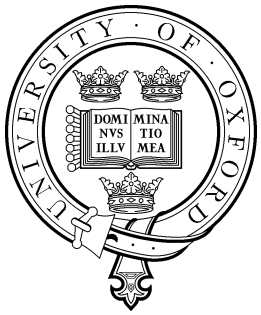
Computing Science

PROCEEDINGS OF THE OXFORD UNIVERSITY COMPUTING LABORATORY STUDENT CONFERENCE 2009

Programme Co-Chairs: Lu Feng, John Lyle, Nicolas Wu

CS-RR-09-14



Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Contents

Foreword

It is my great pleasure to welcome you to the Oxford University Computing Laboratory Student Conference, 2009. These proceedings contain the abstracts of the conference, held on Friday 20 November, 2009, at Keble College.

Traditionally, the Computing Laboratory has held this event once every two years. However, the healthy growth in graduate intake, together with the success of the conference last year, has motivated us to depart from tradition by holding the conference for a second year running.

In response to the call for papers, the Programme Committee received an encouraging number of submissions on a broad range of subjects. The abstracts contained in these proceedings are a reflection of the diversity of interests and research goals of those working in the Computing Laboratory. For some of the authors, this is their first conference submission; for others, this is the first public presentation of new work. The conference thus provides a forum for students to gain experience of presenting their research at conference, and to receive feedback on that research. Moreover, those students participating in the conference as Programme or Conference Committee members have had a taste of the rigours of conference preparation and of the refereeing process.

Each of the following 14 abstracts has been peer-reviewed by three members of the Computing Laboratory. The peer-review groups were all comprised of a mixture of DPhil students and established academic staff. The programme Committee selected the papers based on the comments of the reviews after extensive discussion.

There are many people to whom the Programme and Conference Committees would like to extend grateful thanks. We would like to warmly thank the reviewers who gave freely of their time and energy to participate in the refereeing process. We are grateful to Jeremy Gibbons for agreeing to deliver the conference keynote speech, and to Marta Kwiatkowska and Shamal Faily for offering advice on the practicalities of running a conference. Thanks are also extended to the Computing Laboratory for its support and sponsorship, without which the conference could not have taken place, and to Keble College for kindly hosting us. Finally, but by no means least, we thank the presenters for their hard work and for their contribution to such a varied and stimulating programme.

Nicolas Wu : Programme Co-Chair

Organisation

Programme Committee

Christopher Broadbent, Lu Feng, Ronald Kainda, John Lyle, Afifah Waseem, Nicolas Wu

Conference Committee

Christopher Broadbent, Sara-Jane Dunn, Jun Ho Huh, Daniel James, Jim Whitehead

Steering Committee

Marta Kwiatkowska (Honorary Chair), Shamal Faily (Past Chair)

Referees

Christopher Broadbent, Ramón Casero Cañas, Alastair Donaldson, Stephen Drape, Sara-Jane Dunn, Shamal Faily, Lu Feng, Ivan Flechais, Jeremy Gibbons, Ralf Hinze, Daniel James, Peter Jeavons, Ronald Kainda, Mark Kattenbelt, Joe Loughry, John Lyle, Andrew Martin, Toby Murray, Mehrnoosh Sadrzadeh, Nikos Tzevelekos, Jamie Vicary, Afifah Waseem, Nicolas Wu

The Use of Image Partition Forests for Automatic Spine Identification in Abdominal CT Slices

Stuart Golodetz, Irina Voiculescu and Stephen Cameron

Oxford University Computing Laboratory

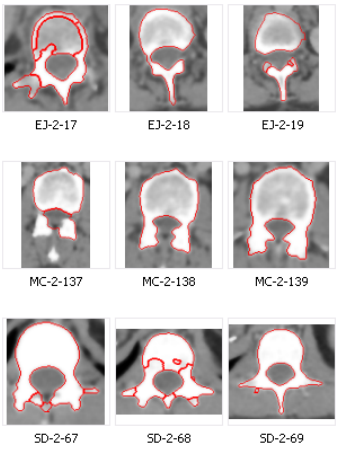


Fig. 1. Example results of our spine identification method: single red borders indicate 'outer' borders of the identified spine; double borders indicate 'inner' ones. Result MC-2-137 is an example of a failure case due to small pieces of spine which are disconnected from the primary feature.

From 3D visualization, to volume estimation, to automatic landmark-based registration, there is a plethora of medical imaging applications which rely on initially knowing where key features (e.g. organs and tumours) are to be found in medical images (CT, MRI, etc.) The process of identifying these features is difficult to automate, however, for a number of reasons:

- The boundaries in the image between adjacent features can be indistinct
- It is difficult to encode positive shape constraints for features which may differ significantly from slice to slice
- The greyscale (Hounsfield Unit, in the case of CT) distributions for distinct features often overlap, making it difficult to distinguish features by values alone

Radiologists, who are expert at reading medical scans, do not rely merely on greyscale values to tell features apart, but make use of their knowledge of anatomy to decipher an image. This anatomical knowledge can take many forms, but one of the most straightforward is localization information, i.e. knowing which features they expect to see in certain places in the image. Computer programs can equally make good use of this information to narrow down their search for a feature to a particular region of the image, or to validate the candidate features suggested by other algorithms.

To incorporate localization information into a computer program, it needs to be supplied in an image-independent way, i.e. relative to a fixed frame of reference. It makes sense to search for (say) kidneys in regions specified relative to a fixed point such as the spine; it makes far less sense to search for them in regions specified purely in image coordinates, which

have little anatomical relevance. In order to define a frame of reference, however, features such as the spine and ribs need to be automatically extracted from the image. The work we describe here focuses on automatically identifying the spine. Existing spine identification methods, e.g. [1, 2], achieve good results, but are unnecessarily complex for localization purposes, since we are only interested in using the spine to establish a coordinate system (e.g. [2] fits a four-part model to the spine, which is more than we need). By contrast, our approach is well-suited to the specific application of localization because it produces good results whilst remaining simple to understand and easy to implement.

The approach works by performing a novel multi-level region flooding algorithm on an intermediate data structure (constructed from an image) called an *image partition forest* (IPF). This is essentially a hierarchical sequence of partitions of the original image into regions which we hope are of semantic interest. The IPF construction process is based on the watershed and waterfall algorithms from mathematical morphology: the watershed algorithm is first used to construct the finest partition of the image (the lowest layer in the partition forest), and the waterfall is then used to construct a sequence of ever-coarser partitions of the image (via an iterative process of region merging) until some termination criterion is satisfied.

We needed some way to refer to the selection of a set of regions from multiple layers of the forest: this is an important part of the spine identification algorithm which follows. Our approach was to recognise that the selection of a parent node in the forest is equivalent to the selection of all of its children (because a parent node is the union of its children): this allowed us to develop novel algorithms to handle multi-layer selection, based on a minimal node representation of the selected regions, which we will present.

Having developed a concept of multi-layer selection, our region flooding algorithm then works in three stages, as follows:

1. **Seed Finding**. Traverse the forest to find regions which satisfy a user-specified *seed criterion*.
2. **Region Flooding**. Determine a *preliminary feature* (represented as a multi-layer selection) by region flooding from the various seeds.
3. **Post-Processing**. Remove any regions which were undesirably added by the flooding process (the regions to be removed are selected using a user-specified *removal criterion*).

We tested this approach on seven series of images, with good results for 87.3% of the slices tested. (Each result was visually examined and given a quality rating on the scale A = perfect, B = almost perfect, C = adequate for localization, F = failure. The percentage refers to slices which were rated A/B. The system as a whole was developed in collaboration with a radiologist.) Our method thus seems quite robust. It is worth noting that since the result is presented visually to the user (e.g. a radiologist) as a multi-layer selection, it is very easy for them to verify the output and make alterations where desired.

Our work to define a robust frame of reference for localization purposes is ongoing, but we believe our results for the intermediate step of spine identification are very promising. We will present results of our method on a number of different CT slices (e.g. Figure 1), courtesy of the Churchill Hospital, Oxford, which will show the results of automatically identifying the spine using the technique described.

References

1. Neculai Archip, Pierre-Jean Erard, Michael Egmont-Petersen, Jean-Marie Haefliger, and Jean-Francois Germond. A Knowledge-Based Approach to Automatic Detection of the Spinal Cord in CT Images. *IEEE Transactions on Medical Imaging*, 21(12), December 2002.
2. Jianhua Yao, Stacy D O'Connor, and Ronald M Summers. Automated Spinal Column Extraction and Partitioning. In *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro*, pages 390–393, 2006.

A simpler approach to waterfall

Chris Nicholls, Irina Voiculescu and Stuart Golodetz

Oxford University Computing Laboratory

This project is motivated by the need to *segment* greyscale images originating from computerised tomography (CT) scanners. That is, we would like to fit contours around organ tissue featured in each image. A pair of algorithms called the watershed and waterfall algorithms have been shown to be effective for this purpose [1], although other approaches exist. This paper presents a new, simplified, implementation of the *waterfall* algorithm.

The watershed algorithm, introduced by Beucher and Lantuejoul [2], produces a segmentation of an image, grouping together regions of pixels deemed to be similar. Usually, 'similar' refers to similarity in the greyscale values, though other approaches are possible. A typical problem of the watershed algorithm is that it over-segments images significantly, leading to far more regions than can be handled sensibly, as illustrated in Figure 1.

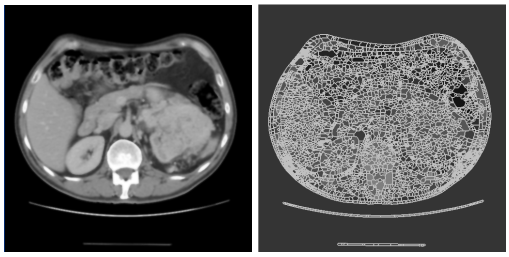


Fig. 1. Example of over-segmentation, output by applying the watershed algorithm to an axial slice of a CT volume. The individual regions are small and do not correspond to any anatomic features.

The waterfall algorithm [3, 4] is an iterative process which can extract further structure from an initial watershed segmentation. The waterfall yields a partition forest hierarchy, which is a comprehensive data structure which can subsequently be used for feature identification. Figure 2 illustrates the various layers that result from applying the waterfall algorithm to the segmentation shown in Figure 1. Each iteration of the algorithm yields a higher-level grouping of the regions in the previous layer.

