much as we did in Example 2.4 and label some vertices 0 or 1 according to the unary constraints. Then the original instance has a solution if and only if there is no directed path from a vertex labeled 1 to a vertex labeled 0. Thus $\mathrm{CSP}(\mathcal{D}_{\mathrm{STCON}})$ can be solved by invoking the complement of STCON, the s, t-non-connectivity problem, several times.

Both STCON and $\mathrm{CSP}(\mathcal{D}_{\mathrm{STCON}})$ can clearly be solved in polynomial time. By the Immerman-Szelepcsényi theorem [Immerman 1988; Szelepcsényi 1988] both problems are NL-complete.

In the same way, the s, t-connectivity problem for undirected graphs is closely related to $\mathrm{CSP}(\mathcal{D}_{\mathrm{USTCON}})$, where $D_{\mathrm{USTCON}} = \{0, 1\}$ and $\mathcal{D}_{\mathrm{USTCON}} = \{C_0, C_1, =\}$. These problems are L-complete by [Reingold 2008].

2.3. The dichotomy conjecture

The most fundamental problem in the area was formulated in the landmark paper [Feder and Vardi 1998].

CONJECTURE 2.7 (THE DICHOTOMY CONJECTURE). *For every finite*[2] *constraint language $\mathcal{D}$, the problem $\mathrm{CSP}(\mathcal{D})$ is in P or is NP-complete.*

[2]It is conjectured in [Bulatov et al. 2005] that the dichotomy remains true without the finiteness assumption. Namely, the local-global conjecture states that $\mathrm{CSP}(\mathcal{D})$ is in P (NP-complete) whenever $\mathrm{CSP}(\mathcal{D}')$ is in P (NP-complete) for every (some) finite $\mathcal{D}' \subseteq \mathcal{D}$.

Recall that if P $\neq$ NP, then there are problems of intermediate complexity [Ladner 1975]. Feder and Vardi argued that CSPs over a fixed constraint language is a good candidate for a largest natural class of problems with P versus NP-complete dichotomy.

At that time the conjecture was supported by two major cases: the dichotomy theorem for all languages over the two-element domain [Schaefer 1978] and the dichotomy theorem for languages consisting of a single binary symmetric relation [Hell and Nešetřil 1990].

Feder and Vardi have identified two sources of polynomial-time solvability and made several important contributions toward understanding these sources. In particular, they observed that the known polynomial cases were tied to algebraic closure properties and asked whether polynomial solvability for CSP can always be explained in such a way. Subsequent papers have shown that this is indeed the case and this connection to algebra brought the area to another level.

The algebraic approach is outlined in section 3 and some fruits of the theory discussed in section 4.

2.4. Alternative views

Note that a constraint language $\mathcal{D}$ with domain D, $\mathcal{D} = (D; R_1, R_2, \dots)$, is exactly a relational structure, or equivalently relational database, with universe D.

Recall that a *conjunctive query* over the database $\mathcal{D}$ is an existential sentence whose quantifier-free part is a conjunction of atoms. $\mathrm{CSP}(\mathcal{D})$ is exactly the problem of deciding whether $\mathcal{D}$ satisfies a given conjunctive query. For example, the instance

$$R(x),\ S(y,y,z),\ T(y,w)$$

has a solution if and only if the sentence

$$(\exists x,y,z,w \in D)\ R(x) \wedge S(y,y,z) \wedge T(y,w)$$

is true in $\mathcal{D}$.

From this perspective, it is natural to ask what happens if we allow some other combination of logical connectives $\{\exists, \forall, \wedge, \vee, \neg, =, \neq\}$. It turns out that out of the 2^7 cases only 3 are interesting (the other cases either reduce to these, or are almost always easy or hard by known results): $\{\exists, \wedge\}$ which is CSP, $\{\exists, \forall, \wedge\}$ which is so called *quantified CSP*, and $\{\exists, \forall, \wedge, \vee\}$. The complexity of quantified CSP is also an active research area [Chen 2012] with possible trichotomy P, NP-complete or Pspace-complete. Recently, a tetrachotomy was obtained for the last choice [Madelaine and Martin 2011] – for every $\mathcal{D}$, the corresponding problem is either in P, NP-complete, co-NP-complete, or Pspace-complete.

The CSP over a fixed language can also be formulated as the homomorphism problem between relational structures with a fixed target structure [Feder and Vardi 1998]. The idea of the translation is shown in Examples 2.4, 2.6.

3. UNIVERSAL ALGEBRA IN CSP

If a computational problem $\mathcal{A}$ can simulate (in some sense) another problem $\mathcal{B}$, then $\mathcal{A}$ is at least as hard as $\mathcal{B}$. This simple idea is widely used in computational complexity; for instance, NP-completeness is often shown by a gadget reduction of a known NP-complete problem to the given one. A crucial fact for the algebraic theory of CSP is that so called primitive positive (pp-, for short) interpretation between constraint languages gives such a reduction between corresponding CSPs (more precisely, if $\mathcal{D}$ pp-interprets $\mathcal{E}$, then $\mathrm{CSP}(\mathcal{E})$ is reducible to $\mathrm{CSP}(\mathcal{D})$). Pp-interpretations have been, indirectly, the main subject of universal algebra for the last 80 years!

The algebraic theory of CSPs was developed in a number of papers including [Jeavons et al. 1997; Jeavons 1998; Bulatov et al. 2005; Larose and Tesson 2009]. The view-

point taken here is close to [Bodirsky 2008]. All results in this section come from these sources unless stated otherwise.

To simplify formulations, all constraint languages are assumed to contain *finitely many* relations, all of them *nonempty*. By a *reduction* we mean a logarithmic space reduction (although first-order reductions are often possible under additional weak assumptions).

3.1. Primitive positive interpretations

An important special case of pp-interpretability is pp-definability.

Definition 3.1. Let $\mathcal{D}, \mathcal{E}$ be constraint languages on the same domain $D = E$. We say that $\mathcal{D}$ *pp-defines* $\mathcal{E}$ (or $\mathcal{E}$ is pp-definable from $\mathcal{D}$) if each relation in $\mathcal{E}$ can be defined by a first order formula which only uses relations in $\mathcal{D}$, the equality relation, conjunction and existential quantification.

THEOREM 3.2. *If $\mathcal{D}$ pp-defines $\mathcal{E}$, then* $\mathrm{CSP}(\mathcal{E})$ *is reducible to* $\mathrm{CSP}(\mathcal{D})$.

Example 3.3. Let R be an arbitrary ternary relation on a domain D. Consider the relations on D defined by

$$S(x,y) \text{ iff } (\exists z)R(x,y,z) \wedge R(y,y,x), \quad T(x,y) \text{ iff } R(x,x,x) \wedge (x=y) \ ,$$

where the existential quantification is understood over D. The relations S and T are defined by pp-formulae, therefore the constraint language $\mathcal{D} = \{R\}$ pp-defines the constraint language $\mathcal{E} = \{S, T\}$.

We sketch the reduction of $\mathrm{CSP}(\mathcal{E})$ to $\mathrm{CSP}(\mathcal{D})$ using the instance

$$S(x_3, x_2),\ T(x_1, x_4),\ S(x_2, x_4) \ .$$

We first replace S and T with their pp-definitions by introducing a new variable for each quantified variable:

$$R(x_3, x_2, y_1),\ R(x_2, x_2, x_3), \quad R(x_1, x_1, x_1),\ x_1 = x_4, \quad R(x_2, x_4, y_2),\ R(x_4, x_4, x_2)$$

and then we get rid of the equality constraint $x_1 = x_4$ by identifying these variables. This way we obtain an instance of $\mathrm{CSP}(\mathcal{D})$:

$$R(x_3, x_2, y_1),\ R(x_2, x_2, x_3),\ R(x_1, x_1, x_1),\ R(x_2, x_1, y_2),\ R(x_1, x_1, x_2) \ .$$

Clearly, the new instance of $\mathrm{CSP}(\mathcal{D})$ has a solution if and only if the original instance does.

This simple theorem provides a quite powerful tool for comparing CSPs over different languages *on the same domain*. A more powerful tool, which can also be used to compare languages with different domains, is pp-interpretability. Informally, a constraint language $\mathcal{D}$ pp-interprets $\mathcal{E}$, if the domain of $\mathcal{E}$ is a pp-definable relation (from $\mathcal{D}$) modulo a pp-definable equivalence, and the relations of $\mathcal{E}$ (viewed, in a natural way, as relations on D) are also pp-definable from $\mathcal{D}$.[3] Formally:

Definition 3.4. Let $\mathcal{D}, \mathcal{E}$ be constraint languages. We say that $\mathcal{D}$ *pp-interprets* $\mathcal{E}$ if there exists a natural number n, $F \subseteq D^n$, and an onto mapping $f : F \to E$ such that $\mathcal{D}$ pp-defines

— the relation F,
— the f-preimage of the equality relation on E, and
— the f-preimage of every relation in $\mathcal{E}$,

[3]This is the classical notion of interpretation from model theory restricted to pp-formulas.

where by the f-preimage of a k-ary relation S on E we mean the nk-ary relation $f^{-1}(S)$ on D defined by

$$f^{-1}(S)(x_{11},\dots,x_{1k},x_{21},\dots,x_{2k},\dots,x_{n1},\dots,x_{nk}) \text{ iff } S(f(x_{11},\dots,x_{n1}),\dots,f(x_{1k},\dots,x_{nk}))$$

THEOREM 3.5. *If $\mathcal{D}$ pp-interprets $\mathcal{E}$, then* $\mathrm{CSP}(\mathcal{E})$ *is reducible to* $\mathrm{CSP}(\mathcal{D})$.

Pp-interpretability is a reflexive and transitive relation on the class of constraint languages. By identifying equivalent languages, i.e. languages which mutually pp-interpret each other, we get a partially ordered set, the *pp-interpretability poset*. Theorem 3.5 then says that the "higher" we are in the poset the "easier" CSP we deal with. 3-SAT is terribly hard – its constraint language is the least element of this poset. Surprisingly, this is "almost" the case for all known NP-complete CSPs! For a precise formulation we need a reduction described in the following subsection.

3.2. Cores and singleton expansions

Let $\mathcal{D}$ be a constraint language on a finite set D. A mapping $f : D \to D$ is called an *endomorphism* if it preserves every relation $\mathcal{D}$, that is, $f(R) := \{f(\mathbf{a}) : \mathbf{a} \in R\} \subseteq R$ for every $R \in \mathcal{D}$.