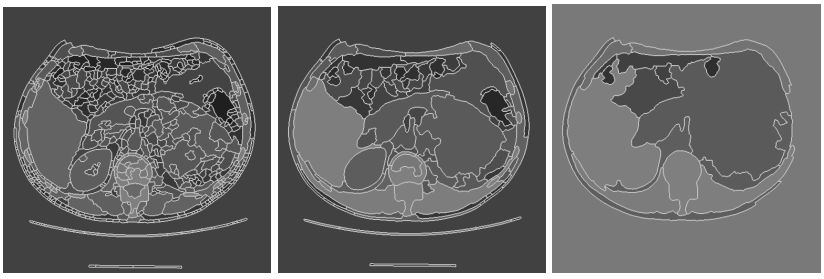


Fig. 2. Hierarchy of segmentations produced by applying the waterfall algorithm to the output of the watershed illustrated in Figure 1, showing regions merging successively (left to right).

Both the watershed and the waterfall algorithms are based on a geographical metaphor. The image is regarded as a landscape, with each grey value at representing the height of the terrain at given (x, y) coordinates. The valleys are in the darker areas, whereas the lighter areas are regarded as peaks.

The waterfall algorithm can then be imagined as a flooding process. The water falls into (low) catchment basins and gradually fills them up to the nearest boundaries, sometimes spilling into adjacent regions. This process continues until the whole image becomes a single basin. The intermediate stages of the process can be regarded as intermediate segmentations of the image, with each basin representing a region.

An implementation of this algorithm, proposed by Marcotegui and Beucher [4], involves the construction of a Minimum Spanning Tree (MST) and the gradual elision of some of its edges. Its nodes are initially the regions of the watershed output and its edges are the lowest pass points on the boundaries between these regions; the nodes and edges in subsequent layers are derived from these initial ones through a merging process.

A regional minimum edge of a graph G is part of a connected subgraph of G whose edges have the same weight as each other, and whose adjacent edges in G have strictly higher weights. The waterfall algorithm relies heavily on finding these regional minimum edges, eliding them and rebuilding the MST – a process which not only requires careful implementation of the MST but, crucially, is relatively complex and hard to implement.

In this paper we present a new data structure for the waterfall algorithm that simplifies the process and improves efficiency compared to current implementations. It is based on a recursive-tree data structure and a recursive relation on the nodes rather than the conventional iterative transformations.

The main advantage of our approach to the waterfall problem is that the algorithm uses a single loop to walk the MST and is therefore simpler to implement. For each iteration, it walks the MST bottom-up in a single pass and merges regions that belong together. The waterfall algorithm, thus improved, produces the same layers of segmented images, combined in a hierarchical structure that can be processed for feature identification.

A further advantage of our approach is that the algorithm can can be written in pure functional style. In particular, we have implemented it in Haskell. For this reason, the memory requirements are not directly comparable to existing imperative implementations, but we are about to integrate this new approach into an existing C++ code base.

We are also in the process of constructing a formal proof of correctness, which we hope to present at a later date. We have tested both algorithms on a number of small, measurable test cases and found that they produce the same output. Empirical tests indicate that this is also true of larger test cases, such as axial slices of CT volumes.

Production of partition forests in this manner is independent of this application and has many applications outside of the field of medical imaging.

References

1. Stuart Golodetz, Irina Voiculescu and Stephen Cameron. *Region Analysis of Abdominal CT Scans using Image Partition Forests.* In Proceedings of CSTST 2008 pages 432-7. October 2008.
2. S. Beucher, C. Lantuejoul. *Use of watersheds in contour detection.* International Workshop on image processing, real-time edge and motion detection/estimation, Rennes, France, Sept. 1979.
3. Serge Beucher. *Watershed, hierarchical segmentation and waterfall algorithm.* In Mathematical Morphology and its Applications to Image Processing, Proc. ISMM 94, pages 69-76, Fontainebleau, France, 1994. Kluwer Ac. Publ.
4. Beatriz Marcotegui and Serge Beucher, *Fast Implementation of Waterfall Based on Graphs.* In Mathematical Morphology: 40 Years On, Springer Netherlands, 2005.

Mapping Spatial Language to Sensor Models

an analysis of spatial cognition and its modelling in probability theory*

Jamie Frost

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Spatial cognition concerns the means by which humans realise some aspect or interpretation of objects in the spatial dimensions, whether to describe an object's motion, or the interpretation of 'locative expressions' to ascertain the identity of locations (or objects) by analysis of the environment augmented with other contextual factors. Space occupies a privileged place in language and our cognitive systems, given the necessity to conceptualise various semantic domains [1], and thus such pervasiveness is naturally imperative in the core framework of intelligent systems that aims to model such cognition. The EUROPA project (European Robotic Pedestrian Assistant) aims to produce a robot capable of solving spatial tasks assigned by pedestrians within an urban environment. Such tasks include conversing with the user to establish the identity of an object or location, escorting the user to said destinations, providing directions or semantic descriptions of locations, and responding to imperative instructions such as "take the first left". Behind the scenes, the robot will gradually accumulate spatial and semantic data with regards to its surroundings via a number of sources. The system will employ a spoken dialogue system to provide the most natural possible interface to the user.

The particular area that we focus on in the scope of this paper is that of attempting to estimate both the location and shape of an object given locative expressions concerning it, for example *"The car is just across the road, by the Computing Lab and between the two professors on the lawn."* In particular, we employ a method from robotics theory known as 'Occupancy Grid Mapping', which decomposes space into a grid of cells, each of which has some probability associated with it being occupied. This incorporates a 'sensor model'; for a robot this embodies the sensing of its environment via physical sensors. For our linguistic context, there is an analogous interpretation.

The problem can be decomposed into two parts. One is to produce probabilistic models for a variety of spatial relations such as 'between', 'by' and 'near', which govern the probability that some observer would consider the relation to be true for some point in space given the contextual information (such as the *reference objects* involved, e.g. 'B' in "A is right by B", and metric properties supplied). There has been much research in this area. [3] for example proposes a notion of 'Spatial Templates', where space is partitioned into regions of 'good', 'acceptable' and 'bad' according to their satisfaction of the spatial relation. But quantitative models previously produced (such as in [4, 5, 2]) have been largely simplistic, neglecting the topological shape of objects involved, and producing *relative* measures of applicability which fail to scale to multiple observations. We conducted an online experiment in which participants were presented with a number of scenes, and asked to assert the validity of a number of different locative expressions with respect to objects in the scene. These results were in turn used to generate the probabilistic models.

The second part of the problem is to use these models to generate the Occupancy Grid Map, and in turn, compute the approximate shape of the object. By making the (psycholin-

* The full paper can be found at http://www.jamiefrost.co.uk/research/papers/ICRA_MappingSpatialLanguageToSensorModels-JamieFrost.pdf

guistically justified) assumption that locative expressions map to a single point within the confines of the object, the occupancy probability can be computed by considering all possible poses of the object in which this point occurs. Via the use of the *level set* and *convex hull* (i.e. the minimal polygon which encloses a set of points) operations, the estimated shape based on the available evidence can be inferred.

As discussed, the spatial models can be used for a number of other uses, including the autonomous description of objects. Such a task requires a second type of spatial function, encapsulating *relevance* rather than validity alone. *"The Eiffel Tower is west of China"* is valid for example, but an irrelevant locative expression. Other future work will include expanding the functional models to encapsulate the broad spectrum of spatial language, including interpretation of parts (e.g. "The corner of the park") and groups of objects (e.g. "The cluster of trees"). There will also be extensive exploration into dialogue theory to facilitate interaction with the user.

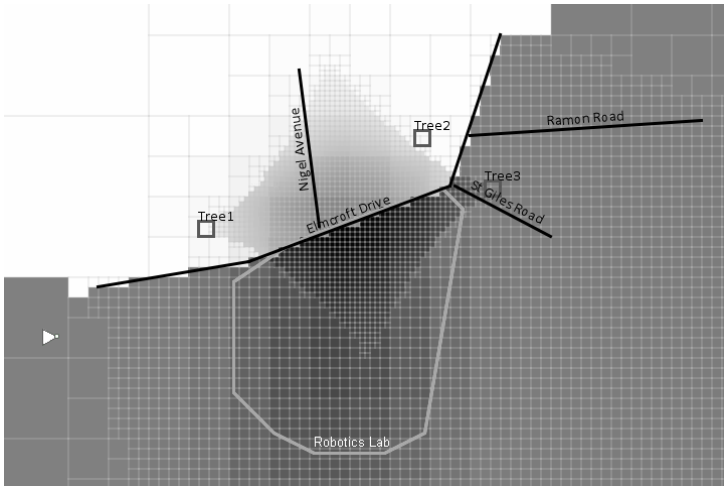


Fig. 1. The estimated shape of 'Robotics Lab' (the labelled grey outline) after the 3 observations: "The Robotics Lab is between Tree1 and Tree3, right of Elmcroft Drive, and 60 metres in front of you". The colours have been modified so that the background probability is grey, the maximum probability is black, and probabilities less than the background are whitish, indicating 'negative inference' in which regions ruled out by the observation are excluded. The observer is the white triangle to the bottom-left of the image.

References

1. Herskovits, A. *Language and Spatial Cognition.* Cambridge University Press, 1986.
2. Kelleher, J., and van Genabith, J. A computational model of the referential semantics of projective prepositions. *Syntax and Semantics of Prepositions 29* (2006), 211–228.
3. Logan, G., and Sadler, D. *Language and space.* MIT Press, 1996, ch. A computational analysis of the apprehension of spatial relations, pp. 493–529.
4. Olivier, P., and Tsujii, J.-I. Quantitative perceptual representation of prepositions semantics. *Artificial Intelligence Review 8* (2004), 147–158.
5. Yamada, A. *Studies in Spatial Descriptions Understanding Based on Geometric Constraints Satisfaction.* PhD thesis, Kyoto University, 1993.

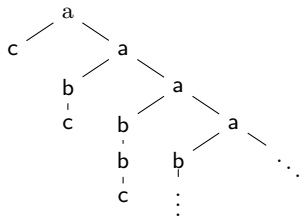
Subtyping for Model Checking Recursion Schemes

C.-H. L. Ong and S. J. Ramsay

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

We present a new type system for proving intersection-type judgements about purely-applicative terms generated from a set of constants. The system is novel in that it contains a restricted form of the usual subtyping rule for intersection types, which leads to an efficient type checking algorithm. In spite of this restriction, the system is adequate for use as the core of a sound and complete model checking algorithm for Kobayashi's resource-usage verification problem [3]. The key to the result is the relatively limited setting of the simply-typed, applicative terms of higher-order recursion schemes.