THEOREM 3.6. *Let $\mathcal{D}$ be a constraint language and f an endomorphism of $\mathcal{D}$. Then* $\mathrm{CSP}(\mathcal{D})$ *is reducible to* $\mathrm{CSP}(f(\mathcal{D}))$ *and vice versa, where $f(\mathcal{D})$ is a constraint language with domain $f(D)$ defined by $f(\mathcal{D}) = \{f(R) : R \in \mathcal{D}\}$.*

A language $\mathcal{D}$ is a *core* if every endomorphism of $\mathcal{D}$ is a bijection. It is not hard to show that if f is an endomorphism of a constraint language $\mathcal{D}$ with minimal range, then $f(\mathcal{D})$ is a core. Moreover, this core is unique up to isomorphism, therefore we speak about *the core* of $\mathcal{D}$.

An important fact is that we can add all singleton unary relations to a core constraint language without increasing the complexity of its CSP:

THEOREM 3.7. *Let $\mathcal{D}$ be a core constraint language and $\mathcal{E} = \mathcal{D} \cup \bigcup_{a \in D} C_a$, where C_a denotes the unary relation $C_a = \{a\}$. Then* $\mathrm{CSP}(\mathcal{E})$ *is reducible to* $\mathrm{CSP}(\mathcal{D})$.

We will call constraint languages containing all singletons *idempotent*. Note that an idempotent constraint language is automatically a core as the only endomorphism is the identity. By Theorems 3.6, 3.7, CSP over $\mathcal{D}$ is reducible to CSP over the singleton expansion of the core of $\mathcal{D}$ and vice versa. It is therefore enough to study CSPs over idempotent constraint languages.

An interesting consequence of these reductions is that the search problem for $\mathrm{CSP}(\mathcal{D})$ is solvable in polynomial time whenever $\mathrm{CSP}(\mathcal{D})$ is. The idea is to gradually guess values for variables using the unary singleton constraints.

3.3. Tractability conjecture

Now we return to the pp-interpretability poset. Recall that "higher" in the poset means "easier" CSP and that 3-SAT corresponds to the least (the hardest) element. When we restrict to idempotent constraint languages (which we can do by the previous discussion), all known NP-complete CSPs are at the bottom of the poset. Bulatov, Jeavons and Krokhin conjectured that this is not a coincidence.[4]

CONJECTURE 3.8 (TRACTABILITY CONJECTURE). *If an idempotent constraint language $\mathcal{D}$ does not pp-interpret (the language of)* 3-SAT*, then* $\mathrm{CSP}(\mathcal{D})$ *is solvable in polynomial time.*

[4]Similar hardness results and conjectures are formulated for other computational/descriptive complexity classes.

This conjecture is also known as the *algebraic dichotomy conjecture* because many equivalent formulations, including the original one, are algebraic.

3.4. Algebraic side

The link between relations and operations is provided by a natural notion of compatibility. An n-ary operation f on a finite set D (that is, a mapping $f : D^n \to D$) is *compatible* with a k-ary relation $R \subseteq D^k$ if f applied component-wise to any n-tuple of elements of R gives an element of R. In more detail, whenever (a_{ij}) is an $n \times k$ matrix such that every row is in R, then f applied to the columns gives a k-tuple which is in R as well.

We say that an operation f on D is a *polymorphism* of a constraint language $\mathcal{D}$ if f is compatible with every relation in $\mathcal{D}$. Note that unary polymorphism is the same as endomorphism. Endomorphisms can be thought of as symmetries, polymorphisms are then symmetries of higher arity.

The set of all polymorphisms of $\mathcal{D}$ will be denoted by $\mathbf{D}$. This algebraic object is always a *concrete clone* meaning that it contains all projection operations (that is, operations of the form $\pi_i^n(x_1, \dots, x_n) = x_i$, also known as dictators) and is closed under composition. Therefore we refer to $\mathbf{D}$ as the *clone of polymorphims* of $\mathcal{D}$.

The clone of polymorphisms controls pp-definability in the sense of the following old result [Geiger 1968; Bodnarčuk et al. 1969].

THEOREM 3.9. *Let $\mathcal{D}$, $\mathcal{E}$ be constraint languages with $D = E$. Then $\mathcal{D}$ pp-defines $\mathcal{E}$ if and only if $\mathbf{D} \subseteq \mathbf{E}$.*[5]

In view of this result, Theorem 3.2 says that the complexity of $\mathrm{CSP}(\mathcal{D})$ only depends on the clone $\mathbf{D}$. More precisely, if $\mathbf{D} \subseteq \mathbf{E}$, then $\mathrm{CSP}(\mathcal{E})$ is reducible to $\mathrm{CSP}(\mathcal{D})$. Moreover, the proof of Theorem 3.9 gives a generic pp-definition of $\mathcal{E}$ from $\mathcal{D}$, which gives us a generic reduction of $\mathrm{CSP}(\mathcal{E})$ to $\mathrm{CSP}(\mathcal{D})$.

Example 3.10. It is a nice exercise to show that the language $\mathcal{D}_{\mathrm{3SAT}}$ of 3-SAT has no polymorphisms but projections. This means that $\mathcal{D}_{\mathrm{3SAT}}$ pp-defines every constraint language with domain $\{0, 1\}$.

Finally, we very briefly discuss the algebraic counterpart to pp-interpretability. The construction in Definition 3.4 corresponds to a similar construction on clones. An alternative viewpoint, which is missing on the relational side, follows from the foundation stone of universal algebra, the Birkhoff HSP theorem [Birkhoff 1935]: pp-interpretability depends on the identities (i.e. universally quantified equations) satisfied by polymorphisms. To illustrate this vague claim, we state one of many (e.g., [Taylor 1977; Hobby and McKenzie 1988; Maróti and McKenzie 2008; Kun and Szegedy 2009; Siggers 2010]) characterizations of the conjectured borderline between P and NP-complete CSPs [Barto and Kozik 2012a].

THEOREM 3.11. *Let $\mathcal{D}$ be an idempotent constraint language and $p > |D|$ a prime. Then the following are equivalent.*

— *$\mathcal{D}$ does not interpret the language of* 3-SAT.
— *$\mathbf{D}$ contains an operation t (equivalently, $\mathcal{D}$ has a polymorphism t) of arity p such that*

$$(\forall x_1, \dots, x_p \in D)\ t(x_1, \dots, x_p) = t(x_2, \dots, x_p, x_1)\ .$$

Even if the tractability conjecture or the dichotomy conjecture (or finer classification conjectures) turns out to be incorrect, we *know* that classes of CSPs in P, L, NL, ... *can be* characterized by identities concerning polymorphisms.

[5] Moreover, every concrete clone is the clone of polymorphisms of some (possibly infinite) constraint language.

4. RESULTS

Universal algebra serves the investigation in two ways: as a toolbox containing heavy hammers (such as [Hobby and McKenzie 1988]) and as a guideline for identifying interesting intermediate cases, which are hard to spot from the purely relational perspective. Major results include the following.

— The dichotomy theorem of Schaefer for CSPs over a two-element domain was generalized to a three-element domain [Bulatov 2011]. A simplification of this result and a generalization to four-element domains was announced by Marković et al.
— The dichotomy theorem of Hell and Nešetřil for CSPs over undirected graphs was generalized to digraphs with no sources or sinks [Barto et al. 0809].
— The dichotomy conjecture was proved for all constraint languages containing all unary relations [Bulatov 2011] (a simpler proof is in [Barto 2011]).

Notably, all known tractable cases are solvable by a combination of two basic algorithms, or rather algorithmic principles – local consistency, and the "few subpowers" algorithm. It is another significant success of the algebraic approach that the applicability of these principles is now understood.

4.1. Local consistency

The CSP over some constraint languages can be decided in polynomial time by constraint propagation algorithms, or, in other words, by enforcing local consistency. Such CSPs are said to have *bounded width*.

This notion comes in various versions and equivalent forms. We refer to [Feder and Vardi 1998] for formalizations using Datalog programs and games, to [Bulatov et al. 2008] for description using dualities, and to [Bulatov 2011; Barto 2014] for a notion suitable for infinite languages.

We informally sketch one possible definition. Let $k \leq l$ be positive integers. The (k,l)-*algorithm* derives the strongest possible constraints on k variables by considering l variables at a time. If a contradiction is found, the algorithm answers "no (solution)", otherwise it answers "yes". These algorithms work in polynomial time (for fixed k,l) and "no" answers are always correct. A constraint language $\mathcal{D}$ (or $\mathrm{CSP}(\mathcal{D})$) has *width* (k,l), if "yes" answers are correct for every instance of $\mathrm{CSP}(\mathcal{D})$. If $\mathcal{D}$ has width (k,l) for some k,l, we say that $\mathcal{D}$ has *bounded width*.

As an example, we consider the constraint language $\mathcal{D}_{2\mathrm{COLOR}}$ and the instance

$$x_1 \neq x_2,\ x_2 \neq x_3,\ x_3 \neq x_4,\ x_4 \neq x_5,\ x_5 \neq x_1\ .$$

The $(2,3)$-algorithm can certify that this instance has no solution as follows:

— We consider the variables x_1, x_2, x_3. Using $x_1 \neq x_2$, $x_2 \neq x_3$ we derive $x_1 = x_3$.
— We consider x_1, x_3, x_4. Using $x_3 \neq x_4$ and the already derived constraint $x_1 = x_3$ we derive $x_1 \neq x_4$.
— We consider x_1, x_4, x_5 and using $x_1 \neq x_4$, $x_4 \neq x_5$ and $x_5 \neq x_1$ we derive a contradiction.

In fact, 2-COLORING has width $(2,3)$, that is, such reasoning finds a contradiction for every unsatisfiable instance. Other examples of bounded width problems include HORN-3-SAT and 2-SAT.

In [Feder and Vardi 1998], the authors proved that problems 3-LIN(p) (and more generally, similar problems 3-LIN(M) over finite modules) do not have bounded width and conjectured that linear equations are essentially the only obstacle for having bounded width. An algebraic formulation was given by [Larose and Zádori 2007]. They proved that analogues of results in section 3 hold for bounded width, therefore no problem

which pp-interprets the language of 3-LIN(M) has bounded width, and conjectured that the converse is also true. After a sequence of partial results [Kiss and Valeriote 2007; Carvalho et al. 2009; Barto and Kozik 2009; Bulatov 2006], the conjecture was eventually confirmed in [Barto and Kozik 2014][6] and independently in [Bulatov 2009].

THEOREM 4.1. *An idempotent constraint language $\mathcal{D}$ has bounded width if and only if $\mathcal{D}$ does not interpret the language of* 3-LIN(M) for a finite module M.[7]

4.2. Few subpowers

Gaussian elimination not only solves 3-LIN(p), it also describes all the solutions in the sense that the algorithm can output a small (polynomially large) set of points in $\mathrm{GF}(p)^n$ so that the affine hull of these points is equal to the solution set of the original instance. A sequence of papers [Feder and Vardi 1998; Bulatov 2002; Bulatov and Dalmau 2006; Dalmau 2006] culminating in [Idziak et al. 2007; Berman et al. 2009] pushed this idea, in a way, to its limit.

We need some terminology to state the result. Let $\mathcal{D}$ be a constraint language and $\mathbf{D}$ its clone of polymorphism. A relation on D is a *subpower* of $\mathbf{D}$ if it is pp-definable from $\mathcal{D}$. Note that the set of solutions of any instance of $\mathrm{CSP}(\mathcal{D})$ can be viewed as a subpower of $\mathbf{D}$. Now $\mathbf{D}$ has *few subpowers* if each subpower can be obtained as a closure under polymorphisms of a small set (polynomially large with respect to the arity).[8]

THEOREM 4.2. *Let $\mathcal{D}$ be an idempotent constraint language. If $\mathbf{D}$ has few subpowers, then $\mathrm{CSP}(\mathcal{D})$ can be solved in polynomial time.*

5. CONCLUSION

We have seen that the complexity of the satisfiability problem for CSP over a fixed constraint language depends on "higher arity symmetries" – polymorphisms of the language. (We have only discussed languages with finite domains. The algebraic theory extends to interesting subclasses of infinite domain CSP [Bodirsky 2008]). Significant progress has been achieved using this insight, but the main problem, the dichotomy conjecture, is still open.

A similar approach can be applied to other variants of CSP over a fixed constraint language. In two of them, the main goal has been reached: the dichotomy for the counting problem was proved in [Bulatov 2013] (substantially simplified in [Dyer and Richerby 2013]) and for the robust satisfiability problem in [Barto and Kozik 2012b]. A generalization of the theory for the optimization problem and valued CSPs was given in [Cohen et al. 2013], and some links to universal algebra are emerging from research in the area of approximation algorithms (such as [Raghavendra 2008]).

Is this approach only applicable to CSPs over fixed languages? Or are we merely seeing a piece of a bigger theory?

ACKNOWLEDGMENTS

The author gratefully acknowledges the support of the Grant Agency of the Czech Republic, grant GACR 13-01832S.

REFERENCES

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach* (1st ed.). Cambridge University Press, New York, NY, USA.

[6] A modification required to handle infinite languages was given in [Barto 2014].

[7] Moreover, if $\mathcal{D}$ has bounded width, then it has width $(2, 3)$ with an appropriate notion of width. Also, the property of having bounded width can be checked in polynomial time given an idempotent $\mathcal{D}$ on input.

[8] The name comes from an equivalent property that $\mathbf{D}$ has only exponentially many subpowers.

Libor Barto. 2011. The dichotomy for conservative constraint satisfaction problems revisited. In *26th Annual IEEE Symposium on Logic in Computer Science—LICS 2011*. IEEE Computer Soc., Los Alamitos, CA, 301–310.

Libor Barto. 2014. The Collapse of the Bounded Width Hierarchy. (2014). manuscript.

Libor Barto and Marcin Kozik. 2009. Congruence Distributivity Implies Bounded Width. *SIAM J. Comput.* 39, 4 (2009), 1531–1542.

Libor Barto and Marcin Kozik. 2012a. Absorbing Subalgebras, Cyclic Terms, and the Constraint Satisfaction Problem. *Logical Methods in Computer Science* 8, 1 (2012).

Libor Barto and Marcin Kozik. 2012b. Robust satisfiability of constraint satisfaction problems. In *Proceedings of the 44th Symposium on Theory of Computing (STOC '12)*. ACM, New York, NY, USA, 931–940.

Libor Barto and Marcin Kozik. 2014. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *J. ACM* 61, 1, Article 3 (Jan. 2014), 19 pages.

Libor Barto, Marcin Kozik, and Todd Niven. 2008/09. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM J. Comput.* 38, 5 (2008/09), 1782–1802.

Joel Berman, Pawel Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. 2009. Varieties with few subalgebras of powers. *Transactions of The American Mathematical Society* 362 (2009), 1445–1473.

Garrett Birkhoff. 1935. On the structure of abstract algebras. *Proc. Camb. Philos. Soc.* 31 (1935), 433–454.

Manuel Bodirsky. 2008. Constraint Satisfaction Problems with Infinite Templates. In *Complexity of Constraints* (2009-05-05) *(Lecture Notes in Computer Science)*, Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer (Eds.), Vol. 5250. Springer, 196–228.

V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. 1969. Galois theory for Post algebras. I, II. *Kibernetika (Kiev)* 3 (1969), 1–10; ibid. 1969, no. 5, 1–9.

Andrei Bulatov. 2009. Bounded relational width. (2009). manuscript.

Andrei Bulatov and Víctor Dalmau. 2006. A simple algorithm for Mal′tsev constraints. *SIAM J. Comput.* 36, 1 (2006), 16–27 (electronic).

Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. 2005. Classifying the Complexity of Constraints Using Finite Algebras. *SIAM J. Comput.* 34 (March 2005), 720–742. Issue 3.

Andrei A. Bulatov. 2002. Mal'tsev constraints are tractable. *Electronic Colloquium on Computational Complexity (ECCC)* 034 (2002).

Andrei A. Bulatov. 2006. Combinatorial problems raised from 2-semilattices. *J. Algebra* 298, 2 (2006), 321–339.

Andrei A. Bulatov. 2011. Complexity of Conservative Constraint Satisfaction Problems. *ACM Trans. Comput. Logic* 12, 4, Article 24 (July 2011), 66 pages.

Andrei A. Bulatov. 2013. The Complexity of the Counting Constraint Satisfaction Problem. *J. ACM* 60, 5, Article 34 (Oct. 2013), 41 pages.

Andrei A. Bulatov, Andrei Krokhin, and Benoit Larose. 2008. Complexity of Constraints. Springer-Verlag, Berlin, Heidelberg, Chapter Dualities for Constraint Satisfaction Problems, 93–124.

Jin-Yi Cai and Xi Chen. 2012. Complexity of Counting CSP with Complex Weights. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC '12)*. ACM, New York, NY, USA, 909–920.

Catarina Carvalho, Víctor Dalmau, Petar Marković, and Miklós Maróti. 2009. CD(4) has bounded width. *Algebra Universalis* 60, 3 (2009), 293–307.

Hubie Chen. 2012. Meditations on Quantified Constraint Satisfaction. In *Logic and Program Semantics*, Robert L. Constable and Alexandra Silva (Eds.). Lecture Notes in Computer Science, Vol. 7230. Springer Berlin Heidelberg, 35–49.

David A. Cohen, Martin C. Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. 2013. An algebraic theory of complexity for discrete optimisation. *SIAM J. Comput.* 42, 5 (2013), 1915–1939.

Víctor Dalmau. 2006. Generalized majority-minority operations are tractable. *Log. Methods Comput. Sci.* 2, 4 (2006), 4:1, 14.

Martin E. Dyer and David Richerby. 2013. An Effective Dichotomy for the Counting Constraint Satisfaction Problem. *SIAM J. Comput.* 42, 3 (2013), 1245–1274.

Tomás Feder and Moshe Y. Vardi. 1998. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. Comput.* 28, 1 (1998), 57–104.

David Geiger. 1968. Closed systems of functions and predicates. *Pacific J. Math.* 27 (1968), 95–100.

Johan Håstad. 2007. On the efficient approximability of constraint satisfaction problems. In *Surveys in combinatorics 2007*. London Math. Soc. Lecture Note Ser., Vol. 346. Cambridge Univ. Press, Cambridge, 201–221.

Pavol Hell and Jaroslav Nešetřil. 1990. On the complexity of H-coloring. *J. Combin. Theory Ser. B* 48, 1 (1990), 92–110.

David Hobby and Ralph McKenzie. 1988. *The structure of finite algebras*. Contemporary Mathematics, Vol. 76. American Mathematical Society, Providence, RI. xii+203 pages.

Pawel Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. 2007. Tractability and learnability arising from algebras with few subpowers. In *Proceedings of the Twenty-Second Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*. IEEE Computer Society Press, 213–222.

Neil Immerman. 1988. Nondeterministic Space is Closed Under Complementation. *SIAM J. Comput.* 17, 5 (Oct. 1988), 935–938.

Peter Jeavons. 1998. On the algebraic structure of combinatorial problems. *Theoretical Computer Science* 200, 12 (1998), 185 – 204.

Peter Jeavons, David Cohen, and Marc Gyssens. 1997. Closure properties of constraints. *J. ACM* 44, 4 (1997), 527–548.

Emil Kiss and Matthew Valeriote. 2007. On tractability and congruence distributivity. *Log. Methods Comput. Sci.* 3, 2 (2007), 2:6, 20 pp. (electronic).

Gábor Kun and Mario Szegedy. 2009. A New Line of Attack on the Dichotomy Conjecture. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC '09)*. ACM, New York, NY, USA, 725–734.

Richard E. Ladner. 1975. On the structure of polynomial time reducibility. *J. Assoc. Comput. Mach.* 22 (1975), 155–171.

Benoît Larose and Pascal Tesson. 2009. Universal algebra and hardness results for constraint satisfaction problems. *Theor. Comput. Sci.* 410 (April 2009), 1629–1647. Issue 18.

Benoit Larose and László Zádori. 2007. Bounded width problems and algebras. *Algebra Universalis* 56, 3-4 (2007), 439–466.

F. Madelaine and B. Martin. 2011. A Tetrachotomy for Positive First-Order Logic without Equality. In *26th Annual IEEE Symposium on Logic in Computer Science—LICS 2011*. IEEE Computer Soc., Los Alamitos, CA, 311–320.

Miklós Maróti and Ralph McKenzie. 2008. Existence theorems for weakly symmetric operations. *Algebra Universalis* 59, 3-4 (2008), 463–489.

Christos H. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA. xvi+523 pages.

Prasad Raghavendra. 2008. Optimal algorithms and inapproximability results for every CSP?. In *STOC'08*. 245–254.

Omer Reingold. 2008. Undirected Connectivity in Log-space. *J. ACM* 55, 4, Article 17 (Sept. 2008), 24 pages.

Francesca Rossi, Peter van Beek, and Toby Walsh. 2006. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Thomas J. Schaefer. 1978. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*. ACM, New York, 216–226.

Mark H. Siggers. 2010. A strong Malcev condition for locally finite varieties omitting the unary type. *Algebra universalis* 64, 1-2 (2010), 15–20.

R. Szelepcsényi. 1988. The Method of Forced Enumeration for Nondeterministic Automata. *Acta Inf.* 26, 3 (Nov. 1988), 279–284.

Walter Taylor. 1977. Varieties obeying homotopy laws. *Canad. J. Math.* 29, 3 (1977), 498–527.

Stanislav Živný. 2012. *The complexity of valued constraint satisfaction problems*. Springer, Heidelberg. xviii+170 pages.

SEMANTICS COLUMN

MICHAEL MISLOVE, Tulane University
mwm@math.tulane.edu

This issue marks the first "real column" in the Semantics series. I am pleased that the guest author, Achim Jung, agreed to write this column. Achim is renowned for his research in denotational semantics, and in particular in domain theory, and he also established himself as a leader in the UK computer science academic community during his two terms as Head of Department at the University of Birmingham.