A (deterministic) higher-order recursion scheme is a definitional device for a finite or infinite tree labelled by symbols from some ranked alphabet. The tree can be seen to be generated by a process of (infinitary) rewriting using a system of simply-typed, higher-order grammar rules. Starting from some initial, non-terminal symbol, the rules are unfolded *ad infinitum*, thus determining some infinite tree. For example, we can specify a recursion scheme by giving the two grammar rules: $S \to F\,\mathsf{c}$ and $Fx \to \mathsf{a}\,x\,(F(\mathsf{b}\,x))$ and noting that the alphabet of terminal symbols is exactly a, b and c, of arities 2, 1 and 0 respectively. Rewriting from initial non-terminal symbol S produces the $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$-labelled, infinite tree depicted below:



The *$\mathcal{L}$-model checking problem* for recursion schemes is, given a formula φ of the tree-logic $\mathcal{L}$ and a tree $[\![\mathcal{G}]\!]$ defined by a recursion scheme $\mathcal{G}$: is $[\![\mathcal{G}]\!]$ a model of φ? In a 2006 LICS paper [5], Ong performed a careful analysis of the μ-calculus model checking problem for recursion schemes, from which he determined an effective procedure to decide it. Since the μ-calculus is regarded as a kind of 'gold standard' among such logics, Ong's result opened the door to serious application of recursion schemes in computer-aided verification, where they seem to have a great deal of potential. Not only are recursion schemes a particularly expressive device for defining infinite trees (which can be employed to abstractly model the computation trees of programs) but their higher-order grammar rules are already a close-fit to the declarative structure of higher-order functional programs.

Kobayashi has been successfully pursuing this direction with respect to the *resource-usage verification problem* for an extension of the simply-typed λ-calculus. In his 2009 POPL paper [3], he showed how the resource-usage verification problem in this setting can be reduced to the model checking problem for recursion schemes and the class of deterministic, trivial tree-automata (which take the place of the logic). In the same paper, Kobayashi gave a novel decision procedure for this model checking problem, reducing its solution to a

type inference problem in an intersection type system. The model checking problem has a positive solution if and only if the recursion scheme is typable in an intersection type system determined by the automaton. The procedure searches for types in order to provide a witness to the claim, employing type checking to assess the correctness of candidate solutions.

Later in the same year Kobayashi presented a refinement of this algorithm that was highly optimised in order to make it practically efficient [2]. This work was borne out by a tool, TRecS, that implemented the algorithm and was used to verify a variety of example recursion schemes, derived both from verification literature and OCaml source code. Kobayashi noted that, in order to scale effectively at higher orders, it was necessary to leverage the natural subtype ordering on the intersection types so that the search space for inference could be pruned wherever possible. Our work elaborates upon this note, presenting a new intersection type system that allows reasoning about subtype inequalities whilst retaining an efficient type checking algorithm.

Intersection type systems extend the vocabulary of simple type systems by introducing an additional connective '$\wedge$' to denote the intersection of two types. Hindley gives a good introduction to the subject in [1]. Roughly speaking, an inhabitant of a type $\sigma \wedge \tau$ can be seen to inhabit both σ and τ. This interpretation gives an obvious subtype ordering with, for example, $\sigma \wedge \tau \leq \sigma$, $\sigma \wedge \tau \leq \tau$ and $\sigma \to \tau \leq \sigma' \to \tau'$ just if $\sigma' \leq \sigma$ and $\tau \leq \tau'$. Although in the literature there have been many formulations of intersection type systems which appeal to the subtype ordering in order to construct derivations, the emphasis is rarely on defining systems that admit efficient type checking. By considering only subtyping with respect to the purely applicative terms (which is made possible in the setting of recursion schemes) we were able to design a somewhat more restricted type system than those previously reported – allowing instances of subtyping only in arguments at application – but which retains the properties desirable for model checking.

Our goal was to restrict the form of typing derivations so that they became amenable to proof search. In fact, the system has a strong form of the subformula property, which gives rise to a short, recursive algorithm for type checking. When the time to decide the subtype relation is bounded above by a constant, the procedure's worst case time complexity is linear in the product of the size of the type to be checked and the size of the typing environment. Furthermore, a type-checking algorithm for Kobayashi's original system (without subtyping) can be seen as an instance of the same scheme when subtyping is replaced by 'subsetting'.

We hope to integrate the type checking algorithm into our own implementation of Kobayashi's algorithm, due to Nistor [4], which is to serve as a test-bed for a number of other experimental extensions we have planned for the future.

References

1. Hindley, J. R. Types with intersection: An introduction. *Formal Aspects of Computing 4* (1992), 470–486.
2. Kobayashi, N. Model-checking higher-order functions. In *PPDP '09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming* (New York, NY, USA, 2009), ACM, pp. 25–36.
3. Kobayashi, N. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA, 2009), ACM, pp. 416–428.
4. Nistor, L. N. Model-checking higher-order recursion schemes. Master's thesis, Oxford University Computing Laboratory, 2009.
5. Ong, C.-H. L. On model-checking trees generated by higher-order recursion schemes. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science* (Los Alamitos, CA, USA, 2006), IEEE Computer Society, pp. 81–90.

HOMER: a Higher-order Observational equivalence Model checkER

David Hopkins

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

1 Introduction

We present HOMER, [4], an observational equivalence checker for higher-order programs. *Observational equivalence* is a compelling notion of program equivalence. Two programs are considered equivalent just if, in every program context, substituting one term for the other does not result in any observable difference in the computational outcome. Our language of study is Idealized Algol (IA), [8], a prototypical programming language combining both functional and imperative features. IA can be thought of as a call-by-name variant of core ML. In particular we consider the third-order fragment of IA, including iteration. Perhaps surprisingly, observational equivalence of this fragment is decidable, [5]. (However, at fourth-order observational equivalence becomes undecidable, [6].)

Example 1 *Consider the programs* $P_1 \equiv \mathsf{new}\, x := 0 \,\mathsf{in}\, \{p\,(x := 1);\ \mathsf{if}\ !x = 1\ \mathsf{then}\ \Omega\ \mathsf{else}\ \mathsf{skip}\}$ *and* $P_2 \equiv p\,\Omega$. *Here* Ω *is the immediately diverging program and* p *is some function (to be defined by the context) which takes a command as an argument and returns another command. In both* P_1 *and* P_2, *if* p *ever runs its argument, then the computation as a whole will diverge. However, if* p *terminates without evaluating its argument then the computation terminates. Hence, the two programs are observationally equivalent. This shows that there is no program which can evaluate its argument and then undo all of its side effects.*

Example 2 *Consider the programs*

$$\begin{aligned} M_1 &\equiv \mathsf{fun}\, F\ :\ (exp \to exp) \to exp\,. \\ &\qquad \mathsf{new}\, x \,\mathsf{in}\, \{F\,(\mathsf{fun}\, y :\ exp\,.\, \mathsf{if}\ !x = 0\ \mathsf{then}\ x := y\ \mathsf{else}\ x := y - 1;\ !x)\} \\ M_2 &\equiv \mathsf{fun}\, F\ :\ (exp \to exp) \to exp\,. \\ &\qquad F\,(\mathsf{fun}\, y\ :\ exp\,.\, \mathsf{new}\, x \,\mathsf{in}\, \{\mathsf{if}\ !x = 0\ \mathsf{then}\ x := y\ \mathsf{else}\ x := y - 1;\ !x)\} \\ M_3 &\equiv \mathsf{fun}\, F\ :\ (exp \to exp) \to exp\,.\, F\,(\mathsf{fun}\, y\ :\ exp\,.\, y) \end{aligned}$$

In M_1 *the value of* x *will be maintained if* F *calls its argument multiple times. However, in* M_2 *its value will be reset to* 0 *each time. So in* M_2 *the guard is always true and so the inner function acts as the identity. Hence* $M_1 \ncong M_2 \cong M_3$. *This shows that scope extrusion fails in IA.*

2 Homer

HOMER takes two IA programs and checks if they are observationally equivalent. It works by compiling programs into *Visibly Pushdown Automata* (VPA), [2], which are a precise representation of the program's *fully abstract game semantics*, [1]. The tool then passes the resulting automata to a VPA toolkit which we have implemented to test for their equivalence.

Thanks to the fully abstract semantics, it is known that observational equivalence reduces to the VPA Equivalence Problem. Our checker is thus sound and complete; because it model checks open terms, our approach is compositional. Further, if the two terms are not equivalent, HOMER will produce both a game-semantic and an operational-semantic counter-example, in the form of a play and a separating context respectively. It can also model check regular properties (such as those describable by LTL formulas) by checking if the language of the VPA-translate of the program is included in that of the formula, provided as a regular expression. To our knowledge, HOMER is the first implementation of a model checker for 3rd-order programs.

3 Extensions

HOMER works very well on small examples but runs into the state space explosion problem on larger programs. One approach to overcoming this, which worked well for a similar tool, MAGE, [3], is to use laziness. This is particularly effective when trying to solve reachability problems rather than equivalence as we may be able to get away with only partially constructing the model. Directly using the techniques from MAGE is not possible, since it only deals with finite automata whereas HOMER works with VPA. The presence of a stack significantly increases the difficulty of a lazy compilation, as it is not obvious whether a particular pop-transition can ever be taken when you are only presented with local information. However, by carefully analysing the compilation and the shape of the resulting VPA, we have come up with a lazy algorithm which significantly speeds up HOMER's reachability testing.

Finally an interesting challenge is switching to call-by-value in order to get a language closer to standard ML. Call-by-value IA, known as RML, is rather trickier to reason about than the call-by-name version. For example, in normal IA, every time a new variable is declared it has a definite scope. However, just by converting to call-by-value allows us to pass variable locations out of the scope they are declared in. This complication leads to undecidability with much smaller language fragments than in IA. Even at order two, observational equivalence is undecidable, [7]. For a small fragment of second-order RML, it is known that the problem can be reduced to that of equivalence of regular expressions. We believe we can utilise the power of VPA (and possibly DPDA, [9]) to extend this result and hope to integrate the results into HOMER.

References

1. ABRAMSKY, S., AND MCCUSKER, G. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. In *Algol-like languages, Birkhauser* (1997).
2. ALUR, R., AND MADHUSUDAN, P. Visibly pushdown languages. In *STOC* (2004).
3. BAKEWELL, A., AND GHICA, D. R. On-the-fly techniques for game-based software model checking. In *TACAS* (2008).
4. HOPKINS, D., AND ONG, C. H. L. Homer: A higher-order observational equivalence model checker. In *CAV* (2009).
5. MURAWSKI, A., AND WALUKIEWICZ, I. Third-order idealized algol with iteration is decidable. In *FOSSACS* (2005).
6. MURAWSKI, A. S. On program equivalence in languages with ground-type references. In *LICS* (2003).
7. MURAWSKI, A. S. Functions with local state: regularity and undecidability. *Theor. Comput. Sci.* (2005).
8. REYNOLDS, J. C. The essence of Algol. In *Algorithmic Languages* (1981).
9. STIRLING, C. Decidability of DPDA equivalence. *Theor. Comput. Sci.* (2001).

Faster FDR Counterexample Generation Using SAT-Solving*

Hristina Palikareva**

Oxford University Computing Laboratory, Oxford, UK

1 Introduction

Model checking techniques can be partitioned into those which are *symbolic*, based on abstract representation of sets of states, and those which are based on *explicit* examination of individual states. The former generally represent sets of states as Boolean formulae and use techniques such as SAT-solving and BDD manipulation to check properties. The latter can be enhanced by techniques such as partial-order reductions, CEGAR, hierarchical state-space compression, etc. The main obstacle when applying both approaches in practice is the *state-space explosion problem* by which the number of states in a concurrent system grows exponentially with the number of its parallel components.

The general problem we investigate is refinement checking in the process algebraic settings and, more specifically, in the context of CSP [Hoa85,Ros98]. Unlike in conventional model checking, where specifications are generally defined as formulae in some kind of temporal logic, in process algebras both specifications and implementations are modelled as processes. Refinement checking reduces then to checking for reverse containment of behaviours and, therefore, to reverse language inclusion.

FDR [Ros94,G+05] is a well-established tool for refinement checking of CSP. When deciding whether an implementation process *Impl* refines a normalised specification process *Spec*, FDR traverses the Cartesian product of the state spaces of *Spec* and *Impl* in a BFS manner, checking for compatibility of mutually reachable states. Hence, until now, FDR has followed the explicit model checking approach. There has been, however, some work on symbolic model checking of CSP [PY96,SLDS08], based on compositional encoding of CSP processes.

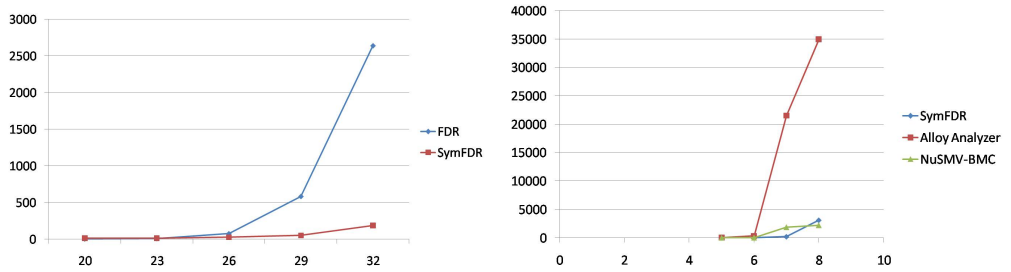
2 Bounded Refinement Checking

Bounded model checking (BMC) [BCCZ99] is a flourishing symbolic model checking technique that focuses on searching for counterexamples of bounded length only. The underlying idea is to fix a bound k and unwind the model for k steps, thus considering behaviours and counterexamples of length at most k. In practice, BMC is conducted iteratively by progressively increasing k until a counterexample is detected, k reaches a precomputed *completeness threshold* indicating that the model satisfies the specification, or the problem becomes intractable. At each step, the corresponding part of the model is encoded into a SAT instance and counterexamples, if any, are generated using SAT-solving techniques. Note that without knowing the completeness threshold, the BMC algorithm is incomplete. Hence, BMC is mostly suitable to detecting bugs, not to full verification (proving absence of bugs).

We address the problem of applying BMC to concurrent systems involving the interaction of multiple processes running in parallel. We adapt the BMC framework to the context of CSP and FDR, yielding *bounded refinement checking*. Therefore, we exploit the SAT-solver to decide bounded language inclusion as opposed to bounded reachability of error states, as in most existing model checkers. Within the scope of this talk, we only consider the translation of *trace* refinement to SAT checking. We propose a Boolean encoding of CSP processes resting on FDR's hybrid two-level approach for calculating the operational semantics using supercombinators. Since the original syntactic translation of BMC to SAT cannot be applied directly to the context of CSP, we present a semantic translation algorithm based on watchdog transformations [RGM+03].

*Based on a paper presented at AVoCS'09, see http://web.comlab.ox.ac.uk/people/Hristina.Palikareva/publications/avocs_09.pdf

**Joint work with Joel Ouaknine and Bill Roscoe



(a) Peg Solitaire

(b) The Hanoi Towers — comparing against other BMC tools

Fig. 1. Experimental Results

3 Experimental Results

We have developed a prototype tool SymFDR which, when combined with state-of-the-art SAT-solvers such as MiniSAT [ES03], sometimes outperforms FDR by a significant margin when finding counterexamples. We compare the performance of SymFDR with that of FDR, FDR used in a bounded DFS mode, PAT [SLD08] and, in some cases, NuSMV, Alloy Analyzer and straight SAT encodings. On some test cases, such as peg solitaire and the chess knight tour, SymFDR's performance is very encouraging, coping with problems that are beyond FDR's capabilities. In general, though, FDR outperforms SymFDR, particularly when a counterexample does not exist. We conclude that SymFDR is likely to outperform FDR in large combinatorial problems for which a solution exists, the length of the longest solution is relatively short (growing at most polynomially) and is predictable in advance. The search space of those problems can be characterised as very wide (with respect to BFS), but relatively shallow. We suspect that problems with multiple solutions also induce good SAT performance. Experiments with the towers of Hanoi suggest that SAT-solving techniques offer advantages up to a certain threshold and weaken afterwards.

4 Future Work

We envision several directions for future work. We intend to implement McMillan's algorithm combining SAT and interpolation techniques to yield complete unbounded refinement checking [McM03]. This method has proven to be more efficient for positive BMC instances (instances with no counterexamples) than other SAT approaches. Other avenues for further enhancing FDR's performance include partial-order reductions and CEGAR.

References

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, 1999.

[ES03] Niklas Een and Niklas Sorensson. An extensible SAT-solver. In *SAT*, 2003.

[G+05] Michael Goldsmith et al. *Failures-Divergence Refinement. FDR2 User Manual.* Formal Systems (Europe) Ltd., June 2005.

[Hoa85] C. A. R. Hoare. Communicating sequential processes. *Comm. of the ACM*, 21, 1985.

[McM03] Kenneth L. McMillan. Interpolation and SAT-based model checking. In *CAV*, 2003.

[PY96] A. Parashkevov and J. Yantchev. ARC - a tool for efficient refinement and equivalence checking for CSP. In *AAPP*, 1996.

[RGM+03] A. W. Roscoe, M.H. Goldsmith, N. Moffat, T. Whitworth, and I. Zakiuddin. Watchdog transformations for property-oriented model checking. In *FME*, 2003.

[Ros94] A. W. Roscoe. *Model-checking CSP*, chapter 21. Prentice-Hall, 1994.

[Ros98] A. W. Roscoe. *The theory and practice of concurrency.* Prentice Hall, 1998.

[SLD08] Jun Sun, Yang Liu, and Jin Song Dong. Model checking CSP revisited: Introducing a process analysis toolkit. In *ISoLA*, 2008.

[SLDS08] Jun Sun, Yang Liu, Jin Song Dong, and Jing Sun. Bounded model checking of compositional processes. In *TASE*, 2008.

Reachability in Parametric One-Counter Automata

Christoph Haase

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Intoduction

The ubiquitousness of computer systems in our everyday life means that we rely more and more on their functional correctness. In particular in application areas such as air or road traffic, we cannot accept faulty systems as they can cause fatal consequences when taking responsibilities for the life and health of humans. *Model checking* [1] is a verification technology that came up in the last 25 years and has successfully been used to verify systems on an industrial scale. Its biggest advantage in comparison to other technologies is that it provides a way to *automatically prove the absence of errors* in computer systems. In order to model check a system, one constructs a conservative abstraction of it and provides the property to be verified in some specification language. Such a property could for example be that there is no trace in the system that reaches a bad state starting from an inital state. A model checker can be used to algorithmically prove or disprove a property on the abstract model. In case the property does not hold, the model checker provides a counter example that shows an erroneous behavior of the system.

Traditionally, an abstracted system consists of a finite number of states and configurations and allows for checking specifications regarding the relative order in which events in the original system occur. Although this abstraction is sufficient for checking most safety properties, it is too coarse when one is interested in verifying quantitative properties of a system. To overcome this drawback, extensions of finite state systems with timing information, weights or probabilities have been proposed and successfully applied in real-world applications. The main challenge in verifying these extended systems is that their set of possible configurations often becomes infinite, which makes it impossible to algorithmically prove properties by naively exploring all possible configurations.

In this paper we are going to give a short account on some recent work on the computational complexity of deciding reachability in parametric one-counter automata. Such automata extend classical non-deterministic finite state automata with one counter over the natural numbers. When taking a transition, the counter can be tested for zero, or incremented or decremented by some integer or parametric value. The reachability problem asks, whether there exists a valuation of the parameters such that a final configuration is reachable from an initial configuration. One-counter automata can be used to model systems that consume goods, e.g. energy or dollars, and whose concrete behaviour depends on parameters provided by the environment. Determining a reachability problem can then be seen as an answer to the question whether there exists an environment that allows for reaching some good or bad configuration in the system.

The interested reader is refered to [2] for the full version of this paper.

Parametric One-Counter Automata

A *parametric one-counter automaton* $\mathcal{A}$ is a tuple $\mathcal{A} = \langle Q, \Delta, P \rangle$, where Q is the finite set of *states* and P is a finite set of *parameters*. The *transition relation* is $\Delta \subseteq Q \times COp \times Q$,

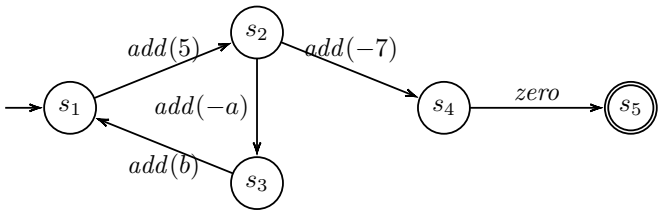


Fig. 1. An example of a one-counter automaton. Instantiating a with 4 and b with 1 would allow for reaching the configuration $(s_5, 0)$ starting from the configuration $(s_1, 0)$.

where $COp := \{add(z) : z \in \mathbb{Z}\} \cup \{add(p) : p \in P\} \cup \{add(-p) : p \in P\} \cup \{zero\}$ is the set of *counter operations*. The *add* operations can add some integer or parametric value to the counter and the *zero* operation can test the counter value for zero.