Achim has chosen to focus on pedagogy with a column entitled, "Teaching Denotational Semantics". Achim played a pivotal role in establishing the Midlands Graduate School in the Foundations of Computer Science, which runs an annual series of summer schools and other events. He was instrumental in obtaining initial funding for the series, assuring its success. Achim uses the course he teaches at the MGS summer school as a guide for what to include in a denotational semantics course, providing both a great template from which to work, as well as a clear rationale for why a short course on denotational semantics should be taught in the way he outlines. It's an excellent overview that offers guidance for those who are just starting to teach such a course, as well as a source of new ideas for those who have an established course already in hand.

Above and beyond the outline Achim gives us, there is an equally important question that his column answers: "*Why* should we teach denotational semantics?" Denotational semantics courses are fairly well represented in Europe, where the area has its origins. But on my side of the Atlantic, semantics is relegated to a day, or at best a week, in programming language courses. Achim's outline provides a laundry list of reasons why denotational semantics deserves a full course, offering a number of lessons that are applicable across the whole range of computer science. For instance, such a course demonstrates:

— *Abstraction:* abstracting away from low-level details allows one to treat a problem on a more abstract level, where it can be understood better and more easily solved. This teaches students to avoid irrelevant details, and to focus on the issues at the heart of the problem.
— *Modularity:* breaking a problem down into subproblems that are easier to solve, and then using solutions to the subproblems to build a solution to the main problem. This is one of the great lessons of abstract mathematics that comes to the fore in denotational semantics.
— *Proof techniques:* the stock and trade of denotational semantics is proving properties of programs and it provides numerous proof tools for this purpose. Achim points out several techniques that are standard components of the approach: structural induction, term model construction, logical relations and domain theory.

— *Mathematical thinking:* using mathematics as both a language in which to phrase problems precisely, and as a tool kit for solving those problems.

These are some of the benefits of a semantics course for computer science students. As for mathematics students, Achim argues that they are not interested in such a course. That may be true, but mathematics students also have a lot to learn from a course like Achim's: it demonstrates how mathematics can be used effectively to solve hard problems in a related discipline. Denotational semantics also offers a great middle ground between the abstract definition-theorem-proof world of abstract mathematics, and the reliance on combinatorics and calculations prevalent in complexity theory and algorithmic analysis. This latter point applies as well for computer science students, who should find in a semantics course a chance to experience abstract reasoning as a means for gaining a broader perspective about their discipline, as well as a new suite of tools to use in solving its problems.

Teaching Denotational Semantics

Achim Jung, University of Birmingham

1. INTRODUCTION

In 1969 Dana Scott suggested, [Scott 1993], that a Tarskian semantics could be given to programming languages by employing ordered structures of a certain kind, now known as *domains*. One of his key insights was that recursion can be modelled via the *least fixpoint* of endofunctions on domains. Shortly afterwards Gordon Plotkin, explored this proposal further; among the many results of his landmark paper [Plotkin 1977] he formulated and proved *computational adequacy*, which can be said to justify Scott's least-fixpoint semantics. Much of the development of denotational semantics since then can be traced back to these two papers and few would contest their status as "classics" of our subject.

50 years later, one would expect to see Scott's and Plotkin's work reflected in the standard syllabus of computer science degrees around the world, on a par with other classic results of our subject, such as the relationship between automata and languages, the undecidability of the halting problem, and NP-completeness. This, however, is not the case, despite a clutch of excellent textbooks appearing in the late 1980s and early 1990s: [Schmidt 1986; Tennent 1991; Nielson and Nielson 1991; Gunter 1992; Winskel 1993; Mitchell 1996]. Of the many possible reasons for this state of affairs, I believe the following are the most important: (a) the subject is notationally subtle and heavy at the same time, (b) students do not have enough mathematical background, and (c) the "rewards" for the time and effort invested appear limited to the student.

In this note I want to report on my own experiences of teaching denotational semantics, which has taken place in a variety of contexts and to some very different audiences, with my primary reference point being a five-lecture course delivered as part of the Midlands Graduate School in the Foundations of Computing Science (MGS), which since 2001 has taken place annually at one of the four partner universities: Birmingham, Leicester, Nottingham, and Sheffield.

My aim here is to present my choice of topics and methodology for these five lectures in the light of the problems listed above, and to report on the specific experience in that context. So the focus is on reflection, not on presenting the material in detail and what follows are not course notes, but I still hope that I will give enough detail so that a knowledgeable reader may be able to adopt some ideas for their own teaching.

I am grateful to Mike Mislove for giving me the opportunity to present my musings in the newly established SIGLOG Newsletter and I hope that my example will encourage others to use this venue to discuss matters of *communicating research* as well as research itself.

2. APPROACH

As I said at the beginning, denotational semantics is a challenging subject to teach, dealing as it does with an unusually wide range of mathematical ideas and complex

notations. Students of computer science are typically poorly prepared for such a study, and students of mathematics are not interested. Both constituencies have difficulties seeing the point of it all, and this phenomenon is my first main issue: How to motivate the subject and how to pique students' interest?

Many texts choose to start with a simple `while`-language, introduce its denotational semantics as state-transforming functions and explain the meaning of a `while`-loop as a least fixpoint. From a pedagogical point of view, it seems appropriate and correct to begin with a formalism that is familiar to the students but I'd like to propose that this is also a key problem: The language is simple and the semantics is hard. However, if we are to "sell" denotational semantics, then it should be the other way round!

Alternatively, one could begin immediately with PCF; it can be argued, after all, that it is nothing more than the core of any modern functional programming language and students can be assumed to have taken a course in that subject before. Thomas Streicher's excellent book [Streicher 2006] does exactly that. I have found, however, that in practice the learning curve is too steep for many in my target audience (which includes mathematicians as well as computer scientists).

The approach that I have adopted and that I want to advertise here, is the one that Carl Gunter chose for his textbook [Gunter 1992]. In a nutshell, the idea is to discuss the semantics of the simply typed lambda calculus as a primer for PCF. Apart from offering a gentler path towards domain semantics and the adequacy proof, it lays the foundation for other subjects that students may want to study, in particular, type theory.

No doubt, from a pedagogical perspective Gunter's approach has many inherent problems of its own — and one aim of this note is to discuss those — but there are many attractive features which tip the balance in its favour. Foremost, the presentation can be motivated by the desire to *prove theorems about calculi*. Without going into any detail, these concern compositionality, soundness, completeness, definability, adequacy, and extensionality. The methods employed in the proofs are themselves rich and varied, and they are ubiquitous in denotational semantics: proof by structural induction, term model construction, logical relations and domain theory. In fact, their applicability is not confined to denotational semantics and one of the joys of teaching at the Midlands Graduate School is to see these themes and methods emerge in different courses.

This emphasis on theorems marks me out as a mathematician, I admit, and it does not work for everyone. As an alternative motivation that may work better for more practically-minded computer science students, I'd like to advertise the approach that Martín Escardó has successfully tried at the Midlands Graduate School. He begins by writing down the PCF term for the "Gandy–Berger functional", [Berger 1990], which tests whether a predicate on Cantor space (the latter implemented as int $\to$ bool) is satisfiable. The term is small but quite incomprehensible and Martín's point is that it is not clear at all that it must terminate for total predicates. He then goes fairly deeply into the domain semantics of PCF and develops the topological machinery that one needs for the termination proof. The Gandy-Berger functional is not just a pedagogical device; it appears, for example, in Alex Simpson's implementation of exact real number integration, [Simpson 1998].

Returning to my approach, a second key argument for beginning with the lambda calculus is that despite its similarity to functional programming, still appears sufficiently "foreign" to students so that giving a semantics seems appropriate; in other words, the language is "hard," and it remains to make sure that the semantics is (or appears) "easy." Here we take advantage of the fact that the simply-typed lambda calculus only requires ordinary functions for its semantics, and that the discussion of partiality and fixpoints can be deferred to the second part when we talk about PCF.

This goes some way to address Problem (b) mentioned in the Introduction, the lack of mathematical knowledge in our target audience.

Disappointingly, for the first problem — the notational complexity so typical of denotational semantics — I cannot offer a silver bullet, as that's just the way things are. However, we have an advantage when we are giving lectures as often some detail can be said rather than written down, and sometimes, it can even be suppressed entirely. Since I don't use slides but always present with the help of the board only, it is absolutely essential for me to simplify as much as possible. I'd like to propose that this is actually *a very good thing*, because students need to see how we practitioners organise the material for ourselves, how we keep on top of the notations, what we emphasise and what we neglect.

3. LECTURE I: THE SIMPLY TYPED λ-CALCULUS

I usually begin the lectures with a discussion of different approaches to semantics, relating them to the ways in which a child learns the meaning of words. The denotational approach is perhaps the most natural one: pointing to a dog and simultaneously saying the word "doggie" to the child. There is already an interesting point to be made, in that the "objects" which our chosen calculus denotes are sets and functions, and these do not have any physical representation we can point to but are a construction of our (mathematical) mind expressed in mathematical language. It follows that denotational semantics does not feel denotational at all but in essence comes down to a translation from a syntactic calculus to mathematical language. (Translation is what we use when we learn a foreign language but clearly not when we learn the first words from our parents.) I think it is important to confront this issue head on at the very beginning because it is one that tends to confuse and even confound learners when they encounter semantic definitions for the first time. One can then make the point that even straightforward translation can be useful if the languages involved are sufficiently different, and that one should feel entitled to question why the particular translation that denotational semantics offers provides any insight at all. I also stress that it is theorems we are after, not just alternative descriptions, and that the onus is on me to demonstrate that the theorems we will prove are interesting and useful.

At the Midlands Graduate School, my course is usually complemented with a concurrent one on the lambda calculus, so my introduction to the language can be brief. But even without that support, it is not too hard to present the syntax to an audience of beginning PhD students. There are, after all just two clauses to the definition of simple types and three to the definition of terms.

However, the issue of *variable binding* is much more subtle than it may appear to the students, and we know now that our best semantic accounts require a formidable mathematical apparatus. We don't yet know whether it is possible to present these in the context of an introductory lecture course, so for the time being we must stick to the tried and tested ways of keeping the problems of binding under control.

Given the constraints of a rather short lecture course, I have come to the conclusion that Church's original formalism is the most appropriate. In it each variable has a *fixed type*, which may be part of the name (as in x^σ) or which may be accessed via a global *type assignment* function: $\mathrm{type}(x) = \sigma$. The only proviso one needs to make (and the reason for it becomes clear early on in the course) is that there should be an unbounded number of variables for every type. This is the approach that one finds in [Winskel 1993, Chapter 11] and also [Plotkin 1977], for example.