A *valuation* $v : P \to \mathbb{Z}$ assigns to each parameter some integer value. Given a one-counter automaton $\mathcal{A}$ and a valuation v, a *run* from $q_1 \in Q$ to $q_n \in Q$ is a sequence of *configurations* $(q_1, 0) \to (q_2, c_2) \to \cdots \to (q_n, 0)$ such that every c_i is greater or equal to zero and for each pair $(q_i, c_i), (q_{i+1}, c_{i+1})$ there is some $(q_i, cop, q_{i+1}) \in \Delta$ such that $c_{i+1} = c_i + z$ if $cop = add(z), z \in \mathbb{Z}$; $c_{i+1} = c_i + v(p)$ if $cop = add(p)$, $c_{i+1} = c_i - v(p)$ if $cop = add(-p)$, $p \in P$; and $c_i = c_{i+1} = 0$ if $cop = zero$ for $1 \leq i < n$. As an example, consider the one-counter automaton shown in Figure 1. Setting $v(a) := 4$ and $v(b) := 1$ would allow for the following run: $(s_1, 0) \to (s_2, 5) \to (s_3, 1) \to (s_1, 2) \to (s_2, 7) \to (s_4, 0) \to (s_5, 0)$.

Given two states $q_i, q_f \in Q$, the reachability problem is to determine whether there exists a valuation of the parameters such that there exists a run starting in $(q_i, 0)$ and ending in $(q_f, 0)$. Observe that there is no a priori bound on the values the parameters can take, which makes it hard to decide when to "stop looking for possible values." Nevertheless, it follows from [2] that the problem is decidable, even with a relatively low complexity:

Theorem 1. *The reachability problem for parametric one-counter automata is* NP*-complete.*

An explanation of this result would go beyond the scope of this short article. Roughly speaking, one can find a general pattern in every run of any one-counter automaton that allows for translating the reachability problem into a formula of polynomial size in the existential fragment of Presburger arithmetic [3]. Satisfiability in this fragment is NP-complete, which gives the upper bound. The lower bound follows from a reduction from the SubsetSum problem, when numbers are encoded in binary and even if there are no parameters involved.

Regarding future work, we would like to investigate the decidability and complexity of model checking specifications written in temporal logics such as CTL against parametric one-counter automata.

References

1. Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking*. The MIT Press, January 1999.
2. Haase, C., Kreutzer, S., Ouaknine, J., and Worrell, J. Reachability in succinct and parametric one-counter automata. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR09)* (September 2009), M. Bravetti and G. Zavattaro, Eds., vol. 5710 of *Lecture Notes in Computer Science*, Springer, pp. 369–383.
3. Lipshitz, L. The diophantine problem for addition and divisibility. *Transaction of the American Mathematical Society 235* (1976), 271–283.

Investigating Formal Database Design Using Z and Templates

Nicolas Wu

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Relational databases [2] that support querying systems such as SQL have become commonplace in managing large amounts of data in nearly every context where the storage and retrieval of information is required. The design of these databases typically involves the use of diagramatic representations of the database, and relationship-entity diagrams are often used to capture the way in which data is related within the database. Although good at capturing this aspect of a database specification, these approaches do little to assist in the expression of constraints that may exist within the system, and the automatic verification of constraint satisfaction is, at best, an afterthought in these frameworks.

Our research is concerned with the accurate representation of integrity constraints as predicates in a notation that supports the reasoning of those constraints. Capturing integrity constraints is in itself useful, but doing so in a formal language that allows the mechanical analysis and manipulation of those conditions has the potential to provide assurance of certain properties of the database at the design stage, rather than through consistency checks before transactions commit. To this end, the Z notation [8, 9] is a respectable candidate as a means of working with constraints, since Z schemas have been shown to effectively capture these constraints and verify that they are sound, and both have their roots in predicate calculus and set theory. Indeed, several authors have made use of this connection in an educational context [5, 3], by leveraging not only the logical and philosophical common ground of the two notations, but also the structural similarities that they share.

The common form of a database schema described in Z might be as in the following example, where the schema $BOOK$ serves as a constrained description of a particular book instance, and $Book$ represents a relational schema with constraints on the primary key:

$$BOOK \mathrel{\widehat=} [book_{id} : \mathbb{N};\ title : TITLE;\ author : NAME;\ copies : \mathbb{N};\ missing : \mathbb{N} \mid copies > missing]$$

$$Book \mathrel{\widehat=} [books : \mathbb{P}\, BOOK \mid \forall\, b_1, b_2 : books \bullet b_1.book_{id} = b_2.book_{id} \Rightarrow b_1 = b_2]$$

Since the design of the relational model of data finds its roots in predicate logic and set theory—foundations that are shared with the formal specification notation Z—it is of little surprise that the two make such a good pairing. Indeed, significant research in this area has already been undertaken by several authors [6, 4].

Despite the formality of the models presented in the literature, one problem remains with this approach: the specification at the meta-model level of the schemas above cannot be formalised in Z in a way that preserves the syntactic similarity between the database schemas and the representation in Z, since this would require a higher-order logic [7]. The problem with our solution is that our first try at a description has indicated a means of producing database specifications in Z, and we have done so by providing an *instance* of a set of schemas to describe the overall methodology. Although this instance is formal, and descriptive enough for a human to understand and apply in other contexts, there has been no formal description of the *general* shape of schemas that that should be used for arbitrary relations and databases. Our formal description is too ambiguous for an automated system

to generate relation schemas from some parameterised environment. This is a limitation that all other presentations of database design using Z suffer from. To solve this problem, we suggest the use of a templating system in order to generalise our description through extensive parameterisation.

The Formal Template Language [1], captures the form of sentence fragments in a formal way that allows general instantiation, and supports proofs. Templates allow parametricity without requiring a whole new framework of semantics to be imposed on Z itself. We illustrate the use of the FTL through its application to database design, where the general description of the relation, like the one we had above, can captured by the following templates:

$$\ll RELATION \gg \mathrel{\widehat=} [[\![\ll attribute \gg : \ll ATTRIBUTE \gg]\!] \mid \ll constraint \gg]$$

$$\ll Relation \gg \mathrel{\widehat=} [\ll relation \gg s : \mathbb{P} \ll RELATION \gg \\ \mid \forall r_1, r_2 : \ll relation \gg s \bullet r_1.\ll key \gg = r_2.\ll key \gg \Rightarrow r_1 = r_2]$$

Here, placeholders are delineated by special brackets, written $\ll$ and $\gg$, which indicate identifiers that are to be replaced by fragments of text as indicated in an *instantiation environment*. Lists of placeholders, written between $[\![$ and $]\!]$, allow the repetition of template fragments with different mappings for particular placeholders. This has allowed us to explicitly indicate the parts of the schema that are to be specialised for a solution.

As we have already mentioned, a full database specification would include many more schemas than just the relational intention and extension: further schemas are required to instantiate state, and provide operations. By using a carefully constructed set of templates, much overhead can be avoided since schemas with essentially the same structure can be instantiated automatically. Further tedium can be avoided since the FTL can also be used in conjunction with its instantiation calculus to generalise proof obligations with templates. This application of the FTL is novel, and significantly helps in the design of database models in a framework that encourages the use of constraints by design.

References

1. Amálio, N., Stepney, S., and Polack, F. A Formal Template Language Enabling Metaproof. In *Proceedings of FM 2006* (2006), J. Misra, T. Nipkow, and E. Sekerinski, Eds., vol. 4085 of *Lecture Notes in Computer Science*, Springer, pp. 252–267.
2. Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM 13*, 6 (1970), 377–387.
3. Davies, J. W. M., Simpson, A. C., and Martin, A. P. Teaching Formal Methods in Context. In *Proceedings of CoLogNET/FME Symposium on Teaching Formal Methods* (2004), vol. 3294 of *Lecture Notes in Computer Science*, Springer, pp. 185–202.
4. de Barros, R. S. M. Deriving Relational Database Programs from Formal Specifications. In *Proceedings of FME '94: Industrial Benefit of Formal Methods*, vol. 873 of *Lecture Notes in Computer Science*. Springer, 1994, pp. 703–703.
5. Edmond, D. *Information Modeling: Specification and Implementation.* Prentice-Hall, 1992.
6. Gray, D. The Formal Specification of a Small Bookshop Information System. *IEEE Transactions on Software Engineering 14*, 2 (1988), 263–272.
7. Martin, A. P., and Simpson, A. C. Generalising the Z schema calculus: database schemas and beyond. In *Proceedings of APSEC 2003* (2003), pp. 28–37.
8. Spivey, J. M. *The Z notation: A Reference Manual.* Prentice-Hall, 1992.
9. Woodcock, J. C. P., and Davies, J. W. M. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall, 1996.

Checking Model Consistency using Data-Flow Testing

Chen-Wei Wang

Oxford University Computing Laboratory

An information system is a computing system for the collection and provision of data. Such systems typically hold large amounts of business data. The meaning of these stored data are given through subtle relationships with other data and sophisticated business rules. It is critical that this meaning is properly maintained as the stored data are updated and transformed. If there is a loss of this kind of semantic integrity, then the value of the information and the system itself may be sharply diminished.

To facilitate the development process essential design decisions, including entity associations and semantic integrity constraints, may be usefully captured in a model. We may evaluate the current research from the aspect of how the model is synchronised with its implementation. One approach is *model-based development*, where the synchronisation is done manually upon changes in either the model or the implementation. Contractual models—invariant, pre-, and post-conditions—are documented to express developers' intents, and the corresponding implementation (in, e.g. Java [1] or C# [2]) should be constructed in such a way that the system behaviour satisfies the contracts. However, in order to verify formally that the implementation conforms to the model, a large number of proof obligations are required to be discharged by a theorem prover. An alternative approach is *model-driven development* [3], where the synchronisation is automatic. Model operations are specified as simple data transformations instead of complicated algorithms. Changes are only made to the model as requirements evolve, and a conforming implementation can always be automatically generated through a series of model transformations, based upon assumptions and pre-defined rules about the specific problem domain.