While it is clear that Church's calculus is the most economical available, see Figure 1, it may still be worthwhile to spend some time here to ponder its pros and cons more generally as well as those of the alternatives.

$$\frac{}{x^\sigma : \sigma} \qquad \frac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau} \qquad \frac{M : \tau}{\lambda x^\sigma . M : \sigma \to \tau}$$

Fig. 1. The term formation rules of the simply typed lambda calculus in Church style

$$\frac{x \notin \mathrm{dom}(\Delta)}{\Gamma, x : \sigma, \Delta \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x . M : \sigma \to \tau}$$

Fig. 2. The term formation rules of the simply typed lambda calculus in Curry style

One of its disadvantages is that apart from FORTRAN it has not been adopted in any practical programming language. A more serious objection is that the approach does not generalise to more sophisticated type systems. Regarding the last point, there is little I can say in defence of my choice other than to reiterate the point that formalisms should be introduced as and when they are needed, and they should not be allowed to dominate the presentation.

If nevertheless a Curry-style presentation with explicit type environments is chosen, then it seems to me that one should go the full mile and deal with them as one does in actual programming languages and in type theory, that is, type environments are lists and later declarations of a variable override those that come earlier, a phenomenon known as *shadowing*. The machinery required to make this work is formidable but at least it is honest. One has to talk explicitly about "structural rules", that is, weakening, exchange and duplication of type assumptions although that may not be apparent from the term formation rules, Figure 2, but they will be necessary for a discussion of the equational theory.

I don't know of any textbook on denotational semantics that goes down this route, and I agree that dealing with type contexts is probably not a core concern of denotational semantics, but the various compromises that one finds are not very convincing either: One common assumption, for example, is that terms are really only representatives of their α-equivalence classes and semantics is given to the latter rather than the former. Pedagogically, this seems to me problematic as it amounts to the admission that we can't model the original concrete language, and it means that semantic considerations have encroached on the syntax and made it less realistic. Another device that appears in the literature is to say that typing contexts are *sets* and to require that $x \notin \mathrm{dom}(\Gamma)$ in the formation rule of lambda abstraction, but this is credible only if one is dealing with α-equivalence classes, as nested occurrences of the same variable may appear after β-reduction even if this was not the case at the outset.

After this digression, let us return to the material that I do present. The semantics of types is the usual one: an arbitrary set A_ι is chosen for the ground type ι and function types are interpreted by the set of all functions. Semantic environments ρ are functions from a set of variables to their respective semantic domains. The semantic function $[\![-]\!]_\rho$ is defined for those terms of the language whose set of free variables is contained in the domain of ρ.

As another aside for the cognoscenti, in the chosen presentation the semantic value of a term M of type σ in a semantic environment ρ is an *element* of the set A_σ. For the completeness proof that follows, and also for the discussion of PCF, this is the most convenient approach. One could call this the *model-theoretic* point of view. If typing contexts are made explicit, then we give semantics to *judgements* $\Gamma \vdash M : \sigma$, not to terms, and the semantic values are always *functions* from $[\![\Gamma]\!]$ to $[\![\sigma]\!] = A_\sigma$; we could call this the *categorical* point of view.

The semantic clause for lambda abstraction reads

$$[\![\lambda x : \sigma.M]\!]\rho \quad = \quad a \in A_\sigma \mapsto [\![M]\!](\rho, x \mapsto a)$$

which more obviously alludes to the mathematical view of a function as a set of ordered pairs than the "semantic lambda" that one sometimes finds. On the other hand, this plays down the (important) view that a denotational semantic function should be a *homomorphism* from the language to the model. This is a subtle point and I wish I were able to express more clearly and convincingly why I find my formulation more appropriate.

After presenting the three semantic clauses one has the first opportunity to mention the principle of *compositionality* as the semantics is pieced together from the semantic values of the parts. It is important to point out, however, that the matter is slightly more subtle than this because the semantics of a lambda abstraction $\lambda x.M$ makes reference to the semantics of M in a *different* semantic environment.

Also, this is a good time to pause and to explain the two views of the semantics that has been defined:

— If our primary interest is in the calculus, then we have given a *model* for the calculus employing sets and mathematical functions. Once we have introduced equations for the calculus, we can then ask whether the model validates the equations (*soundness*) and whether the equations capture equality in the model (*completeness*).
— If the primary interest is in mathematical functions, then we may view the simply-typed lambda calculus as a *language* for these. The natural question then is how *expressive* this language is.

4. LECTURE II: SOUNDNESS

Presenting the eight rules for deriving equalities between lambda terms is straightforward, except that one has to be able to write very fast on the board. However, not having to deal with typing contexts helps, and even the types of the terms themselves can be suppressed because there is no need to repeat the rules for typing a term. Knowing the type of a term is necessary only in the case of (η).

The rules come in three groups; first the three rules that establish that $\approx$ is an equivalence relation:

$$\frac{}{M \approx M} \qquad \frac{M \approx N}{N \approx M} \qquad \frac{M \approx N \quad N \approx P}{M \approx P}$$

Then the two congruence rules:

$$\frac{M \approx M' \quad N \approx N'}{MN \approx M'N'} \qquad \frac{M \approx M'}{\lambda x.M \approx \lambda x.M'}$$

Finally, the three rules specific to the lambda calculus:

$$\frac{y \notin \mathrm{var}(M)}{\lambda x.M \approx \lambda y.M[y/x]}\,(\alpha) \qquad \frac{}{(\lambda x.M)N \approx M[N/x]}\,(\beta) \qquad \frac{M : \sigma \to \tau \quad x^\sigma \notin \mathrm{var}(M)}{M \approx \lambda x^\sigma.Mx}\,(\eta)$$

Again, since α is not "built into" my version of the calculus, we have a rule for it; in a first course on semantics, I view this as an advantage.

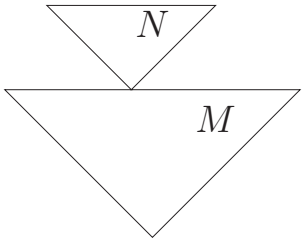
One can then start to prove the soundness theorem and find that neither (α) nor (β) have an obvious proof because substitution is not a connective of the calculus. This is yet another opportunity to return to the theme of the semantic function as a homomorphism.

I then present a precise definition of substitution for terms (taken from [Hindley and Seldin 1986]) and go through some of the cases of the proof of the substitution lemma.

The critical one is of course when a bound variable needs to be renamed, that is, we want to compute $[\![(\lambda y.M)[N/x]]\!]$ in the situation that $y \in \mathrm{FV}(N)$. The substitution rules stipulate that $(\lambda y.M)[N/x]$ be rewritten to $\lambda z.M[z/y][N/x]$ where $z \notin \{x\} \cup \mathrm{FV}(M) \cup \mathrm{FV}(N)$. The computation of the semantics of this takes a number of steps but it is satisfying for the students to see how everything fits together:

$$
\begin{array}{lll}
 & [\![\lambda z.M[z/y][N/x]]\!]\rho & \\
= & a \mapsto [\![M[z/y][N/x]]\!](\rho, z \mapsto a) & \text{(definition of } [\![\lambda \ldots]\!]) \\
= & a \mapsto [\![M[z/y]]\!](\rho, z \mapsto a, x \mapsto [\![N]\!](\rho, z \mapsto a)) & \text{(induction hypothesis, } x \neq z) \\
= & a \mapsto [\![M[z/y]]\!](\rho, z \mapsto a, x \mapsto [\![N]\!]\rho) & \text{(Lemma, } z \notin \mathrm{FV}(N)) \\
= & a \mapsto [\![M]\!](\rho, z \mapsto a, x \mapsto [\![N]\!]\rho, & \\
 & \qquad\qquad y \mapsto [\![z]\!](\rho, z \mapsto a, x \mapsto [\![N]\!]\rho)) & \text{(induction hypothesis)} \\
= & a \mapsto [\![M]\!](\rho, z \mapsto a, x \mapsto [\![N]\!]\rho, y \mapsto a) & \text{(definition of } [\![z]\!]) \\
= & a \mapsto [\![M]\!](\rho, x \mapsto [\![N]\!]\rho, y \mapsto a) & (z \notin \mathrm{FV}(M)) \\
= & [\![\lambda y.M]\!](\rho, x \mapsto [\![N]\!]\rho) & \text{(definition of } [\![\lambda \ldots]\!])
\end{array}
$$

The computation shows that the identity of the fresh variable z is immaterial for the result to hold, which goes some way to address any concerns that students might have over the fact that substitution *on terms* is not a deterministic operation. If there is time, then one can usefully insert a brief discussion of *binding diagrams* (where bound variables are replaced by arrows to the binding lambda) as an alternative "syntax." The substitution lemma is then almost a complete triviality and its proof immediately apparent from the following picture



One can use this visualisation to re-enforce the principle of compositionality.

The soundness proof is then soon completed and there is usually still time left to introduce general Henkin models, the idea being that we may not need all set-theoretic functions for the interpretation of lambda abstractions, although the students have no reason to believe this at this point. One can also come back to the view of the semantic function as a homomorphism, as Henkin models are defined as certain multi-sorted algebras with explicit application operators. (Here I am following [Mitchell 1996, Section 4.5].)

5. LECTURE III: COMPLETENESS

The aim of this lecture is to prove *Friedman's completeness theorem* which states that the full set-theoretic model built over an infinite set (for example $\mathbb{N}$) is complete with respect to $\alpha\beta\eta$-equality. It proceeds in two stages, first a term model construction is used to show that the equational rules are complete with respect to general Henkin models, and then a relation between the term model and the full set-theoretic model establishes completeness with respect to the latter.

In the construction of the term model $\mathcal{T}$ our choice of a Church-style presentation comes in handy: The "material" from which the semantic domains are built are truly the terms of our calculus, that is, T_σ consists of equivalence classes of (open) terms of type σ, and for every σ there are at least the (countably infinitely many) variables of

that type to start from. We write the equivalence classes with respect to the basic $\alpha\beta\eta$ theory as $\langle M \rangle$ and define application by

$$\mathrm{app}\ \langle M \rangle\ \langle N \rangle = \langle MN \rangle$$

One now has to show that a Henkin model is thus obtained, that is, we need to show extensionality and "richness" of the function types. For the former, we exploit the fact that we have an unbounded supply of variables, and we use the η-law:

$$\begin{array}{rcll} \forall N:\ \mathrm{app}\ \langle M \rangle\ \langle N \rangle & = & \mathrm{app}\ \langle M' \rangle\ \langle N \rangle & \text{assumption} \\ \mathrm{app}\ \langle M \rangle\ \langle x \rangle & = & \mathrm{app}\ \langle M' \rangle\ \langle x \rangle & \text{choose } N = x \text{ fresh} \\ \langle Mx \rangle & = & \langle M'x \rangle & \text{definition of app} \\ \langle \lambda x.Mx \rangle & = & \langle \lambda x.M'x \rangle & \text{congruence rule for lambda} \\ \langle M \rangle & = & \langle M' \rangle & \eta \end{array}$$

Showing that for every type $\sigma \to \tau$ we have enough elements in $A_{\sigma\to\tau}$ requires us to show that the semantics in this model amounts to simultaneous substitution:

$$[\![M]\!]\rho = \langle M[r] \rangle$$

where r is a map that picks out an element of $\rho(x)$ for every variable x in the domain of ρ. The proof is by induction on the structure of M. Here is the case of lambda abstraction: Assume $\mathrm{type}(x) = \sigma$; we have

$$[\![\lambda x.M]\!]\rho \quad = \quad \langle N \rangle \in T_\sigma \mapsto [\![M]\!](\rho, x \mapsto \langle N \rangle)$$

by definition of $[\![-]\!]$. We need to show that the function on the right hand side behaves the same as the element $\langle (\lambda x.M)[r] \rangle$ to which end we apply it to an element $\langle N \rangle \in T_\sigma$:

$$\begin{array}{rcll} & \mathrm{app}\ \langle (\lambda x.M)[r] \rangle\ \langle N \rangle & & \\ = & \mathrm{app}\ \langle \lambda x'.M[x'/x][r, x' \to x'] \rangle\ \langle N \rangle & x' \text{ fresh} & \\ = & \langle (\lambda x'.M[x'/x][r, x' \to x'])N \rangle & \text{definition of app in } \mathcal{T} & \\ = & \langle M[x'/x][r, x' \to x'][N/x'] \rangle & \beta & \\ = & \langle M[r, x \to N] \rangle & & \end{array}$$

and we see that despite our attempts to simplify things we have plenty of notational fiddliness still to deal with!

Completeness now follows easily by considering the semantic environment that maps every variable x to its $\alpha\beta\eta$ equivalence class $\langle x \rangle$.

Students are unlikely to have seen a completeness proof in full before and one should therefore take the opportunity to point out that the generality of the method. Also, although one has to be careful with the details, it is in my view a very worthwhile and satisfying exercise to see how each and every equation has its role to play in the proof.

For the second stage of the proof of Friedman's completeness theorem we return to the full set-theoretic model $\mathcal{N}$ built over $[\![\iota]\!] = \mathbb{N}$. Friedman's proof uses a "partial surjective homomorphism h" from $\mathcal{N}$ to the term model $\mathcal{T}$ but a more fruitful perspective is to regard the graph of h as a *logical relation*. So at this point I introduce logical relations between two Henkin models and prove the *fundamental lemma*, which doesn't take very long.

If there is sufficient time, one can generalise this to logical relations of arbitrary arity, and discuss the *lambda definability problem* (and Ralph Loader's undecidability result). If there isn't, then this is still a worthwhile topic to explore in the exercises.

In any case, having established the fundamental lemma for logical relations, one may now swiftly finish Friedman's proof of the completeness of $\mathcal{N}$ by showing that a logical relation between $\mathcal{N}$ and $\mathcal{T}$, which happens to be a surjective function at ground type, will be functional and surjective at every type (though it will be a *partial* function).

$$\begin{aligned} \underline{0}, \underline{1}, \ldots &: \text{int} \\ \text{succ}, \text{pred} &: \text{int} \to \text{int} \\ \text{zero?} &: \text{int} \to \text{bool} \\ \text{if}_\sigma &: \text{bool} \to \sigma \to \sigma \to \sigma \\ Y_\sigma &: (\sigma \to \sigma) \to \sigma \end{aligned}$$

Fig. 3. The constants of PCF

6. LECTURE IV: PCF

From the discussion of expressivity in the previous lecture, it is easy to reiterate the point that the simply typed lambda calculus is far too parsimonious to be considered a language for describing functions (and higher-order functions) on the natural numbers, making the step to PCF easy and natural. All the constants of PCF are readily introduced, see Figure 3. However, I usually defer talking about Y until I have presented the rewrite rules for the other constants. Again, these are easily motivated by the desire to compute normal forms. I use the small-step presentation from Plotkin's original paper [Plotkin 1977], where each rule may be seen as a refinement of an equation as discussed previously. In fact, small-step semantics appears so natural that it is worthwhile for the students to get a chance — perhaps in the exercises — to reflect on how one might implement the rewriting process in practice (and to contrast it with the algorithm derived from the big-step presentation). In the lectures we use $M \Downarrow \underline{n}$ purely as a shorthand for a finite sequence of rewrite steps that ends in $\underline{n}$.

As we arrive at the heart of the course — the semantics of recursion — we take time to discuss the constant(s) Y and how they are used to represent recursion. I begin by pointing out that the way we get a recursive program in ordinary programming languages is always by *naming terms*, something that PCF does not provide. For non-recursive terms, naming would be just a convenience — a case of Landin's *syntactic sugar* — but in a recursive term this is not so. A recursive term declaration such as

$$f = \lambda x.\ \text{if zero? } x \text{ then } \underline{1} \text{ else } x * f(\text{pred } x)$$

is really an equation for the unknown f. It is not a general equation $M = N$, but one where the left-hand side is the unknown itself, that is, it has the shape of a *fixpoint equation*. Making this explicit amounts to introducing a function on the right-hand side:

$$f = (\lambda g.\lambda x.\ \text{if zero? } x \text{ then } 1 \text{ else } x * g(\text{pred } x))\ f$$

If this function is abbreviated to M, then the fixpoint equation has the form $f = Mf$. We may now introduce YM as a *purely formal name* for the solution to this equation, just like the imaginary constant i is a purely formal name for a solution to $x^2 = -1$. Nothing needs to be known about YM other than that it may be replaced by $M(YM)$ wherever it is encountered; again, this is in analogy to the way students learned to manipulate i in high school: Nothing needs to be known about it except that i^2 may always be replaced by -1.

I believe it is very useful for students to see how this works in practice, by reducing a term such as the one above for some concrete value of x, for example, $YM\,\underline{2}$. In the exercises one can then go a bit further and ask the students to implement primitive recursive functions in PCF, and likewise μ-recursion. It is a useful insight for them to see that the former requires terms of rank 2 only, while μ-recursion requires rank 3. This opens up an interesting line of thought regarding the differences between Turing-machine computability (which the students should be familiar with) and higher-order computability.

To continue the motivational journey from recursive definitions to the constant Y, one now needs to realise that the way we treat the "formal terms" YM is captured by a rewrite rule for Y that is no different in structure from those for the other PCF constants. One then sees that the *semantics* of this constant amounts to a functional that returns a fixpoint for every (definable) endofunction. I believe it is important to spend some time on this point and to make it clear that by introducing Y we are *postulating* that every (fixpoint) equation has a solution, something that is certainly not true in mathematics. One can quite usefully contemplate why this should be the case in computer science, and one valid point of view is indeed that it doesn't hold there either.

So now that we have all components of the language, we turn to the semantics. Given the training the students have had with the simply typed lambda calculus, very little time needs to be spent on the semantics of PCF minus Y, in other words, one can cut straight to the chase and explain the difficulties one has with interpreting Y in the usual full set hierarchy $\mathcal{N}$ where functions do not need to have fixpoints.

This brings us to domain theory, introduced as an abstraction of the idea of interpreting function types as partial rather than total functions. Without going into the detail of this "partial function model," one can introduce the order between them (as containment of the function graphs) and observe that a programmable function should preserve it.

One then shows how partial functions can be replaced by total ones into a "lifted set," that is, a flat domain. But then all sets should be lifted and by looking at the Booleans, one finds that while there are only *nine* partial functions from $\mathbb{B}$ to $\mathbb{B}$, there are *11* monotone and total functions from $\mathbb{B}_\perp$ to $\mathbb{B}_\perp$. This provides an opportunity to talk about *strictness* and CBV vs CBN.

Next we look at $[\mathbb{N}_\perp \to \mathbb{N}_\perp]$, again comparing this to the partial function space $[\mathbb{N} \rightharpoonup \mathbb{N}]$ and see that its order structure is much richer. This is the moment to define abstractly the notion of an *ω-chain-complete partial order*, or *ccpo*, and that of an ω-chain-continuous function.

The four key theorems to prove are (a) that the function space of two ccpos is again a ccpo, (b) that application and abstraction are continuous functions, (c) that every continuous function has a fixpoint, and (d) that the fixpoint map itself is continuous. I am embarrassed to admit that of these I only ever manage to prove (c) in any detail. It is good material for the exercise class, though.

I stress to the students that because of (a) and (b), ccpos and ω-chain-continuous functions are the *third* example of a Henkin model for the simply typed lambda calculus, after the full set hierarchy and the term model.

As an aside, it is a fact that continuity is not necessary for the adequacy proof but it is of course much harder to prove that monotone functions have fixpoints, which would require either transfinite induction or otherwise the clever argument of Pataraia. Both can be considered in the exercises.

7. LECTURE V: ADEQUACY AND THE CONTEXT LEMMA

This lecture starts with pointing out that one should not expect an axiomatisation of the equational theory induced by the domain model for PCF, as this would entail solving the halting problem. On the other hand, the equations that we do have, all have been refined to rewrite rules, so we can talk about computation as reduction to normal form (again, it may be helpful to point out that we could have done the same for the simply typed lambda calculus). Correctness is still an issue but it can be dealt with swiftly. Completeness, on the other hand, is available at ground type only and now takes the rather interesting form of *adequacy*:

Theorem. *If P is a closed term of type int and if $[\![P]\!] = n \in \mathbb{N}$ then P reduces to $\underline{n}$ in finitely many steps.*

In class, the proof is given in full and uses the well-known "logical relation" technique (which, I am told, has indeed a long history). For a closed term M of ground type (let's restrict attention to the integers from now on) and $a \in \mathbb{N}_\perp$, one defines

$$(a, M) \in R_{\text{int}} \text{ iff } a = \perp \text{ or } (a = n \text{ and } M \Downarrow \underline{n})$$

This is extended to higher types as usual:

$$(f, M) \in R_{\sigma\to\tau} \text{ iff } \forall N : \sigma \text{ closed and } a \in A_\sigma.\ (a, N) \in R_\sigma \Longrightarrow (f(a), MN) \in R_\tau$$

We now proceed by the following simple steps:

Lemma. *$\perp\ R\ M$ always.*

Lemma. *If $M' \to M$ and $a\ R\ M$ then $a\ R\ M'$.*

Lemma. *If $a_i\ R\ M$ for the elements of a chain $a_0 \sqsubseteq a_1 \sqsubseteq \ldots$ then $\bigsqcup a_i\ R\ M$.*

It is in the proof of the last lemma where we notice that we only need that sups of functions are computed pointwise but not that functions are continuous. In other words, the proof would go through if we built our model with functions which are only required to be monotone, an observation which I learned from Alley Stoughton.