However, if the various structural and integrity constraints expressed in the model are inconsistent, then either the model will admit no implementation, or the implementation produced will not behave according to the intended requirements. In this paper, we are concerned about this issue of model consistency. It is not practical to conduct an exhaustive exploration of the model state space, nor to consider all possible interactions between operations. Instead, we concentrate on inter-method usage patterns of attributes, captured in a set of method call sequences.

We will build upon existing work on Booster [4], a language and compiler based upon B [5] and Z [6], for the automatic generation of object data stores. In Booster, the model interface is object-oriented with features of classes, attributes, and associations. We use first-order predicates to capture business requirements as class invariants, and to describe intended model operations, i.e. methods, as relational constraints between the pre- and post-execution values of attributes. A Booster method may also build upon the intents of other methods by referencing their names.

Our approach consists of two steps. First, we identify for each attribute `att` call sequences representing `att`'s various inter-method usage patterns, by applying the data-flow testing technique to all methods defined in a given Booster model. Second, we derive a test suite from these call sequences. Each test case is represented as a Boolean constraint, ensuring that invoking the call sequence both demonstrates an inter-method usage of `att` and maintains the system integrity. We then rely upon the developers to inspect this test suite of constraints. More precisely, they can examine whether each test case constraint is either too weak, too strong, or consistent, with respect to their original understandings about the requirements. They may then take actions accordingly to fix methods or invariants.

As an example, say the value of an attribute `att` is defined in a method `m1`

```
m1 {x > 0 => att' = x + 1}
```

and its value is used in another method `m2`

```
m2 {y > 0 => z' = z - att}
```

then we can derive a test case `x > 0 & y > 0 & z > att` for the inter-method usage pattern of `att`, where the first two conjuncts ensure that the def and use patterns are covered by the execution path, and the last one is to ensure, say, that the invariant `z > 0`.

The first contribution of our approach is that our data-flow analysis handles, but is not limited to, dependencies (i.e. caller-callee relationships) between methods. More precisely, we aim for testing every pair of method `m1` that defines the value of an attribute `att` and method `m2` that uses `att`'s value. In our approach, it is not necessarily the case that `m1` refers to `m2` as part of its definition, or vice versa. On the other hand, in the context of procedural or object-oriented programming code, the data-flow analysis is completely based upon tracking, for every method, merely within the range of methods upon which it depends. Consequently, existing approaches are incapable of picking up `m1` and `m2`—with no in-between dependency—as a test case of inter-method usage pattern for `att`.

The second contribution of our approach is that for each inter-method usage, captured in the form of a method call sequence, we calculate a precise Boolean constraint, ensuring both the demonstration of the inter-method usage pattern and the maintenance of invariants. This is distinguished from what we traditionally expect from data-flow testing, where each test case merely consists of a pair of line numbers that locates a data usage pattern from the program texts. Our generated test suite reflects the consequences of developers' initial design, for calling those method sequences. Therefore, by examining this test suite, developers may reassess the contractual model and take actions accordingly to fix it. Although this examination, unavoidably, has to be manual, since the requirements are described in some natural language, our approach still has its value of helping developers concentrate on the data-flow aspect.

References

1. Lilian Burdy, Yoonsik Cheon, David Cok, Michael D. Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. An overview of JML tools and applications. *Software Tools for Technology Transfer*, 7(3):212–232, June 2005.
2. Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The spec# programming system: An overview. In *CASSIS 04': Construction and Analysis of Safe, Secure and Interoperable Smart devices*, volume 3362 of *LNCS*, pages 49–69. Springer, 2005.
3. David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing.* John Wiley & Sons, Inc., 2003.
4. Jim Davies, Charles Crichton, Edward Crichton, David Neilson, and Ib Holm Sørensen. Formality, evolution, and model-driven software engineering. *Electronic Notes in Theoretical Computer Science*, 130:39–55, May 2005.
5. J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, 1996.
6. J. M. Spivey. *The Z Notation: a Reference Manual.* Prentice Hall International (UK) Ltd, 2nd edition, 1992.

Threads Are Objects Too

Eric Kerfoot Steve McKeever

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Understanding and employing concurrency correctly is a difficult endeavour for any programmer familiar solely with sequential programs. This paper outlines the CoJava implementation of the Active Object Design Pattern [4] which tackles the challenges of complexity, communication, and correctness in parallel code. Race conditions and deadlock are prevented statically at *compile time*, while *runtime* correctness is achieved through rigorous testing based on runtime specification checking.

CoJava [3] is a core subset of the Java [2] language that uses a form of the JML [5] language to define type specifications. Using type or class annotations, a *threaded* type is defined which presents a concurrent thread of execution as an object. A call to a threaded object's method is translated into a message sent to that object's message queue, which is processed concurrently by a separate thread or process. Any result value produced is returned to the caller, thus method invocation and execution is decoupled.

For example, given a CoJava type called `Counter` with methods `inc()` and `add(int)`, the following demonstrates a threaded instance `c` of the type. Note that the first invocation will not wait for the method to execute, thus the second can be invoked immediately. The effect of these calls is to send two messages to `c` which will be processed in the order received, while the caller need not wait for any response. Thus CoJava characterizes threads as objects and communication as asynchronous method calls, concepts already familiar to the programmer. Data races and deadlock are not prevented by this approach, therefore additional mechanisms must be applied to enforce concurrent correctness properties.

```
/*@ threaded @*/ Counter c = new /*@ threaded @*/ Counter(10);
c.inc(); c.add(5);
```

Race Conditions and Deadlock

Race conditions occur when mutable data shared by multiple threads is concurrently accessed and mutated. CoJava's solution is to disallow the sharing of mutable data using a type-based approach, similar to other active object definitions [1, 7]. An *admissible* type is a primitive type, a threaded type, an immutable object type, or a type implementing the interface `StringSerializable`. Sharing values of these types between thread contexts is safe since they are, respectively: copied bitwise, have their own context, never change state, or can be copied by automatically serializing to strings and then deserializing. Only methods whose arguments and return values have admissible type may be called on threaded receivers.

Deadlock occurs when multiple threads wait indefinitely for an event to occur, which cannot due to this waiting. For example, if threaded object `a` calls a method of threaded object `b` and then waits for a response, `a` cannot respond if `b` were to call any of its methods. Both objects are now waiting indefinitely for each other, and so will any other object attempting to communicate with them. Deadlock is a symptom of circular aliasing relationships.

CoJava tackles the problem by not allowing a client to wait indefinitely for a response. When a method of a threaded receiver is called, the return value is always an instance of `Result` which, like promise objects [6], is a receptacle for the method's actual return value. Consider the example of calling `get()` on object `c`:

```
Result r = c.get(); int i = r.intResult(1000);
```

The argument to `intResult(int)` defines a finite time period, in milliseconds, which the method may wait for the result from the method call to arrive. If it does not arrive in this time frame, `r` indicates a timeout event has occurred. A client therefore cannot wait indefinitely for responses from threaded objects, thus deadlock is translated into a recoverable error.

Specification

JML specifications are composed of predicate expressions (contracts) which define correctness properties for types and methods. Contract expressions must have no side-effects in any thread when evaluated, thus they may only use side-effect free methods. When checking contracts at runtime, the `Result` type cannot be used in contracts since its important methods are not side-effect free. Special code must be generated to allow the use of threaded objects in contracts to provide ostensible side-effect-freeness. Unlike in method bodies, this allows the calling of threaded object methods directly without `Result`.

If a contract evaluates to false, the fact is reported as an exception in the threaded object being checked, rather than the caller. This makes blame assignment more complex, and differs from the sequential case where the caller will be able to catch the exception. The CoJava solution requires the subsystem to catch the exception, report the fact to the error output, and then attempt to continue processing messages for the offending object. There is no good way to cause the exception to be thrown in the calling thread, since the error may occur long after the initial invocation and so would be out of context.

Conclusion

This paper has briefly outlined the CoJava threaded object model of concurrency. CoJava statically guarantees a data race and dynamic deadlock free environment which is compatible with object-oriented specification. The CoJava tool is available at the project website: `http://devel.softeng.ox.ac.uk/cojava`.

References

1. Clarke, D., Wrigstad, T., Östlund, J., and Johnsen, E. B. Minimal ownership for active objects. In *APLAS '08: Proceedings of the 6th Asian Symposium on Programming Languages and Systems* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 139–154.
2. Gosling, J., et al. *The Java Language Specification.* GOTOP Information Inc., 5F, No.7, Lane 50, Sec.3 Nan Kang Road Taipei, Taiwan, 1996.
3. Kerfoot, E., McKeever, S., and Torshizi, F. Deadlock freedom through object ownership. In *IWACO '09: International Workshop on Aliasing, Confinement and Ownership in Object-Oriented Programming* (New York, NY, USA, 2009), ACM, pp. 1–8.
4. Lavender, R. G., and Schmidt, D. C. Active object: an object behavioral pattern for concurrent programming. *Proc.Pattern Languages of Programs,* (1995).
5. Leavens, G. T., Baker, A. L., and Ruby, C. JML: A notation for detailed design. In *Behavioral Specifications of Businesses and Systems*, H. Kilov, B. Rumpe, and I. Simmonds, Eds. Kluwer Academic Publishers, 1999, pp. 175–188.
6. Liskov, B., and Shrira, L. Promises: linguistic support for efficient asynchronous procedure calls in distributed systems. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation* (New York, NY, USA, 1988), ACM, pp. 260–267.
7. Srinivasan, S., and Mycroft, A. Kilim: Isolation-typed actors for java. In *European Conference on Object Oriented Programming ECOOP 2008* (2008).

On The Feasibility of Platform Attestation

John Lyle

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

One of the miracles of the Internet is that it has been successful despite being built with, and still using, software that was never designed to work in a hostile environment. Over the last decade, however, we have been paying for this initial good fortune. Cyber crime has become lucrative and therefore prevalent, as more valuable data are being stored and processed online. Criminals have become more organized, ingenious and, as a result, wealthy[1]. The Heartland Payment System[2] is an alarming example. Over 250 thousand businesses used it to process 100 million credit card transactions every month. Malicious software was installed on the payments system, which then stole potentially tens of millions of card details. Similar incidents at RBS Worldpay and Hannaford Bros., also involved malicious software[2], demonstrating the computer security industry's failure to keep up with the rapid growth and evolution of the Internet.

The ability to assess and evaluate a remote computer would go a long way to avoiding some of these catastrophes. Users would be able to make informed decisions before allowing their data to go online, and companies would be able to use internet services with greater confidence in the results. As well as mitigating existing problems, it would also allow other industries, such as healthcare and pharmaceuticals, to exploit the cost-saving benefits of online services. In response to this need, the Trusted Computing Group (TCG) have introduced technology for establishing trust in a remote platform, based on the concept of *integrity reporting*.

The idea behind integrity reporting is straightforward: a platform can only be trusted if it reports the identity of every piece of software it has run. In the TCG approach, programs are uniquely identified through a cryptographic hash of their executable. A computer's *configuration* refers to the complete list of hashes containing every program that has been run since it was booted. Of course, we cannot trust a platform to report its own configuration honestly, and so a special piece of hardware, a Trusted Platform Module (TPM), is required. The TPM is a chip connected to the CPU, providing a protected memory space so that hashes can be stored immutably. When a TPM-enabled platform boots, each application is hashed and stored before it is allowed to execute, so that even a malicious program will be measured and recorded. The final configuration list can then be signed by a key held in the TPM and reported to the interested party. This process is called *remote attestation*. To ensure that a genuine TPM was used to create the signed list, the key is certified by a trusted authority. In this way, a computer can prove to a challenging party the identity of all software it is running. The challenger can then decide how to proceed based on this information.

However, there are many problems with attestation. Privacy is a consideration, as the challenger must identify every piece of software executed on the attesting platform. This might allow them to discriminate based on their own criteria[3, 4], requiring software from only one vendor, for example. Reporting the exact hash values could also make an attacker's job easier[5], as he or she will be able to quickly identify which known flaws exist. Another issue is that attestation only reports a platform's *execution state* rather than its *security state*[4], which many consider to be the ultimate goal. If it is not clear that one software configuration is necessarily more trustworthy than another, why report it? Perhaps the only way of bridging this gap in semantics is through testing or verification of software, both

time consuming processes. A related issue is that integrity measurement can only assert the identity of software at load time, and says nothing about its runtime state[6, 7]. In-memory attacks (such as exploiting a buffer overflow) will not be reported in an attestation, but will alter the expected behaviour of the machine. Finally, the complexity of managing a whitelist of trustworthy configurations has frequently been cited as a problem. England[7] claims that the 4 million windows drivers (growing at 4000 per day) makes even identifying the software running on a platform a challenge. Other researchers have made similar points[3, 8], citing the frequency of software updates as a major practical problem.

Although the problem of maintaining a whitelist is commonly referred to, few have made any effort to put numbers to it. This is essential, as the more program hashes, the more significant the problem. And if it *is* significant, it has an impact on the feasibility of all trusted computing research. However, the number of hashes will depend on the type and purpose of the platform, and so a context must be chosen for any experiments. It seems likely that the enormous diversity of software on home computers limits the application of attestation there, but an individual web server may be more reasonable. This is a similar scenario to that of the Heartland Payment System incident, and we decided to investigate how feasible it is to interpret the results of an attestation from a web service.

We set up a web service platform and counted the number of hash measurements in its boot process, taking into account two-and-a-half years of software updates and patches. We found that a total of 1414 measurements were required, at a rate of around 35 per month. From our results and observations, we suggest that the use of integrity reporting will be best suited for checking patch levels and simple properties, such as whether an administrator has logged into the machine or not. Although we have only looked at one scenario, we believe that researchers can be much more positive about the use of trusted computing. However, at present it is not suitable for assessing the *trustworthiness* of a remote service, as the software is a rapidly moving target. If this is the overall aim, then more needs to be done to improve on standard trusted computing processes. We have identified new and existing methods for reducing the burden on trusting parties, such as reducing the amount of running software, and using improved isolation mechanisms for separating the *trusted computing base* of a platform from any untrusted components.

References

1. Franklin, J., Paxson, V., Savage, S., Perrig, A.: An inquiry into the nature and causes of the wealth of internet miscreants. In: CCS '07, ACM (Nov 2007)
2. Krebs, B.: Payment Processor Breach May Be Largest Ever. The Washington Post Website (Jan 2009)
3. Sadeghi, A.R., Stüble, C.: Property-based Attestation for Computing Platforms: Caring About Properties, Not Mechanisms. In: NSPW '04, ACM Press (2004) 67–77
4. Poritz, J.A.: Trust[ed | in] Computing, Signed Code and the Heat Death of the Internet. In: SAC '06, ACM Press (2006) 1855–1859
5. Kühn, U., Selhorst, M., Stüble, C.: Realizing property-based attestation and sealing with commonly available hard- and software. In: STC '07, ACM Press (Nov 2007) 50–57
6. Schellekens, D., Wyseur, B., Preneel, B.: Remote Attestation on Legacy Operating Systems With Trusted Platform Modules. In: REM '07. Volume 197. (2008) 59–72
7. England, P.: Practical Techniques for Operating System Attestation. In: TRUST '08. Volume 4968 of LNCS., Springer (Mar 2008) 1–13
8. Haldar, V., Chandra, D., Franz, M.: Semantic Remote Attestation - Virtual Machine Directed Approach to Trusted Computing. In: Virtual Machine Research and Technology Symposium, USENIX (2004) 29–41

Towards Architectural Trust Properties
Establishing Architectural Elements and Dependencies

Cornelius Namiluko

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Trusted computing enables the secure reporting of a platform's integrity through a process called attestation, in which one entity (human, machine or process) can determine whether a given platform is in an acceptable configuration. Suppose an entity was presented with two trustable platforms with identical configurations, how would they choose one that will provide a greater guarantee of confidentiality and integrity? This question becomes even more challenging when you start considering complex systems such as grid or cloud computing that may use a collection of cooperating platforms for a single task, e.g. executing a job. To answer this question, we need to identify the attributes that can differentiate two systems, we call these *trust properties* and define them as *structural, environmental or operational attributes of a system that enhance the degree to which the system can behave consistent to its specifications and further allows other entities to reason about the overall security state of a system.* We identify the source of the attributes to be a system's architecture combined with attributes of its runtime environment. We are therefore interested in understanding *the trust properties of an architecture and how these properties influence trustworthiness of systems based on that architecture.*

Fielding [3] defines a system's architecture as an abstraction of a its runtime behaviour while Leavens and Sitarama [2] point out that a system's architecture plays an important role in the trustworthiness of the system. Furthermore, an architecture determines the placement of elements and their interaction and defines the constraints on how the elements interact and how they can be combined [3, 5]. Therefore, identifying and understanding the properties of architectural elements is crutial in predicting properties of the resulting architecture. It is for this reason that we approach the challenge of identifying an architectures trust properties by first identifying the building blocks of trusted systems and the relationships among them.

In our earlier work [4], we identified key architectural elements for trusted grid architecture. In this work, we identify a different level of abstraction in order to capture general purpose elements - not just those restricted to grid or service-oriented architectures. These elements exist in an architecture as components (processing units), connector (communicating unit) or data (input/output to components) [1] and are summarised in Table 1.

We analysed several trusted system architectures to discover their use of the identified elements and to deduce the dependencies that exist among the elements. Some of the elements operate independently, while others depend implicitly or explicitly on other elements. We identified three types of dependencies as follows:

Functional dependency - represents a relationship where one element uses another to achieve its functional goals. For example, an element that uses random numbers functionally depends on an element that generates random numbers. This relationship has an implication that sound trustworthiness verification for a particular requirement must consider all the elements that are depended upon to satisfy the requirement. In addition, a dependent element can only satisfy its functional goals if all elements it depends on are present.

Operational dependency - exists where one element either provides initialisation data essential for the operation of another element or ensures that another continue operating in the expected manner. For example, if an element requires a configuration switch to indicate

Element	*Description*
Hardware-based TPM	A trusted platform module implemented as a hardware chip
Software-based TPM	A software component that meets the TPM specification
Access Control	A component that makes decisions on access to a given object
Integrity Measurement Service	A component that generates integrity measurements for the platform
Whitelist	A data element containing known-good configuration values
Configuration Token	A data element representing the state of a platform
Attestation Token	A credential containing identity information and a TPM key
Isolation Service	Provides compartment for different guests running on the same host
Attestation Service	Enables the secure reporting and verification of platform integrity
Reporting Service	Reports non-security-sensitive system attributes
Policy Decision Point (PDP)	Uses algorithms to determine what policy to enforce
Policy Enforcement Engine	Responsible for enforcing policy decisions made by the PDP
Storage Service	Provides non-volatile storage facilities
Cryptographic System	Provides software based cryptographic functions
Trusted Path	An interface to human that guarantees integrity and confidentiality
Trusted Channel	A connector that guarantees integrity and confidentiality

Table 1. Description of Trusted Architecture Elements

whether to encrypt data or not, then an element operationally depends on another element to provide that switch. Operational dependency differs from functional dependency in that the dependent may or may not be aware about the existence of the elements depended on. **Trustworthiness dependency** - occurs between elements that may exist independently, but where one element's existence improves the overall trustworthiness of the operation of another element. To identify this relationship, each element was examined for possible threats, and any element that is not explicitly invoked but helps to mitigate or reduce the risk creates a trustworthiness dependency with the element being examined.

The identification of elements and their relationships will serve as a foundation for deducing properties of an architecture because it is the properties of constituent elements and the constraints on their interaction that determine the overall properties of an architecture.

References

1. Stephen T. Albin. *The art of software architecture: design methods and techniques*, chapter Introduction to Software Architecture, pages 9–11. Wiley Publishing Inc., 2003.
2. Cliff B. Jones Denis Besnard, Cristina Gacek. *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, chapter Architectural description of dependable software systems, pages 127–142. Cambridge University Press, 2000.
3. Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California Irvine, 2000.
4. Cornelius Namiluko Jun Ho Huh, John Lyle and Andrew Martin. Application whitelists in virtual organisations. Submitted to the Future Generation Computer Science Journal, August 2009.
5. Mary Shaw. The coming-of-age of software architecture research. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, page 656, Washington, DC, USA, 2001. IEEE Computer Society.

The Complexity of Divide and Conquer

Dmitri Akatov and Georg Gottlob

Oxford University Computing Laboratory

1 Introduction

A hypergraph is a generalization of a graph, allowing (hyper)-edges to contain any positive number of vertices, rather than exactly two for the case of graphs. Hypergraphs are a useful abstraction tool in database theory and data exchange, in particular for query containment testing and core computations. Many such problems are NP-hard in general, however become tractable, if the underlying hypergraph of the query is acyclic [4]. Unfortunately, acyclic hypergraphs comprise only a small subclass of all hypergraphs.