The three lemmas allow us to show the following, which has the same shape as the fundamental lemma for logical relations:

Lemma. *If M is closed, then $[\![M]\!]\ R\ M$.*

The proof is by induction over the structure of M (so we need to extend R to open terms) whereas the others are by induction over the type, the key steps being lambda abstraction and the fixpoint combinator. Adequacy then follows immediately from this and the definition of R.

The final highlight of the course is a demonstration that the semantics can be invoked to show *Milner's context lemma*. So we introduce *contexts* as "terms with holes" and the contextual preorder $\lesssim$. The key to the argument is to employ the relation R from the adequacy proof:

Lemma. *Let M, M' be closed and of type $\sigma = \sigma_1 \to \sigma_2 \to \ldots \to \sigma_k \to \text{int}$. Then the following are equivalent:*

(1) $M \lesssim M'$
(2) $\forall P$ *closed*. $PM \Downarrow \underline{n} \Longrightarrow PM' \Downarrow \underline{n}$
(3) $\forall N_1, \ldots, N_k$ *closed*.$MN_1 \ldots N_k \Downarrow \underline{n} \Longrightarrow M'N_1 \ldots N_k \Downarrow \underline{n}$
(4) $[\![M]\!]\ R\ M'$

The proof is so elegant that it is worthwhile to spell it out in full:

PROOF. (1) $\Leftrightarrow$ (2) It is possible to treat the "hole" in a context as if it were a variable because M and M' are assumed to be closed.

(3) is a special case of (2), consider $P = \lambda x.xN_1 \ldots N_k$.

(3) $\Rightarrow$ (4) Use the definition of the logical relation: Let $a_i\ R\ N_i$ for $i \in \{1, \ldots, k\}$ and consider $M'N_1 \ldots N_k$ and $[\![M]\!](a_1) \ldots (a_k)$. We want to show that $[\![M]\!](a_1) \ldots (a_k)$ is in relation to $M'N_1 \ldots N_k$. We know that $[\![M]\!]\ R\ M$ and hence $[\![M]\!](a_1) \ldots (a_k)\ R\ MN_1 \ldots N_k$. There are two cases:

— If $[\![M]\!](a_1) \ldots (a_k) = \perp$ then $[\![M]\!](a_1) \ldots (a_k)\ R\ M'N_1 \ldots N_k$ by definition.

— If $[\![M]\!](a_1)\dots(a_k) = n$ then again by the definition of R: $MN_1\dots N_k \Downarrow \underline{n}$, and hence $M'N_1\dots N_k \Downarrow \underline{n}$ by assumption. In this case, too, $[\![M]\!](a_1)\dots(a_k)\ R\ M'N_1\dots N_k$ follows.

(4) $\Rightarrow$ (2) Let P be closed. We have $[\![P]\!]\ R\ P$, hence $[\![PM]\!] = [\![P]\!]([\![M]\!])\ R\ PM'$ by assumption. Now if $PM \Downarrow \underline{n}$ then by correctness $[\![PM]\!] = n$. By the definition of R, $PM' \Downarrow \underline{n}$ follows.

The course ends by pointing the students to the classic texts by Scott [Scott 1993] and Plotkin [Plotkin 1977], as well as to the textbooks mentioned above, and especially [Streicher 2006] for a continuation of the story begun in this course.

8. CONCLUDING REMARKS

If you have taught denotational semantics yourself, you will have made different choices and you will have had different experiences. I'd love to hear from you! If you agree or disagree with the particular choices I have presented here, then I'd love to hear from you, too! Perhaps you have suggestions for making the subject more interesting or more tractable still; that would also be welcome.

I should like to end by expressing my gratitude to the many colleagues and students (especially here at Birmingham) with whom I have discussed the contents of my course over the years, and I ask for forgiveness from those from whom I have taken an idea without giving proper credit. Most of all, I am indebted to Dana Scott and Gordon Plotkin for laying the foundation for this beautiful subject.

REFERENCES

U. Berger. 1990. *Totale Objekte und Mengen in der Bereichstheorie*. Ph.D. Dissertation. Ludwig-Maximilians-Universität München.

C. Gunter. 1992. *Semantics of Programming Languages. Structures and Techniques*. MIT Press.

J. R. Hindley and J. P. Seldin. 1986. *Introduction to Combinators and λ-Calculus*. Cambridge University Press.

J.C. Mitchell. 1996. *Foundations for Programming Languages*. MIT Press.

H. R. Nielson and F. Nielson. 1991. *Semantics with Applications: A Formal Introduction for Computer Science*. Wiley.

G. D. Plotkin. 1977. LCF Considered as a Programming Language. *Theoretical Computer Science* 5 (1977), 223–255.

D. A. Schmidt. 1986. *Denotational Semantics*. Allyn and Bacon.

D. S. Scott. 1993. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science* 121 (1993), 411–440. Reprint of a manuscript written in 1969.

A. Simpson. 1998. Lazy functional algorithms for exact real functionals. In *Mathematical Foundations of Computer Science 1998 (Lecture Notes in Computer Science)*, Vol. 1450. Springer Verlag, 456–464.

Th. Streicher. 2006. *Domain-Theoretic Foundations of Functional Programming*. World Scientific. 132pp.

R. D. Tennent. 1991. *Semantics of Programming Languages*. Prentice Hall.

G. Winskel. 1993. *The Formal Semantics of Programming Languages. An Introduction*. MIT Press.

CALLS

34TH ACM SIGMOD-SIGACT-SIGAI SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS (PODS 2015)

Call for Papers
June 1-3, 2015
Melbourne, Victoria, Australia
http://www.sigmod2015.org

SERIES. The PODS symposium series, held in conjunction with the SIGMOD conference series, provides a premier annual forum for the communication of new advances in the theoretical foundation of management, traditional or non-traditional. For the 34th edition, PODS aims at broadening its scope, and calls for research papers providing original, substantial contributions along one or more of the following aspects: (1) deep theoretical exploration of topical areas central to data management; (2) new formal frameworks that aim at providing the basis for deeper theoretical investigation of important emerging issues in data management; and (3) validation of theoretical approaches from the lens of practical applicability in data management.

TOPICS. Topics that fit the interests of the symposium include the following (as they pertain to databases): design, semantics, optimization; data modeling; data structures and algorithms; tree- and graph-structured data; search; information retrieval; approximation; model theory; logics; algebras and complexity; dynamic aspects; foundations of ”big dataąnd śmall data”; data analytics; streaming, real-time, and sensor data; processes, workflows, web services; verification and synthesis; incompleteness; inconsistency; uncertainty; constraints; metadata; semantic, linked, networked, and crowdsourced data; data and knowledge integration and exchange; distribution and parallelism; cloud computing; domain-specific data; mining and learning; privacy; security; provenance.

IMPORTANT DATES.

— Cycle 1: Abstract 3 Oct 2014; Paper 10 Oct 2014; Notif 19 Dec 2014; Revised paper 30 Jan 2015; Notif 9 Mar 2015
— Cycle 2: Abstract 28 Nov 2014; Paper 5 Dec 2014; Notif 20 Feb 2015;

Camera ready for both cycles: 22 Mar 2015

EUROPEAN JOINT CONFERENCES ON THEORY AND PRACTICE OF SOFTWARE (ETAPS)

Call for Papers
April 11-19, 2015
London, UK
http://www.etaps.org

IMPORTANT DATES.

— Abstracts due: 10 October 2014 AoE
— Papers due: 17 October 2014 AoE
— Rebuttal (ESOP and FoSSaCS only): 3-5 December 2014
— Author notification: 19 December 2014

— Camera-ready versions: 16 January 2015

SUBMISSION INSTRUCTIONS. ETAPS conferences accept two types of contributions: research papers and tool demonstration papers. Both types will appear in the proceedings and have presentations during the conference. (TACAS has more categories, see below.) A condition of submission is that, if the submission is accepted, one of the authors attends the conference to give the presentation. Submitted papers must be in English presenting original research. They must be unpublished and not submitted for publication elsewhere. In particular, simultaneous submission of the same contribution to multiple ETAPS conferences is forbidden. The proceedings will be published in the Advanced Research in Computing and Software Science (ARCoSS) subline Springer's Lecture Notes in Computer Science series. Papers must follow the formatting guidelines specified by Springer at the URL `http://www.springer.de/comp/lncs/authors.html` and be submitted electronically in pdf through the Easychair (HotCRP) author interface of the respective conference. Submissions not adhering to the specified format and length may be rejected immediately.

RESEARCH PAPERS. Different ETAPS 2015 conferences have different page limits. Specifically, FASE, FOSSACS and TACAS have a page limit of 15 pages, whereas CC, ESOP and POST allow at most 20 pages. Additional material intended for reviewers but not for publication in the final version - for example, details of proofs - may be placed in a clearly marked appendix that is not included in the page limit. ETAPS reviewers are at liberty to ignore appendices and papers must be understandable without them. TACAS solicits not only regular research papers, but also case study papers.

TOOL DEMONSTRATION PAPERS. Submissions should consist of two parts. The first part, at most 4 pages, should describe the tool presented. Please include the URL of the tool (if available) and provide information that illustrates the maturity and robustness of the tool (this part will be included in the proceedings). The second part, at most 6 pages, should explain how the demonstration will be carried out and what it will show, including screen dumps and examples. (This part will be not be included in the proceedings, but will be evaluated.) ESOP and FOSSACS do not accept tool demonstration papers. In addition to tool demonstration papers (max 6 pages in their case), TACAS solicits also longer tool papers (max 15 pages) adhering to specific instructions about content and organization.

EDBT/ICDT 2015 JOINT CONFERENCE

Call for Tutorials
March 23-27, 2015
Brussels, Belgium
`http://edbticdt2015.be/index.php/edbt-icdt-call-for-tutorials`

SERIES. The EDBT series of conferences is an established and prestigious forum for the exchange of the latest research results in data management. The series of ICDT conferences provides an international forum for the communication of research advances on the theoretical foundations of database systems.

TUTORIALS. We solicit proposals for tutorials for presentation at the EDBT/ICDT joint conference. Proposals must provide an in-depth survey of the chosen area with the option of describing some particular pieces of work in detail. Proposals must be no more than four pages, in the ACM double-column format used for EDBT/ICDT submissions,

and must include enough details to provide a sense of both the scope of material to be covered and the depth to which it will be covered.

IMPORTANT DATES.

— Submission of proposals for tutorials: 7 November 2014
— Notification to authors: 12 December 2014
— Conference: March 23-27, 2015

Please submit your proposals (in PDF) via e-mail to the Tutorial Chair, Pablo Barcelo (pbarcelo@dcc.uchile.cl).