Hypergraph decompositions usually transform a hypergraph into an acyclic structure (usually a tree) with individual nodes of this structure in turn labelled by smaller structures which do not have to be acyclic. As long as we can restrict the size or "cyclicity" (usually termed *width*) of these smaller structures to a fixed number, most NP-hard problems involving queries with the decomposed hypergraph as their underlying hypergraph become tractable.

Gottlob et al. [3] give a good overview of various decomposition methods and their relationships to each other. In particular they consider *tree decompositions* and *hypertree decompositions*. The former is a well-known decomposition method for graphs originally studied in conjunction with graph minors [7], but also applicable to hypergraphs, with the associated notion of *tree width*. The latter is a strict generalization of the former, with the associated notion of *hypertree width*, and was specifically introduced for hypergraphs. A particularly nice property of queries with bounded hypertree width is that the *Boolean Conjunctive Query* problem is not only tractable, but also complete for the complexity class LOGCFL [5]. Also, the problem of recognizing hypergraphs of bounded hypertree width is in LOGCFL. This complexity class lies very low within the NC hierarchy (and hence within P) inbetween NC^1 and AC^1 and hence is highly parallelizable.

2 A new type of hypergraph decomposition

The trees associated with tree and hypertree decompositions generally do not possess any special structure and in the worst case have depth linear in the size of the hypergraph and a branching factor of 1. This, however, has negative effects on the parallelization of the BCQ problem. In fact, parallelization works best if a problem splits into smaller subproblems of approximately the same size, in the style of many well-known divide-and-conquer type algorithms. For hypergraph decompositions this corresponds to the underlying trees being approximately balanced.

We define a new type of decomposition, which possesses the above property, called a *balanced cut decomposition*. The Boolean Conjunctive Query (BCQ) problem for the class of queries of bounded *balanced cut width* (BCW) is hard for LOGCFL. The smallest complexity class containing the BCQ problem for queries of bounded BCW, which we could find in the literature, is the class NTiSp(poly, $\log^2$), the space-bounded subclass of NP only allowing square-logarithmic usage of the worktape [6]. Since the latter class is suspected not to be

contained in P unless P=NP, the existance of a polynomial-time algorithm for this problem remains an open question. However, we present a quasi-polynomial algorithm (running in time $O(n^{\log n})$, which moreover can be easily parallelized.

3 A new complexity class

We could not establish a hardness proof for NTiSp(poly,$\log^2$), and the way the non-deterministic algorithm works (the tape is accessed in a very specific manner), strongly suggests that the above problem properly lies inbetween LOGCFL and NTiSp(poly,$\log^2$).

We define a new computational model: *(Non)deterministic Auxiliary Transparent Pushdown Automata*. As the name suggests, this is a direct generalization of (Non)deterministic Auxiliary Pushdown Automata: In our model the auxiliary stack is *transparent*. This means that any element of the stack can be read at any time, however only the top-most element can be popped or pushed onto. We show that the BCQ problem for queries of bounded BCW is complete for the class of NAuxTPDAs running in polynomial time, using space $O(\log n)$ and maximal stack height $O(\log^2 n)$. We give this class the codename CAESAR, suggesting the type of algorithms it can run. Finally, we show that recognizing hypergraphs of bounded BCW is also feasible in CAESAR.

4 Future Work

Future work includes a better analysis of relations between (N)AuxTPDAs and other models of computation, in particular (N)AuxPDAs, Alternating and (Non)deterministic Turing Machines and various resource bounded variations of these. This would allow for a better placement of CAESAR within the complexity hierarchy relating it to the various classes presented in [8], [2] and [1]. Another direction of work is to establish whether the problem of recognizing hypergraphs of bounded BCW is complete for CAESAR or whether it belongs to a lower complexity class. Finally, an important aspect of future work is the creation and testing of computer programs implementing the (parallel) algorithms presented in this paper.

References

1. Beigel, R., and Fu, B. Molecular computing, bounded nondeterminism, and efficient recursion. In *In Proceedings of the 24th International Colloquium on Automata, Languages, and Programming* (1998), vol. 25, pp. 816–826.
2. Goldsmith, J., Levy, M. A., and Mundhenk, M. Limited nondeterminism, 1996.
3. Gottlob, G., Leone, N., and Scarcello, F. A comparison of structural csp decomposition methods. *Artificial Intelligence 124*, 2 (December 2000), 243–282.
4. Gottlob, G., Leone, N., and Scarcello, F. The complexity of acyclic conjunctive queries. *J. ACM 48*, 3 (May 2001), 431–498.
5. Gottlob, G., Leone, N., and Scarcello, F. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences 64*, 3 (May 2002), 579–627.
6. Monien, B., and Sudborough, I. H. Bandwidth constrained np-complete problems. *Theoretical Computer Science 41* (1985), 141–167.
7. Robertson, N., and Seymour, P. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms 7*, 3 (September 1986), 309–322.
8. Ruzzo, W. L. Tree-size bounded alternation. *Journal of Computer and System Sciences 21*, 2 (October 1980), 218–235.

Instance-Based Hyper-Tableaux for Coherent Logic

Evgenij Thorstensen

Oxford University Computing Laboratory

Automated theorem proving (ATP) is a subfield of logic in computer science that deals with the problem of efficiently deciding the validity or satisfiability of formulae in some logical language. In our case, the logical language is *coherent* or *geometric* logic.

Coherent logic (CL) [5,3] is a syntactic subset of a first-order logic without function symbols where every formula has the form

$$\forall \vec{x}.(A_1 \wedge \cdots \wedge A_n \rightarrow \exists \vec{u_1}.B_1 \vee \cdots \vee \exists \vec{u_j}.B_m)$$

where the A_i are atoms, while the B_i are conjunctions of atoms. (The notation $\vec{x}$ refers to a finite list of variables.) In other words, we consider universally quantified implications from conjunctions of atoms to existentially quantified disjunctions of conjunctions of atoms.

CL is of interest because, while having the expressive power of full first-order logic [5,6], it is possible to do efficient proof search in it without skolemization. This makes CL useful in proof assistants, as skolemization of larger problems can significantly alter their structure. This makes it difficult to apply the intuitions you might have about the problem to the skolemized version your proof assistant is working on. Another advantage we gain from the lack of skolemization is that the *proof* produced for the problem translated to CL can easily be translated to a proof for the original problem. A typical example is the induction step of Newman's Lemma, where a CL formalization gives a very short proof (see [4] for details, and [5] for the translation from first-order logic to CL).

Instance-based methods (IMs) [2] are a family of first-order methods for ATP that work by generating instances of formulae to be checked, instead of combining existing formulae to produce new ones (as done in e.g. resolution). These methods work well on a fragment of first-order logic called the Bernays-Schönfinkel fragment. As satisfiability of CL theories, i.e. sets of CL formulae, falls under this fragment, it makes sense to adapt IMs to CL in a way that avoids skolemization.

Our research [7] presents an instance-based calculus for coherent logic based on the Next-Generation Hyper-Tableau calculus (NG) [1]. The NG calculus has two rules, Ext and Link. The Ext rule is used to extend the tableau with clauses from the set of *working clauses*, while the Link rule is used to add instances of clauses already on the tableau to this set. The clauses generated by Link can in turn give rise to new applications of Ext.

The presented calculus modifies NG to work on clauses with two types of free variables, corresponding to universally and existentially quantified variables in CL formulae. To deal with such clauses, both rules have to be modified. The Ext rule is modified so that existential variables are replaced by fresh constants, making them invariant under later substitutions.

The Link rule must be modified more severely. Instead of generating instances of clauses on the tableau, Link must generate instances in the set of working clauses, taking care not to instantiate existential variables. The move from instantiating clauses on the tableau to instantiating clauses in the clause set is necessary to preserve soundness, as the fresh constants introduced by the Ext rule must be fresh for every instance. Consider a simple example: from the formula $\forall x \exists y Pxy$, which in our clausal form becomes $Px\dot{y}$, it is sound to generate the clause Pat, as t is a fresh constant — so if the formula is true, then it is true when x is instantiated to a, and we can interpret t as the object that makes it true in this

case. However, we may not generate the above and Pbt, as now we must interpret t as the same element both for a and for b — but the two clauses might not be true for the same element in both cases. The sound inference would have been Pbt' with t' fresh. To avoid this problem, our calculus will use the Link rule to generate $Pb\dot{y}$ (as clauses in the set we are working on never change in their existential variables), and when it is time to put this clause on the tableau, the Ext rule will replace $\dot{y}$ by a fresh constant.

These modifications suffice for soundness and completeness, but the changes to the Link rule mean that we no longer use the tableau as a guide for finding new Link rule applications. As the set of working clauses is usually much larger than the set of clauses on the tableau, this may destroy any benefit we gain by using CL in the first place.

We address this by proving that if an Ext rule application fails in a certain way, a Link rule application can be performed instead, with clauses that were previously used in an Ext application (something we keep track of while constructing the tableau). We then prove that the calculus stays complete if no other applications of Link are allowed, trading a little space for a lot of time.

To summarize, we show that CL admits an instance-based calculus that, while keeping the usual benefits of using one, removes the need for skolemization.

References

1. Baumgartner, P. Hyper tableau — the next generation. In *Automated Reasoning with Analytic Tableaux and Related Methods* (1998), H. de Swart, Ed., vol. 1397 of *LNCS*, Springer, pp. 60–76.
2. Baumgartner, P., and Thorstensen, E. Instance based methods — a brief overview. Accepted for publication in KI. `http://users.rsise.anu.edu.au/~baumgart/publications/IMoverview.pdf`, 2009.
3. Bezem, M. On the undecidability of coherent logic. In *Processes, Terms and Cycles: Steps on the Road to Infinity* (2005), A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R. de Vrijer, Eds., vol. 3838 of *LNCS*, Springer, pp. 6–13.
4. Bezem, M., and Coquand, T. Newman's lemma — a case study in proof automation and geometric logic, Logic in computer science column. *Bulletin of the EATCS 79* (2003), 86–100.
5. Bezem, M., and Coquand, T. Automating coherent logic. In *Logic for Programming, Artificial Intelligence, and Reasoning* (2005), vol. 3835 of *LNCS*, Springer, pp. 246–260.
6. Norgaard, J. M. An automated translation of first-order logic formulas to coherent theories. Master's thesis, University of Bergen, 2007.
7. Thorstensen, E. Instance-based hyper-tableaux for coherent logic. Master's thesis, University of Oslo, 2009. `http://www.duo.uio.no/sok/work.html?WORKID=91821&lang=en`.

Author Index