7TH NASA FORMAL METHODS SYMPOSIUM (NFM 2015)

Call for Papers
27-29 April 2015
Pasadena, California, USA
http://www.NASAFormalMethods.org/nfm2015

THEME. The widespread use and increasing complexity of mission- and safety-critical systems require advanced techniques that address their specification, verification, validation, and certification. The NASA Formal Methods Symposium is a forum for theoreticians and practitioners from academia, industry, and government, with the goals of identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems. Within NASA such systems include for example autonomous robots, separation assurance algorithms for aircraft, Next Generation Air Transportation (NextGen), and autonomous rendezvous and docking for spacecraft. Moreover, emerging paradigms such as property-based design, code generation, and safety cases are bringing with them new challenges and opportunities. The focus of the symposium will be on formal techniques, their theory, current capabilities, and limitations, as well as their application to aerospace, robotics, and other mission- and safety-critical systems in all design life-cycle stages. We encourage submissions on cross-cutting approaches marrying formal verification techniques with advances in critical system development, such as requirements generation, analysis of aerospace operational concepts, and formal methods integrated in early design stages and carrying throughout system development.

TOPICS. Topics of interest include, but are not limited to: Model checking, Theorem proving, SAT and SMT solving, Symbolic execution, Static analysis, Runtime verification, Program refinement, Compositional verification, Modeling and specification formalisms, Model-based development, Model-based testing, Requirement engineering, Formal approaches to fault tolerance, Security and intrusion detection, Applications of formal methods to aerospace systems, Applications of formal methods to cyber-physical systems, Applications of formal methods to human-machine interaction analysis.

IMPORTANT DATES.

— Paper Submission: 10 Nov 2014
— Paper Notifications: 12 Jan 2015
— Camera-ready Papers: 9 Feb 2015
— Symposium: 27-29 April 2015

LOCATION AND COST. The symposium will take place at the Hilton Hotel, Pasadena, California, USA, April 27-29, 2015. There will be no registration fee for participants. All interested individuals, including non-US citizens, are welcome to submit, to attend,

to listen to the talks, and to participate in discussions; however, all attendees must register.

SUBMISSION DETAILS. There are two categories of submissions: Regular papers describing fully developed work and complete results (15 pages) Short papers describing tools, experience reports, or descriptions of work in progress with preliminary results (6 pages) All papers should be in English and describe original work that has not been published or submitted elsewhere. All submissions will be fully reviewed by members of the Programme Committee. Papers will appear in a volume of Springer's Lecture Notes on Computer Science (LNCS), and must use LNCS style formatting. Papers should be submitted in PDF format.

PC CHAIRS.
Klaus Havelund, NASA Jet Propulsion Laboratory
Gerard Holzmann, NASA Jet Propulsion Laboratory
Rajeev Joshi, NASA Jet Propulsion Laboratory

30TH ANNUAL ACM/IEEE SYMPOSIUM ON LOGIC IN COMPUTER SCIENCE (LICS 2015)

Call for Papers
July 6-10, 2015
Kyoto, Japan
`http://lics.rwth-aachen.de/lics15/`
(colocated with ICALP 2015)

AIMS. The LICS Symposium is an annual international forum on theoretical and practical topics in computer science that relate to logic, broadly construed. We invite submissions on topics that fit under that rubric. Suggested, but not exclusive, topics of interest include: automata theory, automated deduction, categorical models and logics, concurrency and distributed computation, constraint programming, constructive mathematics, database theory, decision procedures, description logics, domain theory, finite model theory, formal aspects of program analysis, formal methods, foundations of computability, higher-order logic, lambda and combinatory calculi, linear logic, logic in artificial intelligence, logic programming, logical aspects of bioinformatics, logical aspects of computational complexity, logical aspects of quantum computation, logical frameworks, logics of programs, modal and temporal logics, model checking, probabilistic systems, process calculi, programming language semantics, proof theory, real-time systems, reasoning about security and privacy, rewriting, type systems and type theory, and verification.

INSTRUCTIONS. Authors are required to submit a paper title and a short abstract of about 100 words in advance of submitting the extended abstract of the paper. The exact deadline time on these dates is given by anywhere on earth (AoE).

— Title and Short Abstracts Due: January 12, 2015
— Extended Abstracts Due: January 19, 2015
— Author Feedback/Rebuttal Period: March 12-16, 2015
— Author Notification: March 30, 2015
— Final Versions Due for Proceedings: April 27, 2015

Deadlines are firm; late submissions will not be considered. All submissions will be electronic via `https://www.easychair.org/conferences/?conf=lics2015`. Every extended abstract must be submitted in the IEEE Proceedings 2-column 10pt format and may not be longer than 10 pages, including references. LaTeX style files are available from the website.

CONFERENCE CHAIR. Masahito Hasegawa, RIMS, Kyoto U.

PROGRAM COMMITTEE CHAIR. Catuscia Palamidessi, INRIA & E. Polytechnique

WORKSHOP CHAIR. Patricia Bouyer-Decitre, CNRS & ENS Cachan

GENERAL CHAIR. Luke Ong, U. Oxford

SHORT PRESENTATIONS. A session of short presentations, intended for descriptions of student research, works in progress, and other brief communications, is planned. These abstracts will not be published. Dates and guidelines will be posted on the conference website.

KLEENE AWARD FOR BEST STUDENT PAPER. An award in honor of the late Stephen C. Kleene will be given for the best student paper(s), as judged by the program committee.

SPECIAL ISSUES. Full versions of up to three accepted papers, to be selected by the program committee, will be invited for submission to the Journal of the ACM. Additional selected papers will be invited to a special issue of Logical Methods in Computer Science.

SPONSORSHIP. The symposium is sponsored by ACM SIGLOG and the IEEE Technical Committee on Mathematical Foundations of Computing, in cooperation with the Association for Symbolic Logic and the European Association for Theoretical Computer Science.

4TH INTERNATIONAL CONFERENCE ON TOOLS FOR TEACHING LOGIC (TTL 2015)

Call for Papers
June 1-4, 2015
Rennes, France
`http://ttl2015.irisa.fr/`

TOPICS. Topics that fit the interests of Tools for Teaching Logic include (but are not limited to): teaching logic in sciences and humanities; teaching logic at different levels of instruction (secondary education, university level, and postgraduate); didactic software; facing some difficulties concerning what to teach; international postgraduate programs; resources and challenges for e-Learning Logic; teaching Argumentation Theory, Critical Thinking and Informal Logic; teaching specific topics, such as modal logic, computability and logic, and others; dissemination of logic courseware and logic textbooks; teaching Logic Thinking.

IMPORTANT DATES.

— Paper submission: 18 Jan 2015;
— Notification: 1 Mar 2015;
— Final camera-ready due: 29 Mar 2015.

13TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING AND NON-MONOTONIC REASONING (LPNMR 2015)

Preliminary Call for Papers
Lexington, KY, USA
September 27-30, 2015
`http://lpnmr2015.mat.unical.it/`
(Collocated with the 4th Conference on Algorithmic Decision Theory 2015)

AIMS AND SCOPE. LPNMR 2015 is the thirteenth in the series of international meetings on logic programming and non-monotonic reasoning. LPNMR is a forum for exchanging ideas on declarative logic programming, non-monotonic reasoning, and knowledge representation. The aim of the conference is to facilitate interactions between researchers and practitioners interested in the design and implementation of logic-based programming languages and database systems, and those working in knowledge representation and nonmonotonic reasoning. LPNMR strives to encompass theoretical and experimental studies that have led or will lead to the construction of systems for declarative programming and knowledge representation, as well as their use in practical applications. This edition of LPNMR will feature several workshops, a special session dedicated to the 6th ASP Systems Competition, and will be collocated with the 4th Algorithmic Decision Theory Conference, ADT 2015. Joint LPNMR-ADT Doctoral Consortium will be a part of the program. Authors are invited to submit papers presenting original and unpublished research on all aspects of non-monotonic approaches in logic programming and knowledge representation. We invite submissions of both long and short papers.

TOPICS. Conference topics include, but are not limited to:

1. Foundations of LPNMR Systems
2. Implementation of LPNMR systems
3. Applications of LPNMR

SUBMISSION. LPNMR 2015 welcomes submissions of long papers (13 pages) or short papers (6 pages) in the following categories:

- Technical papers
- System descriptions
- Application descriptions

The indicated number of pages includes title page, references and figures. All submissions will be peer-reviewed and accepted papers will appear in the conference proceedings published in the Springer-Verlag Lecture Notes in Artificial Intelligence (LNAI/LNCS) series. At least one author of each accepted paper is expected to register for the conference to present the work. The Program Committee chairs are planning to arrange for the best papers to be published in a special issue of a premiere journal in the field. LPNMR 2015 will not accept any paper which, at the time of submission, is under review or has already been published or accepted for publication in a journal or another conference. Authors are also required not to submit their papers elsewhere during LPNMR's review period. However, these restrictions do not apply to previous workshops with a limited audience and without archival proceedings.

ASSOCIATED WORKSHOPS. LPNMR 2015 will include specialized workshops to be held on September 27 prior to the main conference. Currently planned workshops include:

— Grounding, Transforming, and Modularizing Theories with Variables (Organizers: Marc Denecker, Tomi Janhunen)
— Action Languages, Process Modeling, and Policy Reasoning (Organizer: Joohyung Lee)
— Natural Language Processing and Automated Reasoning (Organizers: Marcello Balduccini, Ekaterina Ovchinnikova, Peter Schueller)
— Learning and Nonmonotonic Reasoning (Organizers: Alessandra Russo and Alessandra Mileo)

IMPORTANT DATES (TENTATIVE).

— Paper registration: April 13, 2015
— Paper submission: April 20, 2015

— Notification: June 1, 2015
— Final versions due: June 15, 2015

VENUE. Lexington is a medium size, pleasant and quiet university town. It is located in the heart of the so-called Bluegrass Region in Central Kentucky. The city is surrounded by beautiful horse farms on green pastures dotted with ponds and traditional architecture stables, and small race tracks, and bordered by white or black fences. The Horse Museum is as beautifully located as it is interesting. Overall, the city has a nice feel that mixes well old and new. The conference will be held in the Hilton Lexington Downtown hotel.

GENERAL CHAIR. Victor Marek, University of Kentucky, KY, USA

PROGRAM CHAIRS.
Giovambattista Ianni, University of Calabria, Italy
Mirek Truszczynski, University of Kentucky, KY, USA

WORKSHOPS CHAIR. Yuliya Lierler, University of Nebrska at Omaha, NE, USA

PUBLICITY CHAIR. Francesco Calimeri, University of Calabria, Italy

CONTACT. `lpnmr2015@mat.unical.it`

ACM SIGLOG Logo Competition

ACM SIGLOG is in urgent need of a logo! Accordingly, the SIGLOG Publicity Committee would like to invite all members of the community to come forward with their proposals. The logo should blend well with the colour scheme of the website, the SIGLOG News cover and the ACM logo. SIGLOG should be written with all capitals.

Please send your designs to `publicity@siglog.org` by October 31st, 2014. We expect to announce the winning entry by the end of the year 2014. The winner will receive a T-shirt with the logo.

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